Klaus Kohl

Handbuch
zum
KK-FORTH

Version 1.2

- Einführung in KK-FORTH

- Beschreibung des KK-FORTH

- Ausführliches Glossar

Wichtige Hinweise:

Alle im Handbuch angegebenen Informationen wurden mit größter Sorgfalt zusammengestellt. Trotzdem kann der Autor keine Gewähr dafür übernehmen, daß die Angaben korrekt und die verwendeten Warenbezeichnungen, Warenzeichen und Programmlistings frei von Schutzrechten Dritter sind. Da sich Fehler nie vollständig vermeiden lassen, ist der Autor für Hinweise jederzeit dankbar.

Lizenznehmer von KKF dürfen eigene Programme, die mit KKF kompiliert wurden und der Interpreter-/Kompilerteil dem Anwender nicht mehr zugängig sind, ohne Lizenzgebühren verwenden, weitergeben oder verkaufen.

Copyright: Ing. Büro
Klaus Kohl
Pestalozzistr. 69
D-W8905 Mering 1
Tel. 08233/30524

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Vorwort

Beim KK-FORTH oder kurz KKF handelt es sich um eine einheitliche FORTH-Version für diverse Prozessoren und Betriebssysteme. Es basiert auf dem FORTH83-Standard, hat aber viele Erweiterungen. Beispielsweise sind einige Befehle des geplanten ANSI-Standard implementiert. Darüber hinaus wurden viele einfache, aber wirkungsvolle Konzepte aus verschiedenen anderen FORTH-Versionen übernommen und erweitert.

Trotzdem bleibt es durch seine grundlegende 16Bit-Struktur und dem abgerundeten Befehlsumfang eine Version, die sich gut zum Erlernen der Programmiersprache FORTH eignet. Deshalb enthält dieses Handbuch auch ein Kapitel zur Einführung in FORTH. Zusätzlich werden mit dem KKF viele Sourcen wie Bildschirmeditor, Assembler, Disassembler oder Beispiele zur Ansteuerung der vorhandenen Hardware mitgeliefert.

Bei den KKF-Versionen für Einplatinencomputer (z.B. für CP/M-Systeme mit serieller Schnittstelle oder den EMUF-Versionen) übernimmt ein Terminalprogramm auf dem PC die Verwaltung der Files und das Editieren der Programme.

1.2 Entwicklungsgeschichte des KKF

Der Autor Klaus Kohl befaßt sich schon seit 6 Jahren mit der FORTH-Programmierung in der Meßtechnik. Beginnend mit einer FORTH-Version für den Harris-Prozessor RTX-2000 im Jahre 1989 (wird im RTX-Board der FORTH-Gesellschaft e.V. und im RTX-EMUF verwendet) entstanden immer wieder spezielle FORTH-Versionen für Singleboard-Prozessoren. Das letzte Projekt war die Erstellung einer FORTH-Version für die ROM-Maske des Zilog-Super8-Prozessors (ist seit April 1991 beim Autor verfügbar).

Jedoch haben alle Realisierung den Nachteil, daß sie zwar sehr an den FORTH83-Standard angelehnt sind, aber trotzdem einen abweichenden Befehlsumfang haben. Deshalb bleibt die Übertragung von Programmen und das Umstellen auf andere Entwicklungsumgebungen zwischen den einzelnen Versionen schwierig.

Das mit der Beschreibung ausgelieferte KK-FORTH soll mit diesem Nachteil aufräumen und ein einheitliches FORTH-System für möglichst viele Prozessoren bereitstellen. Da dabei immer alle Variablen und veränderlichen Daten in einen eigenen Bereich übertragen werden, können auch ROM-fähige KKF-Versionen erstellt werden.

Schon während der Konzeptionierung werden dabei unterschiedliche Systeme berücksichtigt. Beginnend mit IBM-PC, PC-, HD84C015- und RTX2000-EMUF werden von Anfang an mehrere Prozessorgruppen unterstützt. Bei genügendem Interesse werden dann weitere Versionen für 68xxx, 68HC11, 650x oder sogar Transputer realisiert und ins Vertriebsprogramm aufgenommen.

1.3 Features des KKF

Das KK-FORTH hat folgende Besonderheiten:

- Weitgehende Kompatibilität mit dem FORTH83-Standard
- Viele Befehlserweiterung aus volksFORTH und ANSI-Vorschlag
- Durch viele Assemblerbefehle sehr schnell
- Durch Auslagerung der Variablen, DEFER-Wörter und Vokabulare ROM-Fähig
- Das Fileinterface und die Ein-/Ausgabe sind vektorisiert
- Flexible Fehlerbehandlung (umleitbar) mit Fehlermeldungen als Text
- Puffer für die letzten 8 Eingabezeilen (Größe veränderbar)
- Viele Systemwörter können durch eigene Befehle ersetzt werden
- Autostart-Applikationen sind möglich
- Selbst EMUF's können das Fileinterface verwenden (über Terminal)
- Viele Tools wie Assembler, Disassembler und Debugger werden mitgeliefert
- System für Multitasker vorbereitet

Es können noch viele Pluspunkte erwähnt werden. Jedoch das Beste am KK-FORTH ist sein niedriger Preis und die Verfügbarkeit auf verschiedenen Prozessoren. Sollte einmal ein FORTH-Programm auf dem Z80 zu langsam sein, so kann man durch Verwendung eines schnelleren Prozessors und der zugehörigen KKF-Version ohne Änderung der Arbeitsumgebung sofort weitermachen. Bei Übertragung von Programmen zwischen KKF-Versionen sind oft nur die Assembler-Routinen und die Hardware-Ansteuerung (z.B. andere Adressen) anzupassen.

1.4 Installation des KKF

Beim Kauf des KK-FORTH erhalten Sie:

- Ausführliches Handbuch zum KK-FORTH (gleich für alle Versionen)
- Zusatzbeschreibung zur gewünschten KKF-Version
- Diskette(n) mit KK-FORTH, diversen Zusatzfiles und -programmen

Falls Sie das Handbuch schon besitzen, aber eine weitere KKF-Version bestellen, so können Sie zu einem günstigeren Preis nur Zusatzbeschreibung und Diskette erhalten. Auf Wunsch können die für EMUF benötigten EPROM's auch direkt vom Entwickler bezogen werden.

Keines der Programme oder Zusatzfiles sind geschützt. Deshalb können sie sowohl auf andere Disketten als auch auf die Festplatte kopiert werden. Es ist zu empfehlen, vor dem ersten Start des Programmes ein Schreibschutz auf der Diskette anzubringen und (mit DISKCOPY) ein vollständiges Backup zu erstellen.

Bei den RAM-Versionen des KKF (z.B. für PC) kann das KKF direkt von Diskette oder Harddisk gestartet werden. Da im FORTH Befehle zur Bearbeitung von Files des aktuellen Verzeichnisses vorhanden sind, sollte für das KK-FORTH ein eigenes Unterverzeichnis angelegt werden.

Bei EPROM-Versionen muß zuerst das auf Diskette mitgelieferte Bitimage in ein (beim RTX in zwei) EPROM gebrannt und an Stelle des Monitors in die Platine eingesteckt werden. Das ebenfalls auf der Diskette mitgelieferte Terminalprogramm sollte dann zusammen mit den Zusatzfiles auf eine eigene Diskette oder in ein Unterverzeichnis der Harddisk kopiert werden. Je nach Hardware-Konfiguration muß dann das Terminalprogramm gestartet (Baudrate und Port einstellen), die serielle Schnittstelle des PC's mit der Platine verbunden und dann die Versorgungsspannung eingeschaltet werden. Dabei ist darauf zu Achten, daß die Datenleitungen (Pin 2 und 3) gekreuzt und (bei EMUF's ohne Handshake) am PC die Pin's 4 und 5 miteinander verbunden werden müssen. Falls die Baudrate schon vorgegeben ist, meldet sich das KK-FORTH sofort nach dem Einschalten. Beim KKF_RTX wird die Baudrate beim Empfang des Return-Zeichens (dazu ENTER drücken) erkannt und erst danach die Einschaltmeldung geschickt.

Kapitel 2

Einführung

2.1 Allgemeines über FORTH

2.1.1 Was ist FORTH ?

* ein Programm
* eine Programmiersprache
* ein Interpreter
* ein Kompiler
* ein strukturierter Assembler
* eine Programmierumgebung
* eine Philosophie

Es fällt schwer, FORTH zu beschreiben, denn alle obenstehenden Punkte treffen zu. Ein FORTH-Kern hat einen Umfang von 8 bis 20 KBytes und enthält alle Grundbefehle und die Betriebssystemaufrufe. Auch die Ein-/Ausgabe und die Disketten-Schnittstelle sind darin schon enthalten.

FORTH weicht bei der Programmentwicklung von den sonst üblichen Programmiersprachen ab. In BASIC oder PASCAL werden vordefinierte Befehle aneinandergereiht. In FORTH werden statt dessen neue Befehle definiert, wobei diese neuen Befehle entweder auf schon vorhandene Wörter aufbauen oder in Maschinensprache geschrieben werden. Am Ende einer Programmentwicklung startet die Eingabe eines Befehls das gesamte Programm. Durch diese modulare Programmierung kann während der Erstellung des Programmes jeder Befehl sofort nach seiner Definition getestet werden.

Alle Möglichkeiten eines Computers können durch FORTH voll genutzt werden, da es keine Einschränkungen bei der Programmierung gibt. Einziger Nachteil dieser Fähigkeit ist, daß durch Programmfehler der Computer auch abstürzen kann. Eingebauten Fehlerkontrollen und die gute Austestbarkeit von einzelnen Befehlen reduzieren jedoch dessen Wahrscheinlichkeit.

2.1.2 Geschichtliches

FORTH ist nichts Neues, sondern wurde von Charles H. Moore am National Radio Astronomy Observatory (Arizona) in den 60er-Jahren entwickelt und stand 1971 als komplette Implementierung zur Steuerung des 11-Meter Radioteleskops zur Verfügung. Danach nahmen interessierte Studenten und Programmierer die Ideen auf und übertrugen diese "Programmiersprache" auf andere Computer. Der erste Standard entstand aber durch europäische FORTH-Benutzer und nannte sich nach dem Jahr der Erstellung FORTH77-Standard. Überrascht über die große Begeisterung gründete Charles Moore eine Firma (FORTH Inc.), die sich mit der Anpassung und dem Vertrieb von FORTH befaßt und brachte im gleichen Jahr noch das micro-FORTH heraus.

Dem entgültigen Durchbruch erhielt FORTH 1978 durch die Gründung der FORTH INTEREST GROUP (abgekürzt F.I.G.), einer Vereinigung von FORTH-Programmierern in aller Welt. F.I.G. vereinigte die bisher verwendeten FORTH-Versionen und definierte daraus einen eigenen Standard. Dieser FIG-Standard wurde dann auf den unterschiedlichsten Prozessoren portiert und zu Selbstkosten in Form von Listing oder Programmen vertrieben.

FORTH ist eine lebendige Programmiersprache. Deshalb dauerte es nicht lange, bis sich der FIG-Standard zu den unterschiedlichsten Versionen weiterentwickelte. Es entstand die Notwendigkeit, weitere Standards durch ein Gremium erstellen zu lassen. Diese unter dem Namen FORTH Standards Team bekannte Gruppe veröffentlichte die kaum bekannten FORTH78- und FORTH79-Standards. Bis auf wenige Realisierungen (z.B. MVP-FORTH) verzeichneten diese Standards jedoch keine Erfolge.

Erst 1983 wurde ein im vollen Umfang akzeptierter Vorschlag herausgegeben. Dieser FORTH83-Standard wurde sofort in Form von F83, der Version für den IBM-PC und damit kompatiblen Rechnern, der Allgemeinheit zur Verfügung gestellt. Fast alle momentan erhältlichen FORTH-Versionen sowohl im kommerziellen als auch im PD-Bereich halten sich an diesen Standard.

Jedoch ist das noch lange nicht das Ende der FORTH-Entwicklung. Es wurde ein technisches Kommitee zum Entwurf eines ANSI-FORTH zusammengestellt (ANS X3J14), der im November 1987 mit seinen technischen Beratungen begann. Momentan (Stand April 1991) steht nach 14 Sitzungen das sogenannte BASIS15-Dokument als Diskusionsgrundlage zur Verfügung. Wie es momentan aussieht, wird noch in diesem Jahr ein dpANS-Vorschlag, also ein vorläufiger Entwurf des ANSI-FORTH verabschiedet, der dann ein Jahr lang der öffentlichen Kritik ausgesetzt sein wird.

Inzwischen gibt es kaum einen Computer, für den nicht FORTH in irgendeiner Form erhältlich ist. Meistens ist FORTH sogar das erste Programm, das neben dem Betriebssystem überhaubt auf einem neuen Computer läuft. Einem laufenden Programm sieht man nicht an, daß es in FORTH geschrieben ist. Am häufigsten ist es in Steuerungen, Datenerfassung und -auswertung anzutreffen. Auch Kompiler, Datenbank- und Kalkulationsprogramme bauen oft auf FORTH auf. Es gibt auch einige Computernetzwerke zur Verwaltung ganzer Flughäfen, bei denen FORTH die Grundlage bilden.

Neben der Definition eines Standard und Vertrieb der verschiedensten FORTH-Versionen macht es sich F.I.G. zur Aufgabe, die Kommunikation zwischen FORTH-Anwendern zu fördern. Eine Möglichkeit dazu bietet die Vereinszeitschrift "FORTH DIMENSION". Weltweit tauschen Ableger der FORTH INTEREST GROUP regelmäßig bei Tagungen Informationen aus.

In Deutschland heißt dieser Ableger FORTH-Gesellschaft e.V. . Dieser in Hamburg gegründete Verein mit momentan über 400 Mitgliedern hat auch eine eigene Zeitschrift, der "VIERTEN DIMENSION". Sie wird viermal im Jahr an Mitglieder geschickt. Aktive Mitglieder treffen sich regelmäßig in den einzelnen lokalen Gruppen, die sich mit Realisierung von bestimmten Projekten oder mit Austausch von Informationen befaßen. Jedes Jahr findet dann ein Vereinstreffen statt, bei dem oft auch Firmen ihre Neuerungen und Anwender ihre Realisierungsbeispiele vorstellen. Seit diesem Jahr steht auch eine Mailbox mit vielen Informationen aus der FORTH-Welt zur Verfügung.

FORTH-Gesellschaft e.V.
Postfach 1110
W-8044 Unterschleißheim
Tel.: 089/3173784

FORTH-Mailbox (Jens Wilke)
300-2400 Baud, 8, N, 1
Tel.: 089/8714548

2.1.3 Welche Vorteile hat FORTH?

FORTH-Programme haben geringe Entwicklungskosten

* kurzer Entwicklungszeit
* guten Testmöglichkeiten
* sehr kurze Zykluszeiten
* fertige Befehle können als "Black Box" verwendet werden

FORTH ist einer der Programmiersprachen, bei der schon mit wenigen Befehlszeilen neue Hardware oder neue Betriebssystem-Zusätze getestet werden können. Wegen seines Interpreter-Charakter kann dabei interaktiv gearbeitet werden. Falls dabei ein Fehler festgestellt wird, so kann durch einen Befehl der integrierte Editor aufgerufen und das Programm geändert werden. Da im FORTH normalerweise nur die letzten Befehle geändert werden, lassen sich Zykluszeiten im Sekundenbereich erreichen. Nachdem für eine neue Hardware die entsprechenden Befehle definiert sind, können sie übernommen werden, ohne daß man sich mit dem "Inneren" dieser Wörter nochmal befasen muß.

FORTH fordert nur niedriger Wartungsaufwand

* einfaches Anpassen an neue Schnittstellen
* einfache Einbindung von Erweiterungen
* gute Lokalisierbarkeiten von Fehlern

Viele Firmen verwenden alte (und meist langsame) Computer, nur um nicht in die Verlegenheit zu kommen, alte Programme an neue Schnittstellen anzupassen. FORTH dagegen erleichtert das Ersetzen von veralteten Schnittstellen, da nur die wenigen Befehle zu dessen Ansteuerung überarbeitet werden müssen. Das restliche Programm kann dann ohne Änderung kompiliert werden. Erweiterungen können oft als einfache Ergänzung in schon bestehende Programme eingefügt werden.

Ein FORTH-Programm ist wie eine Pyramide aufgebaut. Ein einzelner Befehl ist aus mehreren anderen Befehle aufgebaut. Diese bestehen wiederum aus anderen Befehlen, welche sich dann auf den Grundwortschatz berufen. Da ein Befehl eine genau festgelegte Aufgabe hat, kann oft schon aus der Art des Fehlers auf die Fehlerquelle geschlossen werden. Das weitere Suchen des Fehlers wird dann durch Single-Step-Programmlauf, interaktiver Testbarkeit einzelner Befehle oder Abfrage von Prgrammdaten unterstützt.

FORTH ist interaktiv

* sofortiges Austesten neuer Befehle
* einfaches Umschalten zwischen Interpreter u. Kompiler
* direkte Kontrolle aller Computer-Resourcen
* direkter Zugriff auf gesamten Speicher u. alle Schnittstellen
* Disk-Files als virtueller Speicher zugängig

FORTH unterscheidet nicht, ob man Befehle von der Tastatur direkt eingibt oder ob sie von der Diskette geladen werden. Dadurch kann fast alles vorher getestet werden. Auch die Schnittstellen, der Speicher und das Diskettenlaufwerk ist direkt zugreifbar. Die beiden Grenzen für die Programmierung in FORTH sind die Fähigkeit des Computers und die Fantasie des Programmierers.

FORTH-Programme laufen mit hoher Geschwindigkeit

In vielen Anwendungen ersetzt FORTH sogar Assemblerprogramme. Es sind z.B. mit einem Z80-Prozessor ohne weiteres Frequenzen von mehreren Kilohertz bei Portausgaben möglich. Wenn man dann entsprechende Prozessoren wie NC4000, RTX-2000/1A, FRP-1600 oder SC32 einsetzt, so sind auch Frequenzen bis zu mehreren Megahertz keine Besonderheit.

Einfaches Einbinden von Assembler-Teile in FORTH

Sind bei Standard-Prozessoren die FORTH-Programme zu langsam, so können die zeitkritischen Befehle ohne weiteres in Assembler geschrieben werden und trotzdem wie FORTH-Befehle aufgerufen und getestet werden. Dazu verwendet man einen ebenfalls in FORTH geschriebenen Inline-Assembler. Es handelt sich dabei um einen Singlepaß-Assembler, bei dem weiterhin alle Variablen, Konstanten, Datenfelder und Befehle des FORTH-Programmes verfügbar sind. Dazu wird (zum Leid vieler Assemblerprogrammierer) die Reihenfolge der Eingabe so umgestellt, daß zuerst die verwendeten Register und Adressierungsarten angegeben werden und dann erst der entsprechende Opcode folgt.

FORTH hat kompakten Programmcode

* kurze Befehlsheader von 6 .. 36 Bytes
* indirekt verknüpfter Code (nur Zeiger erforderlich)
* geringer Speicherbedarf von Variablen u. Konstanten
* nur einmaliges Definieren eines Befehls

Damit der Interpreter/Kompiler des FORTH-System die Befehle findet, werden Befehlsnamen mit Verkettungszeiger und der sogenannten Codefeldadresse gespeichert. Da ein FORTH-Befehl zwischen 1 und 31 Zeichen lang sein darf, ergibt sich ein Platzbedarf von 6 bis 36 Bytes pro Befehlseintrag. Dazu kommen dann noch die entsprechenden Programmteile oder Datenfelder.

FORTH ist intern meistens ein indirekt verketteter Programmcode. Deshalb wird beim Kompilieren eines FORTH-Programmes nur ein Zeiger auf die Codefeldadresse des auszuführenden Befehls abgelegt. Ein Zeiger benötigt dabei nur zwei Bytes. Bei Konstanten und Variable werden insgesamt nur zwei zusätzliche Bytes für den Konstantenwert oder für den Speicher für die Variable benötigt.

Wie effizient die Kodierung eines FORTH-Programmes ist, sieht man auch an dem KKF-Kern. In 16KByte stehen neben den nicht direkt sichtbaren Grund- und Initialisierungsroutinen über 450 Befehle zur Verfügung. Damit benötigt ein Befehl im Durchschnitt weniger als 37 Bytes. Da eigene Programme im Normalfall wesentlich kleiner sind, reichen die 64KByte Programmspeicher für die meisten Applikationen.

FORTH ist strukturiert

* modularer Programmaufbau
* schachtelbare Kontrollstrukturen
* verzicht auf GOTO's

Schon die Festlegung, daß komplexe Befehle auf einfache Befehle aufbauen, unterstützt die Strukturierung. In den einzelnen Wörtern sind darüber hinaus noch einige Programmstrukturen wie Schleifen, Fallunterscheidungen und Wiederholungen möglich. Befehle wie GOTO sind in FORTH nicht vorgesehen und auch nicht beabsichtigt. Trotzdem kann man sich mit den DEFER-Befehlen ein leicht veränderbares und erweiterbares Programm erstellen.

FORTH ist eine offene Programmierumgebung

* Entwicklung eigener Kontrollstukturen
* Verarbeitung beliebiger, auch eigener Datentypen
* Der FORTH-Kern kann selbst modifiziert werden
* Arbeitsumgebung ist frei gestaltbar
    - Problemlose Einbindung eigener Testhilfen
    - Programmeditors den eigenen Wünschen anpaßbar

Das man neue Befehle schreiben kann, ist die Voraussetzung für FORTH. Jedoch beschränkt sich diese Programmierung nicht auf das Zusammensetzen vorhandener Befehle. Statt dessen können eigene Kontrollstrukturen oder Datentypen definiert und dann in das Programm eingebunden

werden. In einigen Fällen kann dabei auch der FORTH-Kern so verändert werden, daß diese neuen Datentypen automatisch erkannt und richtig interpretiert/kompiliert werden.

Da die normalerweise verwendete Programmierumgebung mit Assembler, Editor und diversen Tools erst durch Kompilierung der mitgelieferten Sourcen entsteht, ist jedem Anwender die Veränderung freigestellt. Eine eigene Tastenbelegung für den Editor ist dabei noch eine der einfachsten Übungen.

FORTH schützt die Kenntnisse des Entwicklers

Da nur Zeiger und Name der sichtbaren Befehle gespeichert werden, ist die Dekompilierung des Programmes zum entsprechenden Programm sehr schwierig. Da der Aufwand meistens größer als das Neuschreiben ist, bleiben selbst unverschlüsselte Programme dem Anwender verborgen. Wer aber trotzdem sein Programm schützen will, dem ist es natürlich freigestellt, entsprechende Programmroutinen (Adressen sind leicht zu lokalisieren) mittels XOR-Verknüpfung unlesbar zu machen und erst beim Programmstart wieder zu restaurieren.

2.2 Einführung in FORTH

Die folgende Einführung in FORTH kann mit jedem FORTH83-System (z.B. F83, volksFORTH, F-PC, LMI-FORTH und natürlich KK-FORTH) durchgeführt werden. Hier werden nur der allgemeine Aufbau und das Verhalten von FORTH besprochen. Jedoch wird bei einigen Punkten auch auf das spezielle Verhalten des KK-FORTH hingewiesen.

Hier noch einige Anmerkungen zu den verwendeten Textformatierungen. Sie erleichtern das Lesen und ermöglichen die Unterscheidung zwischen Eingaben des Programmierers und den Antworten des FORTH-Systems.

FORTH-BEFEHLE	im Text werden groß und fett geschrieben
`Programme und Eingaben`	werden in nichtproportionaler Schrift angegeben
`Ausgaben`	sind zusätzlich noch unterstrichen

In den Befehlseingaben werden zur besseren Lesbarkeit die normalen Befehle klein, Definitionsbefehle mit großem Anfangsbuchstaben und Kontrollstrukturen groß geschrieben. Die meisten FORTH-Systeme wandeln alle Eingaben vor der Interpretation sowieso in Großbuchstaben, deshalb ist Groß-/Kleinschreibung nur eine Stielfrage.

2.2.1 Wie arbeitet FORTH

Wenn Sie das FORTH starten, so erscheint meist eine Einschaltmeldung und der Kursor blinkt in Erwartung Ihrer Eingabe. Wenn man jetzt eine Zeile eingetippt und mit der RETURN- oder ENTER-Taste zur Behandlung abschickt, so beginnt FORTH mit der Interpretation dieser Zeile.

```
100  DUP  *  3  +  .
```

So sehen typische Eingabezeilen bei FORTH aus. Sie bestehen aus Zahlen, Wörtern oder Zeichen. Jedes dieser "FORTH-Befehle" ist aber mindestens durch ein Leerzeichen getrennt. Es ist deshalb bis auf einige Ausnahmen auch erlaubt, diese Befehle einzeln einzugeben. Ausnahmen sind die Befehle wie **FORGET** , **VARIABLE** oder **"** , die noch einen Befehlsnamen oder eine Zeichenkette erwarten.

Die Aufgabe von FORTH besteht nun darin, nacheinander von Links nach Rechts aus dieser Eingabezeile die nächste Zeichengruppen (also zuerst "**100**") herauszuholen. Danach sucht es in seiner Liste, ob es diesen Befehl kennt. Kann es nicht gefunden werden, dann versucht FORTH, die Zeichenfolge als eine Zahl in der aktuellen Zahlenbasis zu interpretieren. Erst wenn dies ebenfalls nicht möglich ist, wird mit einer Fehlermeldung abgebrochen.

2.2.2 Das Dictionary

Irgendwie muß es jedoch auch dem armen Programmierer möglich sein, die Liste der verfügbaren Befehle abzufragen. Jedoch muß man dazu noch etwas tiefer in das FORTH-System einsteigen:

Die Liste der Befehle sind in sogenannte Vokabulare zusammengefaßt. Ein FORTH-System hat mindestens ein Vokabular, daß (wie solls auch sonst sein) den Namen **FORTH** trägt. Daneben existieren oft noch andere Vokabulare für den Editor, den Assembler oder für Tools. Vokabulare haben den Vorteil, daß gleiche Befehlsnamen für verschiedene Aufgaben definiert werden können.

Sobald man den Namen eines Vokabulars eingibt, so wird diese Befehlsliste als erstes durchsucht. Es ist im FORTH83-Standard zwar nur als Ergänzung angegeben, aber jedes FORTH-System liefert mit den Befehlen **VOCS** eine Liste der verfügbaren Vokabulare.

```
vocs FORTH  ok
```

Es existiert also nur ein Vokabular. Jedoch ist die Definition eines neuen Vokabulars sehr einfach:

```
Vocabulary meinvoc  ok
vocs MEINVOC FORTH  ok
```

Aber ich weiß noch immer nicht, welche dieser Listen vom FORTH durchsucht wird. Jedoch hilft mir auch dabei ein Befehl, der mir angibt, welche Vokabulare er durchsucht und in welches Vokabular neue Befehle geschrieben werden.

```
order FORTH FORTH  FORTH
```

Dabei ist das letzte FORTH das sogenannte CURRENT-Vokabular, wohin die neuen Befehle gelangen. Die anderen beiden FORHT stellen die CONTEXT-Liste mit den zu durchsuchenden Vokabularen dar. Dabei ist nur der letzte Eintrag durch Eingabe des Vokabularnamen veränderbar und wird auch als erster durchsucht.

Die ORDER-Liste kann durch einigen Befehle verändert werden. Meistens geschieht diese Veränderung automatisch durch Programme wie z.B. der Assembler, der dadurch seine eigene Wortliste mit den Opcodes als erster durchsuchen läßt.

```
meinvoc  ok
order FORTH MEINVOC  FORTH  ok
also  ok
order FORTH MEINVOC MEINVOC  FORTH  ok
forth  ok
order FORTH MEINVOC FORTH  FORTH  ok
seal  ok
order FORTH MEINVOC  FORTH  ok
definitions  ok
order FORTH MEINVOC  MEINVOC  ok
forth  ok
order FORTH FORTH  MEINVOC  ok
definitions  ok
order FORTH FORTH  FORTH  ok
```

Nach dieser Spielerei hat man wieder die gleiche Suchreihenfolge wie nach dem Start des FORTH-Systems. Mit **ALSO** kann diese Liste erweitert werden. Es erfolgt erst dann eine Fehlermeldung, wenn keine weiteren Vokabulare mehr in die Suchreihenfolge aufgenommen werden können. Aufpassen muß man mit dem Befehl **SEAL** . Da er immer das oberste Vokabular aus der Liste entfernt, kann er auch das letzte entfernen und damit die Suche von vorne herein zum Scheitern verurteilen (nur noch Abschalten oder Reset möglich). Jedoch ist der Befehl dann sinnvoll, wenn man nur noch sein eigenes Vokabular in der Liste haben will. Dies ist jedoch nur in kompilierten Programmen möglich, da dort nicht mehr der Name gesucht werden muß.

Nachdem wir jetzt die Verwaltung der Liste kennen, können wir uns die darin enthaltenen Befehl anschauen. Wie nicht anders zu erwarten, gibt es dazu einen eigenen FORTH-Befehl:

```
WORDS                                ( danach die ENTER-Taste drücken )
```

Kaum abgeschickt, erscheint eine recht lange Liste von Namen. War Ihnen die Ausgabe zu schnell, so können Sie es mit einer beliebigen Taste stoppen. Nach einem weiterer Tastendruck wird die Liste fortgesetzt. Mit der CTRL+X wird die Ausgabe im KK-FORTH vorzeitig beendet. Bei **WORDS** sieht man immer nur die Befehle des ersten Suchvokabulars. Neue Befehle werden vor dieser Liste angehängt und Verlängern dadurch das Dictionary. Das Zeichen | vor dem Befehl kennzeichnet Worte, bei dem der Namen getrennt vom Befehl in einem eigenen Speicherbereich, dem sogenannten Heap, aufbewahrt wird. Jedes **SAVE** oder **HCLEAR** entfernt die mit | markierten Namen, ohne daß dadurch die Eigenschaft des Befehls verändert wird.

Der Speicherbereich, in dem alle Vokabulare (oft ineinander geschachtelt) eingetragen sind, bezeichnet man als Dictionary. Da das Dictionary von unten (niedrige Speicheradresse) nach oben aufbaut, erscheinen immer die zuletzt eingegebenen Befehle als erstes in der WORD-Liste. Auch die Suche nach Befehlen erfolgt von oben her. Deshalb kann sogar in einem einzigen Vokabular der gleiche Name mehrmals auftauchen. Neue Befehle beziehen sich dann auf das zuletzt definierte Wort.

Bei Tests ist es oft erforderlich, Befehle nur zur Überprüfung ins System aufzunehmen und danach wieder zu Löschen. FORTH erlaubt deshalb auch das kontrollierte Vergessen von Befehlen. Da der Computer aber nicht weiß, welche FORTH-Befehle von anderen Wörtern aufgerufen werden, wurde eine einfache Festlegung getroffen: Alle Befehle, die nach dem zu vergessenden Wort definiert wurden, werden ebenfalls entfernt. Um sicherzugehen, wird auch die Suchreihenfolge auf FORTH zurückgestellt.

Um das neu definierte Vokabular **MEINVOC** zu vergessen, ist folgende Befehlseingabe notwendig:

```
Forget meinvoc
```

FORGET ist einer der Befehle, bei dem noch der Name des gewünschten Befehls abfragt wird. Dieser Name muß unbedingt noch in der gleichen Zeile, getrennt von **FORGET** durch mindestens ein Leerzeichen, stehen. Gibt man statt **MEINVOC** einen Namen aus dem geschützten Bereich des FORTH-Kerns an, so wird mit einer Fehlermeldung abgebrochen. Man kann im KK-FORTH nur alles bis zum zuletzt definierten Wort durch die Eingabe von **SAVE** schützen, wobei aber der Heap gelöscht wird.

2.2.3 Der Datenstack

Was macht aber FORTH mit einer Eingabe, wenn es als Zahl erkannt wurde? Es speichert diesen Wert auf dem in anderen Programmiersprachen sonst nicht direkt zugängigen Datenstack.

Ein Stack ist wie ein Stapel von Blätter. Neue Blätter werden von oben auf den Stapel gelegt. Es sind normalerweise nur die obersten Blätter erreichbar. Dadurch ist egal, ob vorher nur 2 oder hundert Blätter abgelegt wurden.

In FORTH hat jeder Befehl die Aufgabe, eine bestimmte Anzahl von Werten vom Datenstack zu nehmen, sie miteinander zu verknüpfen und das Ergebnis wieder oben auf den Stack zu legen. Deshalb ist das erste Beispiel natürlich ganz anders zu interpretieren, als es sonst in anderen Programmiersprachen üblich ist.

```
100  DUP  *  3  +  . 10003  ok
```

Zuerst wird der Wert 100 auf den Stapel gelegt. **DUP** ist der erste FORTH-Befehl. Wie man einer im Anhang befindlichen Liste entnehmen kann, nimmt **DUP** diesen Wert und legt in ein zweites Mal auf den Stapel. Dies kann man in einer Art Stackbilanz sehr einfach angeben:

DUP	(n -- n n)	Duplizieren
*	(n1 n2 -- n3)	n3=n1*n2
+	(n1 n2 -- n3)	n3=n1+n2
.	(n --)	Vorzeichenbehaftete Ausgabe der Zahl
/	(n1 n2 -- n3)	Liefert Quotient von n1/n2

Bei dieser Art der Beschreibung wird immer in Klammer zuerst der Zustand des Datenstacks vor der Ausführung angezeigt. Nach dem -- steht dann der Inhalt des Datenstacks nach der Ausführung. Dabei haben gleiche Zeichenketten auch den gleichen Wert.

Bei der oberen Befehlszeile würde die Stackveränderung so aussehen:

```
100             ( -- 100 )
DUP             ( 100 -- 100 100 )
*               ( 100 100 -- 10000 )
3               ( 10000 -- 10000 3 )
+               ( 10000 3 -- 10003 )
.               ( 10003 -- )
```

Die Ausgabe des Ergebnisses durch den FORTH-Befehl **.** erfolgt unmittelbar nach einem bei der Zeileneingabe automatisch folgenden Leerzeichen. Es wird also noch in der gleichen Zeile ausgegeben. Außerdem meldet FORTH durch ein lapidares " ok", daß es seine Eingabezeile vollständig bearbeitet hat und die nächste Eingabe erwartet.

Die hier vorgestellte Art der Formelauswertung wird oft UPN (Umgekehrte Polnische Notation) genannt. Vor allem die HP-Taschenrechner haben sie verwendet. UPN hat den Vorteil, daß jede Formel auch ohne Klammer geschreiben werden kann.

aus:	4*6+3*(5+1)
wird bei UPN:	4 6 * 3 5 1 + * +

Bei jedem Computer oder jeder Programmiersprache haben Zahlen ein bestimmtes Format und eine bestimmte Länge. Aufgrund seines Ursprungs und seiner Hauptanwendung haben FORTH-Zahlen eine feste Länge von 16 Bits (1 Zelle oder 2 Bytes) und sind Integerzahlen. 16 Bit ergeben einen Zahlenbereich von -32768 bis +32767, wobei für negative Werte die sogenannte Zweierkomplementärdarstellung verwendet wird. Bei vorzeichenloser Anwendung geht der Wertebereich von 0 bis 65535. Die Überschreitung des Zahlenbereichen wird nicht als Fehler erkannt, sondern modulo 65536 weitergerechnet.

```
35000 31000 + . 464  ok
```

Bei einer Division wird nur der Integer-Quotient zurückgeliefert. Im FORTH83-Standard hat man sich dazu entschlossen, bei der vorzeichenbehafteten Division immer auf den nächst kleineren Wert (Richtung -unendlich) abzurunden. Dieses als "Floored Division" bezeichnete Verfahren ist jedoch umstritten. Bei Division durch Null wird entweder eine entsprechende Fehlermeldung ausgegeben oder als Ergebnis der größte darstellbare Wert 65535, was dem vorzeichenbehafteten Wert -1 entspricht, zurückgeliefert.

```
100 3 / . 33  ok
-100 3 / . -34  ok
```

Um jedoch bei größeren Zahlen nicht zu versagen, enthält FORTH auch Befehle für 32Bit-Werte. Auch bei der Eingabe kann durch Einfügen eines Punktes an einer beliebigen Stelle in der Zahl angegeben werden, das es sich um einen derartigen Wert handelt.

```
100.000  3000.00  0.300  ok
```

FORTH spaltet diese 32Bit wieder auf und legt sie als zwei 16Bit-Werte auf den Datenstack. Dabei wird zuerst der niederwertige Teil und dann (als oberster Stackwert) der höherwertige Teil abgelegt. Diese Werte können mit eigenen Befehlen, die meist mit "2" oder "D" beginnen, bearbeitet werden (z.B. **2DUP 2SWAP D2* DU=**). Auch die Umwandlung von 16Bit- in 32Bit-Zahlen ist möglich (**S>D**).

Man kann sich auch ansehen, was sich inzwischen alles auf dem Datenstack angesammelt hat und den Stack gezielt wieder Löschen. Jedoch gehören diese Befehl nicht mehr in den Standard und können deshalb in anderen Systemen abweichen:

```
.s
                0 ( $0000 , &00000 )
              300 ( $012C , &00300 )
                4 ( $0004 , &00004 )
           -27680 ( $93E0 , &37856 )
                1 ( $0001 , &00001 )
           -31072 ( $86A0 , &34464 )  ok
dclear  ok
.s
(empty)  ok
```

Das **.S** des KK-FORTH zeigt bis zu 16 Stackeinträge in der aktuellen Zahlenbasis und in Klammern hexadezimal (mit $) und dezimal (mit &) an, wobei der oberste Stackeintrag auch als erster Wert ausgegeben wird. Die Kennungen $ bzw. & wurden deshalb verwendet, weil sie auch bei der Eingabe einer Zahl erlaubt sind.

Neben den Zahlen (16 und 32 Bit) gibt es noch die Datentypen Zeichen (Charakter, meist mit c oder char in der Befehlsliste gekennzeichnet), Flags (f, tf oder ff) und Adressen (addr). Sie werden ebenfalls als 16Bit-Werte auf dem Datenstack abgelegt. Bei Zeichen sind normalerweise nur die unteren 8 Bits interessant. Flags werden meisten durch Kontrollstrukturen abgefragt. Sie besitzen nur die beiden Zustände **FALSE** (ff mit einem Wert von 0) und **TRUE** (tf mit Wert <> 0; bei Vergleichen immer -1).

Zusammenstellung: Datentypen des FORTH

Kennung	Datengröße	Kommentar
n	16 Bit	Vorzeichenbehaftete 16Bit-Zahl
u	16 Bit	Vorzeichenlose 16Bit-Zahl
addr	16 Bit	FORTH-Speicheradresse
d	32 Bit	Vorzeichenbehaftete 32Bit-Zahl
ud	32 Bit	Vorzeichenlose 32Bit-Zahl
f	16 Bit	Flag (Wert nicht definiert)
tf	16 Bit	TRUE-Flag mit Wert<>0 (meist -1)
ff	16 Bit	FALSE-Flag mit Wert 0
char	16 Bit	Zeichen

Da FORTH jedoch ein offenes System ist, kann man ohne Probleme noch weitere Datentypen festlegen. Diese können von einfachen Bitmustern über 32Bit-Zeiger (z.B. ptr des KK-FORTH) bis zu Floatingpoint-Zahlen reichen.

2.2.4 Programmieren in FORTH

Nun wird es endlich Zeit, unseren ersten Befehl zu schreiben. Dazu muß man dem FORTH mitteilen, daß er die folgenden Befehle nicht ausführen, sondern unter einem neuen Namen speichern soll. Der Aufruf des neuen Namens soll dann die Ausführung dieser Befehlsfolge bewirken. Dazu gibt es die Befehle **:** und **;** . Trifft FORTH auf einen Doppelpunkt, so wird die nachfolgende Befehlskette bis zum Strichpunkt gespeichert. Die erste nach dem Doppelpunkt folgende Zeichenkette wird dabei als Befehlsname verwendet. Deshalb ist ein Leerzeichen auch das einzige Zeichen, das normalerweise nicht in einem FORTH-Befehl vorkommen kann.

```
: sqr  dup * ;  ok
```

Wie schon erwähnt gibt es Befehle, die noch Informationen aus der Eingabezeile fordern. **:** gehört zu einer Gruppe von kompilierenden Befehlen, die alle das FORTH-Wort **CREATE** verwenden. **CREATE** legt ein neuen Befehl im Dictionary an und fordert deshalb noch den Namen an. Ist dieser nicht verfügbar, weil z.B. das **:** alleine eingegeben wurde, so erscheint eine Fehlermeldung.

Ansonsten muß man die Befehlsliste nicht in einer Zeile unterbringen, sondern kann die einzelnen Wörter getrennt eingeben. FORTH meldet dem Programmierer, daß es noch weiter kompilieren will. Im KK-FORTH geschieht dies durch ein "]" am Anfang der neuen Zeile an Stelle des " ok". Dieses Zeichen wird deswegen verwendet, weil ein gleichlautender FORTH-Befehl tatsächlich in den Kompiler-Modus schaltet.

```
: sqr  SQR exist
 ] dup *
 ] ;  ok
```

Wie man bei dieser erneuten Definition des Befehls sieht, bekommt man sogar eine Warnung, wenn der Name schon existiert. Ab jetzt wird immer nur der neue Befehl verwendet. Wenn es aber aus Platzgründen nötig ist, so kann vorher noch durch `Forget sqr` die alte Definition gelöscht werden.

Ist der Befehl ordnungsgemäß abgeschlossen worden, so kann er verwendet werden. Auch der Übernahme in weiteren FORTH-Befehlen steht nichts mehr im Wege.

```
5 sqr . 25  ok
: ^3 dup sqr * ;  ok
5 ^3 .  125  ok
```

Man muß hier noch deutlich sagen, daß zwischen dem Interpretieren einer Zeile, und dieser Kompilierung ein Riesenunterschied existiert: Beim Interpretieren wird jeder Befehl einzel abgearbeitet und der Stack sofort verändert. Beim Kompilieren wird der Datenstack normalerweise nicht verändert, sondern nur ein neuen Befehl für das CURRENT-Vokabular erstellt.

"Normalerweise" steht deshalb im letzten Satz, weil bei der Verwendung von **:** und bei Kontrollstrukturen noch Rücksprungadressen und Zusatzparameter auf dem Stack abgelegt und wieder abgefragt werden. Falls aber die Kompilierung des Befehls fehlerlos durchgeführt wurde, so ist der Datenstack wieder in seinem ursprünglichen Zustand.

Trotzdem kann es auch bei einem geübten Programmierer vorkommen, daß er sich bei der direkten Eingabe von Befehlszeilen einmal vertippt oder falsche Kontrollstrukturen einsetzt. In diesem Fall erscheint eine Fehlermeldung und es werden neue Eingaben erwartet. Je nach FORTH-Version ist dann die Kompilierung des Befehls beendet oder die Eingabe kann korrigiert werden. Beim KK-FORTH bleibt der Interpreter bei unbekannten Befehlen oder bei falscher Schachtelung der Kontrollstrukturen weiterhin im Kompilermodus und der Anwender kann den Rest des Befehls mit korrigiertem Text eingeben. Falls er sich aber so verfranzt hat, daß nur ein hilft, so genügt die Eingabe von **[** . Der unvollständige Befehl ist aber dann nicht verwendbar, und auch nicht verfügbar. Das Löschen dieser Definition mit **FORGET** ist deshalb erst nach der Sichtbarmachung mit **REVEAL** möglich. Er darf aber in keinem Falle ausgeführt werden, da er einen Programmabsturz verursachen kann.

2.2.5 Der Returnstack

Wenn das FORTH einen Befehl ausführt, muß es sich merken, wohin es nach dessen Ende wieder zurückkehren soll. Dies geschieht ebenfalls auf einem Stapel, dem sogenannten Returnstack. Auch hier wird immer nur von oben ein neuer Wert abgelegt oder ausgelesen. Normalerweise wird dieser Stapel nur versteckt durch die Kernroutinen verwendet. Man kann aber auch eigene Daten dort aufbewahren, solange man sicherstellt, daß beim Befehlsende oder am Ende einer DO...LOOP- (bzw. FOR...NEXT-) Schleife alle Daten wieder entfernt wurden. Wörter, die dies nicht beachten, führen unweigerlich zum Absturz des Systems. Vorprogrammiert währe dieser Absturz, wenn die entsprechenden Befehle im Interpretermodus verwendet werden. Deshalb lösen sie beim KK-FORTH außerhalb von ":"-Definitionen eine Fehlermeldung aus.

Die am häufigsten verwendeten Befehle für die Manipulation des Returnstacks sind **>R** , **R>** , **R@** und **PUSH** . Im ANSI-Vorschlag hat man aber noch entsprechenden Befehle für die 32Bit-Zahlen wie **2>R** , **2R>** und **2R@** definiert. Es sind absichtlich keine Befehle für die Arithmetik vorgesehen.

Die Eigenschaft, daß der Returnstack die Rückkehradresse speichert, wird von einigen Befehlen ausgenützt. Bei sogenannten Inline-Literals oder Inline-Strings wird ein im normalen Programm untergebrachter Wert verarbeitet und die Rückkehradresse so erhöht, daß hinter den Daten das Programm fortgesetzt werden kann. Diese besondere Eigenschaft des FORTH, wird noch mehrmals in diesem Buch sowohl bei der ausführlichen Beschreibung des KK-FORTH als auch beim Arbeiten mit KK-FORTH ausführlich behandelt.

2.2.6 Konstanten, Variablen und USER-Variablen in FORTH

Nur über einen Datenstack die Parameter auszutauschen, ist meistens nicht genug. Oft gibt es Konstanten, die von den verschiedensten Befehlen verwendet und Variablen, die im Laufe des Programmes verändert werden.

Konstanten

Konstanten sind (16Bit-) Zahlenwerte, die im gesamten Programm gleichbleiben. Dazu zählen z.B. Tastencodes, Port- und Speicheradressen. Um bei Änderungen dieser Konstanten wegen einer geänderten Hardware oder bei Umsetzung auf andere FORTH-Versionen nicht das gesamte Programm Befehl für Befehl durchzugehen, kann ein neuer Eintrag in das Dictionary erfolgen. Bei späterer Verwendung der Konstante wird dann der festgelegte Wert auf dem Datenstack abgelegt.

```
.s
(empty)  ok
100 Constant hundert  ok
.s
(empty)  ok
hundert  ok
.s
                100 ( $0064 , &00100 )  ok
. 100  ok
```

Neben der am häufigsten verwendeten 16Bit-Konstante sind in den meisten FORTH-Versionen noch die Definition von 32Bit-Konstanten vorgesehen:

```
3.1415 2Constant pi  ok
pi d. 31415  ok
```

Variablen

Variablen sind Datentypen, die häufig vom Programm abgefragt und auch geändert werden. Im FORTH liefert eine Variable deshalb die Speicheradresse eines 16Bit-Wertes, der dann gelesen oder verändert werden kann. Bei der Erstellung einer Variable braucht man keinen Startwert anzugeben, da er automatisch mit 0 vorbelegt wird. Man muß aber in eigenen Programmen sicherstellen, daß er beim Start auf einen sinnvollen Wert gesetzt wird, da er durch das FORTH nicht verändert wird.

```
Variable test  ok
.s
(empty)  ok
test  ok
.s
              -3218 ( $f36E , &62318 )  ok
2 dump
0956:F36E  00 00 ...
```

```
hex  ok
1234 test !  ok
test 2 dump
0956:F36E  34 12 ...
test @ u. 1234  ok
5 test +!  ok
test @ u. 1239  ok
test on  ok
test @ u. FFFF  ok
test off  ok
test @ u. 0  ok
decimal;  ok
```

Das Aufrufen des Befehls **TEST** liefert die Speicheradresse, in der die Variable gespeichert wird. Diese Adresse kann dann mit **@** gelesen und mit **!** oder **+!** verändert werden. Da Variablen auch häufig nur zur Ablage von Flags verwendet werden, stehen in den meisten FORTH-Versionen auch die Befehle **ON** und **OFF** zur Verfügung. Sie schreiben das TRUE- bzw. FALSE-Flag in die Variable.

Zusätzlich lernen wir an dem oberen Beispiel noch weitere FORTH-Befehle kennen, die häufig für Programmtest oder für die Ein-/Ausgaben benötigt werden:

dump	(addr len --)	Dump eines Speicherbereiches
hex	(--)	Änderung der Zahlenbasis auf 16
decimal	(--)	Änderung der Zahlenbasis auf 10

Der Dump eines Speicherbereiches dient hier zur Analyse der Variable. Da dieses Beispiel mit dem KKF_PC durchgeführt wurde, wird bei der Ausgabe auch das verwendete Segment angegeben. Der Wert der Variable wird in der Reihenfolge Low-Byte ... Highbyte abgelegt. Da aber diese Reihenfolge bei anderen Prozessoren (z.B. 68000 oder RTX-2000) auch umgedreht sein kann, sollten nur die dafür vorgesehenen Befehle verwendet werden. Sehr schwer zu findende Fehler entstehen oft dadurch, daß man mit **!** einen 16Bit-Wert im Speicher ablegt und dann mit **C@** nur ein Byte davon holt und dabei statt dem Low-Byte das High-Byte erwischt.

Bisher gingen wir in den Beispielen immer davon aus, daß alle Werte in der gewohnten Zahlenbasis 10 bearbeitet werden. Da aber FORTH die Werte intern sowieso als Bitmuster bearbeitet, könnte sowohl bei der Eingabe als auch bei der Ausgabe eine andere Zahlenbasis gewählt werden. In der USER-Variable **BASE** ist der momentane Wert der Zahlenbasis gespeichert. Sie ist beim Start des FORTH-Kerns normalerweise auf 10 gesetzt, kann aber fast beliebig geändert und auch gespeichert werden. Setzt man den Wert von **BASE** auf 2, so erwartet und liefert das FORTH Bitmuster. In der Computertechnik verwendet man meistens die Zahlenbasis 16, da man (wie bei DUMP) dann mit zwei "Ziffern" den Wert eines Bytes angeben kann. Dabei verwendet man für die "Ziffern" 10 bis 15 die Buchstaben A bis F. Da die Zahlenbasis 10 und 16 am häufigsten verwendet wird, wurden sie als eigene Befehle definiert. Da nach den Großbuchstaben erst noch Sonderzeichen folgen, sollte **BASE** nicht über 36 gesetzt werden. In einigen FORTH-Versionen kann man durch die Verwendung eines sogenannten Prefix bei der Eingabe die Zahlenbasis für den nachfolgende Wert auch auf 2 (%), 10 (&) oder 16 ($) festlegen. Er wird aber danach wieder auf den ursprünglichen Wert zurückgestellt.

```
decimal  ok
$11 . 17  ok
&11 . 11  ok
%11 . 3  ok
hex  ok
$11 . 11  ok
&11 . B  ok
%11 . 3  ok
binary  ok
$11 . 10001  ok
&11 . 1011  ok
```

```
%11 . 11  ok
decimal  ok
```

USER-Variablen

Eine USER-Variable wird, wie im oberen Beispiel das **BASE** , wie eine normale Variable behandelt. Sie hat aber ein eigenes Definitionswort und wird auch in einem anderen Speicherbereich als die anderen Variablen aufbewahrt.

Die Sonderbedeutung einer USER-Variable wird erst dann klar, wenn man sich vorstellt, daß mehrere FORTH-Programme gleichzeitig in einem Rechner laufen. Es ist dann notwendig, daß ein bestimmter Speicherbereich und auch einige Variablen nur für einen dieser sogenannten Tasks verfügbar ist und nicht durch andere Tasks verändert wird. Obwohl die Definition einer USER-Variable für alle Tasks gilt, liefert dieser Befehl für jeden eine andere Speicheradresse. Dadurch ist sichergestellt, daß jeder Task nur "seine" USER-Variable verändert.

In den meisten FORTH-Systemen werden alle für den Interpreter/Kompiler relevanten Variablen mit **USER** definiert. Falls es das Betriebssystem oder die Hardware unterstützt, so kann gleichzeitig eine Eingabe in der Zahlenbasis 10 und eine Bildschirmausgabe von Bitmustern durch einen anderen Task durchgeführt werden.

Im KK-FORTH sind auch alle Ein-/Ausgabebefehle und die Fehlerbehandlung durch USER-Variablen vektorisiert. Dadurch können z.B. in FORTH geschriebene Interruptprogramme andere Fehlerbehandlungsroutinen als das Hauptprogramm aufrufen.

```
User errorhandler  ok
: error  ?dup IF errorhandler perform THEN ;  ok
```

2.2.7 Standardbefehle von FORTH

Nun wird es aber Zeit, eine Übersicht über die Standardbefehle von FORTH zu erhalten. Jedes System weicht zwar in der Anzahl der Befehle voneinander ab, trotzdem gibt es ein Minimum, auf daß sich alle geeinigt haben. Dieser FORTH83-Standard legt für ca. 110 Befehle genau fest, welcher Name mit welcher Aktion verknüpft ist. Darüber hinaus gibt es viele Befehle, bei denen man schon vom Namen auf die Aufgabe schließen kann.

Hier folgt nun eine kurze Übersicht über die Befehlsgruppen. Eine genaue Beschreibung der einzelnen Befehle ist dem Anhang zu entnehmen.

Systemadressen, -konstanten und -variablen:

LATEST	HERE	PAD	FIRST	LIMIT
DP	S0	R0		
BASE	SPAN	#TIB	STATE	

Behandlung des Datenstacks:

PICK	DUP	OVER	?DUP
ROLL	SWAP	ROT	

Returnstack-Behandlung:

>R	R@	R>	I	J

Arithmetik und Vergleich:

```
AND       OR        XOR       NOT
1+        2+        1-        2-
+         -         NEGATE    ABS
UM*       UM/MOD
*         /         MOD       */
>         =         <
0<        0=        0>
MIN       MAX
U<        D<
```

Ein-/Ausgabe und Diskettenschnittstelle:

```
KEY       EXPECT    QUERY
EMIT      TYPE      CR        SPACE     SPACES
BUFFER    BLOCK     UPDATE    SAVE-BUFFERS   FLUSH
LOAD
```

Zahlenein-/ausgabe und Interpreter:

```
<#        #         #S        HOLD      SIGN      #>
U.        .
.(        (
'         FIND      FORGET
WORD      -TRAILING CONVERT   DEFINITIONS
QUIT      FORTH-83  ABORT
```

Kompiler:

```
BEGIN     WHILE     REPEAT    UNTIL
DO        LEAVE     LOOP      +LOOP
IF        ELSE      THEN
CREATE    DOES>     VARIABLE  CONSTANT  VOCABULARY
:         ;         IMMEDIATE
ALLOT     ,         COMPILE   [COMPILE]
ABORT"    ."
[         ]         [']
```

2.2.8 Strukturierte Programmierung mit FORTH

Normalerweise besteht ein Programm nicht nur aus einer festgelegten Folge von Befehlen. Es muß meistens auf verschiedene Parameter unterschiedlich reagieren oder eine bestimmte Aufgaben solange durchführen, bis alles verarbeitet oder ein Abbruchkriterium erfüllt wurde.

Dies ist jedoch mit den bis jetzt bekannten Befehlen nicht möglich. Doch auch hierzu gibt es besondere FORTH-Befehle, den Kontrollstrukturen. Da diese Strukturen jedoch wissen müssen, wo Anfang und Ende von Schleifen oder Bedingungen sind, sind sie nur während des Kompilierens (in ":"-Definitionen) verfügbar. Kontrollstrukturen sind ineinander schachtelbar, dürfen aber nicht gemischt werden.

Immediate-Befehle

Doch bisher haben wir immer gehört, daß innerhalb von :-Definitionen die Befehle nicht ausgeführt, sondern gespeichert werden. Jedoch muß es Ausnahmen geben, da auch bei **;** mehr passiert als nur die Speicherung eines Befehls.

In FORTH hat jeder Befehl in seinem Header eine sogenannte Namensfeldadresse. In dieser NFA, die nur eine Länge von einem Byte hat, steht neben der Länge des Befehlsnamen (bis 31 Zeichen,

belegt 5 Bit) noch drei weitere Bits für das System zur Verfügung. Eines dieser Bits gibt in allen FORTH-Versionen an, daß dieser Befehl auch in :-Definitionen immer ausgeführt wird. Diese sogenannten Immediate-Befehle werden hauptsächlich zur Umschaltung des Interpreter-/Kompilermodus oder zur Sonderbehandlung von Daten (oder Kommentaren) innerhalb eines Programmes verwendet. Jedoch kann auch jeder Programmierer diese Eigenschaft des Befehls aktivieren, in dem er nach seiner Definition noch **IMMEDIATE** angibt.

```
: meldung  cr ." Befehl  MELDUNG  ausgeführt " cr ; immediate  ok
: mittel  + meldung 2/ ;
Befehl  MELDUNG  ausgeführt
 ok
4 8 mittel . 6  ok
```

Man sieht an dem Beispiel, daß der Befehl **MELDUNG** noch während der Definition von **MITTEL** ausgeführt wird. Er wird aber nicht mehr Kompiliert und hat deshalb keine Wirkung bei der Ausführung von **MITTEL** . Angewendet wird diese Fähigkeit bei Kommentaren, die mit einer öffnenden Klammer beginnen. Der Immediate-Befehl **(** sucht bis zu der ersten schließenden Klammer und ignoriert einfach den Text dazwischen. Dadurch erklärt sich auch, warum ein Kommentar keine verschachtelten Klammern verwenden darf.

Bedingte Programmausführung

(Flag) IF (Teil 1) THEN
(Flag) IF (Teil 1) ELSE (Teil 2) THEN

Die am häufigsten auch in anderen Programmen verwendete Struktur ist die einfache Fallunterscheidung, bei dem je nach Bedingung und Aufbau entweder nichts oder eine von zwei Befehlsfolgen bearbeitet wird. Auch hier geschieht die Parameterübergabe im FORTH-Stiel. Zuerst wird ein Flag auf dem Datenstack abgelegt. Als Flag kann praktisch fast jedes Ergebnis von Vergleichen verwendet werden. Das **IF** entfernt dann dieses Flag und führt nur dann den Teil 1 aus, wenn das Flag einen Wert ungleich 0 hat. Ansonsten wird sofort zum **THEN** gesprungen oder der Teil 2 bearbeitet, falls dieser mit einem **ELSE** markiert wurde. **ELSE** hat dabei auch die Aufgabe, daß nach der Bearbeitung des Teil 1 der zweite Teil übersprungen wird.

```
: zahl.   ( n -- ) ( Gibt Zahlenwerte kleiner 3 als Text aus )
   dup 0 =  IF  ." Null"  THEN
   dup 1 =  IF  ." Eins"  THEN
   dup 2 =  IF  ." Zwei"  THEN
   dup 3 =  IF  ." Drei"  THEN
   drop ;
```

In diesem Beispiel wurden zum ersten mal auch Kommentare bei einer Befehlsdefinition angegeben. Wie schon oben erklärt, dürfen sie auch innerhalb von :-Definitionen verwendet werden. Bei den meisten Listings sieht man eine derartige Befehls-Beschreibung des neuen Befehls, weil er zum einen die Veränderung des Stacks und dann die Aufgaben des Befehls beschreibt.

Bedingte Wiederholungen

```
BEGIN  (Teil 1)  (Flag) WHILE  (Teil 2)  (FLAG) UNTIL
BEGIN  (Teil 1)  (Flag) WHILE  (Teil 2)  REPEAT
```

Hier ist **BEGIN** nur eine Markierung, bei der das Programm fortgesetzt wird, wenn **UNTIL** ein FALSE-Flag (mit Wert 0) erhält. **REPEAT** spring zu **BEGIN** zurück, ohne dazu ein Flag zu testen. Um jedoch auch zwischendurch diese Programmstruktur zu verlassen, gibt es **WHILE** . Dieser Befehl springt bei einem FALSE-Flag hinter das Ende der Programmstruktur. Im KK-FORTH ist es sogar erlaubt, das **WHILE** mehrfach anzuwenden.

Programmschleifen

```
(Ende+1)  (Anfang) .i.DO;  ...  .i.LOOP;
(Grenze)  (Anfang) DO  ...  (Wert) .i.+LOOP;
```

LEAVE	(--)	Schleife sofort verlassen
I	(-- n)	Liefert aktuellen Schleifenwert
J	(-- n)	Liefert Wert der nächst äußeren Schleife

Programmschleifen haben entweder eine feste Anzahl von Durchläufen mit Schrittweite 1 oder sind sowohl in der Anzahl als auch in der Schrittweite flexibel. Durch die Schleife **DO** ... **LOOP** wird die Schrittweite 1 unterstützt. Dazu entfernt **DO** die beiden obersten Datenstackwerte und wiederholt die Schleife solange, bis der Endwert erreicht wird. Oben auf dem Stack liegt bei **DO** der Anfangswert für den ersten Durchlauf. Bei **+LOOP** muß noch der Wert angegeben werden, der zum aktuellen Schleifenparameter addiert wird. Hier wird dann die Schleife erst dann verlassen, wenn die Grenze zwischen Ende-1 und Ende überschritten wird. Mit dem Befehl **I** wird der aktuelle Schleifenwert auf den Datenstack gebracht. Mit **LEAVE** kann eine Schleife sofort verlassen werden.

```
: zähle  10 0 DO  i .  LOOP ;  ok
zähle 0 1 2 3 4 5 6 7 8 9  ok
```

Die Parameter für die Schleifen werden auf dem Returnstack aufbewahrt. Deshalb kann man nicht in Unterbefehlen mit **I** auf die aktuellen Schleifenparameter zugreifen. Außerdem muß bei allen Befehlen für die Programmschleifen der Returnstack so sein, wie er bei **DO** war.

```
(Ende+1) (Anfang) .i.?DO;  ...  LOOP
(Grenze) (Anfang) ?DO  ...  (Wert) +LOOP
```

?LEAVE	(f --)	Schleife verlassen, wenn Flag ungleich 0 ist

In den meisten FORTH-Versionen wurden die Möglichkeiten bei Schleifen durch weitere Befehle wie **?DO** und **?LEAVE** noch erweitert. Falls man verhindern will, daß bei der Befehlsfolge `0 0 DO i . LOOP` die Schleife 65536 mal durchlaufen wird, so kann man dies durch **?DO** verhindern. **?DO** verhält sich wie **DO** , überprüft aber vorher, ob Anfangswert und Endwert identisch sind. **?LEAVE** ersetzt die Befehlsfolge `IF leave THEN` und dient dadurch dem kontrollierten Verlassen der Schleife.

2.2.9 Ein-/Ausgaben in FORTH

Mit den bis jetzt kennengelernten Befehlen ist schon ein Großteil der Programme zu schreiben. Für Menüprogramme fehlen aber immer noch Möglichkeiten zur Abfrage der Tastatur und für die Ausgabe von Texten oder eigene Zahlenformate.

Tastaturabfrage und Zeileneingaben

key	(-- char)	Holt ein Zeichen von Tastatur
key?	(-- f)	Liefert Flag, ob ein Zeichen bereitsteht
expect	(addr len --)	Holt einen String mit bis zu len Zeichen
query	(--)	Holt eine Eingabezeile

Im Standard ist leider nur der Befehl **KEY** zur Abfrage der Tastatur vorgesehen. **KEY** hat aber den Nachteil, daß immer solange gewartet wird, bis tatsächlich ein Zeichencode vorliegt. Dieser Zustand ist wurde im KK-FORTH durch zusätzliche Abfragen ausgeglichen. Doch hier hat der fehlende Standard den Nachteil, daß jeder FORTH-Entwickler eigene Definitionen wie **?TERMINAL** , **?KEY** oder **(KEY** verwendet. Im KK-FORTH wird durch **KEY?** ein Flag ungleich 0 geliefert, wenn ein Zeichen ohne Warten geholt werden kann.

Die unteren zwei Befehle dienen der Eingabe eines ganzen Strings. Dazu wird bei **EXPECT** sowohl Anfangsadresse als auch die maximale Länge des Zahlenstrings eingegeben. Erst wenn man die Return-Taste betätigt, wird die Eingabe beendet und die tatsächlich erhaltene Zeichenanzahl in der (USER-)Variable **SPAN** gespeichert. **QUERY** arbeitet ähnlich, jedoch wird sowohl die Speicheradresse als auch die Länge für eine Befehlszeile vorgegeben. Dabei hat dieser Befehl noch den Vorteil, daß er einige Variablen so verändert, wie sie für die Interpretation der Zeile benötigt werden.

Ausgabe von Zeichen und Strings

EMIT	(char --)	Ausgabe eines Zeichens
EMIT?	(-- f)	Liefert Flag, ob Ausgabe möglich ist
CR	(--)	Cursor zum Anfang der nächsten Zeile
SPACE	(--)	Gibt ein Leerzeichen aus
SPACES	(+n --)	Gibt n Leerzeichen aus
TYPE	(addr len --)	Gibt einen String aus

Da Ausgaben sehr häufig erfolgen, wurden auch sehr viele Möglichkeiten dazu geschaffen. Bis auf **EMIT?** , daß vor allem in Druckerspoolern oder Multitasking-Programmen sinnvoll ist, sind alle Definitionen aus dem FORTH83-Standard.

Jedes druckbare Zeichen, daß man durch **KEY** bekommen hat, kann sofort mit **EMIT** wieder ausgegeben werden. **CR** ist definiert, weil nicht bei allen Betriebssysteme ein Zeilenumbruch durch Steuercodes möglich ist und durch **CR** getrennte Routinen aufgerufen werden können. **SPACE** wird sehr häufig verwendet und spart durch seine Definition eigentlich nur Platz. Der Befehl **SPACES** wird bei der Zahlenausgabe verwendet und hat deshalb die Besonderheit, daß bei negativen Werten nichts ausgegeben wird. **TYPE** dient zur Ausgabe eines ganzen Strings und erwartet dazu die Speicheradresse des ersten Zeichens und die Länge des gesamten Strings.

```
$2a emit * ok
space  $2a emit  * ok
cr $2a emit 5 spaces $2a emit
*     * ok
: string. " Teststring" count type ;  ok
string. Teststring ok
```

Formatierte Zahlenausgabe

Oft werden Daten von Menüprogrammen in einem bestimmten Format ausgegeben. Deshalb sind in FORTH neben den beiden Grundformaten auch die Befehle für eigene Zahlenformatierungen vorgesehen:

<#	(--)	Initialisierung der (USER-)Variable **HLD**
HOLD	(char --)	Setzt Zeichen char vor den Zahlenstring
#	(ud1 -- ud2)	Setzt eine Ziffer vor den Zahlenstring
#S	(ud -- 0.)	Wiederholt **#** , bis das Ergebnis 0 ist
SIGN	(n --)	Setzt "-" vor den String, wenn n<0
#>	(d -- addr len)	Liefert Adresse und Länge des Zahlenstrings

Zur Erstellung des Zahlenstrings ist ein bestimmter Speicherbereich vorgesehen, der durch die von **PAD** gelieferte Adresse nach oben begrenzt wird. Beim Start der Formatierung wird mit **<#** diese Adresse in den Zeiger **HLD** übernommen. Danach kann man dann mit **HOLD** oder **SIGN** ein bestimmtes Zeichen vor den String setzen. Dazu wird zuerst **HLD** erniedrigt und dann das Zeichen an die angegebene Adresse gebracht.

Bei der Zahlenkonvertierung wird immer davon ausgegangen, daß eine vorzeichenlose 32Bit-Zahl vorliegt. Durch **#** wird dann dieser Wert durch die in **BASE** gespeicherte Zahlenbasis geteilt und der Quotient auf dem Stack gelassen. Der Rest dieser Division wird aber in eine Ziffer umgewandelt und vor den String gesetzt. Wie wir gehört haben, können auch Reste größer als 9 entstehen. In diesen Fällen werden daraus Großbuchstaben gemacht (10=A; ... ; 35=Z).

Durch **#S** wird **#** mindestens einmal, aber danach nur solange aufgerufen, bis der Quotient den Wert 0. hat. Um auch vorzeichenbehaftete Zahlen auszugeben, wird vorher das Vorzeichen gespeichert und dann mittels **SIGN** dem Zahlenstring angehängt, falls der Wert negativ ist.

Am Schluß entfernt dann **#>** den 32Bit-Wert und ersetzt ihn durch Adresse und Länge des Zahlenstrings. Deshalb kann der String sofort durch **TYPE** ausgegeben werden.

```
:    ( d len -- ) ( 32Bit-Wert rechtsbündig mit len Zeichen )
   >r  dup >r                    ( Länge und Vorzeichen merken )
   dabs                          ( Absolutwert von d bilden )
   <# #s r> sign #>              ( Zahlenstring erzeugen )
   r> over - spaces  type ;      ( Zuerst Leerzeichen, dann String )

-4. 5 d.r    -4 ok
$-1234. 1 d.r -4660 ok
```

Die Definition von **D.R** dient hier als Beispiel und muß nicht eingegeben werden, weil es in allen FORTH-Versionen schon definiert wurde. Wie man bei der Ausgabe feststellt, wird auch dann der gesamte Zahlenstring ausgegeben, wenn er nicht in die vorgegebene Feldgröße paßt. In Menüprogrammen würde dabei die verwendete Bildschirmmaske zerstört werden.

2.2.10 Die Diskettenschnittstelle des FORTH

Natürlich ist es auf die Dauer nicht tragbar, jeden Befehl immer neu einzugeben. Deshalb gibt es auch Kommandos, die das Laden von Diskette bewirken. Jedoch vorher noch etwas zur Entwicklung des Fileinterface bei FORTH.

FORTH wurde schon zu Zeiten verwendet, als man bei Personalcomputer nur einfache Diskettenlaufwerke verwendete. Damals organisierte man noch nicht in Files, sondern griff direkt auf die einzelnen Sektoren einer Diskette zu. Aus dieser Zeit kommen auch noch die entsprechenden FORTH-Befehle.

Organisiert wurde der Diskettenzugriff so, daß man einen bestimmten Teil einer Diskette in den Speicher ladet, dort verarbeitet und danach wieder zurückschreibt. Man teilt dabei die gesamte Diskette in sogenannte Blöcke ein. Jeder Block hat eine Größe von 1024 Bytes (oder Zeichen), den sogenannten Screens. Dabei sind die Blöcke von 0 bis zur maximalen Diskettengröße durchnumeriert. Um nun auch Programme in diesen Blöcken unterzubringen, teilt man für den Programmeditor noch diese 1024 Zeichen in 16 Zeilen zu je 64 Zeichen auf. Dies gilt jedoch nur für die Darstellung auf dem Bildschirm. Im Speicher wird weiterhin eine durchgehende Zeichenkette von 1024 Bytes verwendet. Darauf ist auch bei der Eingabe von Programmen (vor allem am Zeilenende) zu Achten.

Inzwischen haben sich natürlich die Computer und besonders die Diskettenschnittstellen weiterentwickelt. Es gibt die sogenannten Files, in denen der Computer seine Informationen unterbringt. Files werden durch die Betriebssysteme unterstützt. Besonders das Kopieren oder Überspielen auf andere Datenträger muß jetzt nicht mehr vom Anwender selbst erledigt werden. Da es sich jedoch nicht mehr um einen Standard handelt, wird deren Behandlung im nächsten Kapitel ausgeführt.

BUFFER	(n -- addr)	Speicher für einen Diskpuffer anfordern
BLOCK	(n -- addr)	Einen Diskpuffer von Diskette holen
UPDATE	(--)	Letzten Diskpuffer als geändert markieren
SAVE-BUFFERS	(--)	Alle geänderten Diskpuffer retten
EMPTY-BUFFERS	(--)	Alle Diskpuffer ohne Sicherung freigeben
FLUSH	(--)	Diskpuffer retten und dann freigeben
LOAD	(n --)	Eingabe aus Screen n holen

Mit diesen Befehlen ist der gesamte FORTH83-Standard abgedeckt. Man kann mit **BUFFER** nur den Speicherbereich anfordern oder bei **BLOCK** sogar den Inhalt von Diskette laden lassen.

Nach dessen Veränderung soll er dann mit **UPDATE** markiert werden. Er wird aber erst dann zurückgeschrieben, wenn der Platz für einen anderen Diskpuffer benötigt oder **SAVE-BUFFERS** bzw. **FLUSH** aufgerufen wird. **EMPTY-BUFFERS** löscht den Diskettenpuffer, ohne das vorher die Daten gesichert werden.

Mit **LOAD** hat man den Grundbefehl zur Übernahme eines Programmes von Diskette. Zur Programmerstellung wird aber ein Editor benötigt, der dieses spezielle Fileformat verarbeiten kann. Deshalb wird bei den meisten FORTH-Versionen der Editor als eigenes Programm mitgeliefert oder kann als Screen-File geladen werden.

Es fällt auf, daß es keine Befehle zur Angabe des gewünschten Files gibt. Dieser Umstand führte leider zu den unterschiedlichsten Definitionen. Wie es das KK-FORTH macht, kann im nächsten Kapitel nachgelesen werden.

2.2.11 Nachwort zum FORTH-Grundkurs

In diesen ca. 16 Seiten können bei weitem nicht alle Möglichkeiten des FORTH besprochen werden. Deshalb wurde hier bewußt nur das Grundvokabular und die wichtigsten FORTH-Befehle angesprochen. Falls Ihnen diese Einführung etwas zu schnell oder zu oberflächlich war, so muß ich auf die Literaturangaben in Kapitel 2.5.1 verweisen.

Jedoch ist das noch lange nicht das Ende des FORTH-Kurses, da im nachfolgenden Kapitel 2.3 das Arbeiten mit dem KK-FORTH an vielen Beispielen noch ausführlicher besprochen wird. Da aber dabei der Standard verlassen wird und deshalb einige der Möglichkeiten in anderen FORTH-Versionen nicht zur Verfügung stehen, wurde ein eigenes Kapitel angelegt.

Hier noch ein allgemeiner Tip für FORTH-Anfänger. In den meisten der mir bekannten Fälle, bei dem ein Programmierer diese Programmiersprache erlernte, kam die Begeisterung erst dann auf, als die ersten Beispiele liefen. Dabei stellte sich immer heraus, daß nicht das Durcharbeiten des Lehrbuches, sondern das Lösen einer Aufgabe mit gelegentlicher Hilfe eines FORTH-Profis den schnellsten Lernerfolg bringt.

Man sollte deshalb erst einmal überlegen, was man mit dem in FORTH zu programmierenden Computer machen will und sucht dann nach einem Programmbeispiel, das man dazu verändern kann. Man tut sich auch leichter mit dem Erlernen dieser "Fremdsprache", wenn man nicht verzweifelt versucht, alle Befehle auswendig zu lernen. Es genügen oft die in diesem Kapitel behandelten Befehle und wenige Zusatzwörter (wie Portzugriffe), die in dem nach Befehlsgruppen geordneten Kurzglossar (Anhang A) sehr schnell zu finden sind.

2.3 Arbeiten mit dem KK-FORTH

Ab hier berufen wir uns nicht nur auf die ca. 110 Befehle des FORTH83-Standard, sondern verwenden viele der zusätzlichen 370 Befehle des KK-FORTH. Dabei ist das Ziel dieses Kapitel die Erstellung eines Autostartprogrammes. Dazu lernen wir alle notwendigen Schritte zur Eingabe und zum Laden eines Programmes und einige Besonderheiten des KK-FORTH kennen.

Nicht alle der verwendeten Befehle sind schon bekannt. Sollte die Bedeutung unklar sein, so kann im Anhang B beim alphabetischen Glossar nachgeschlagen werden.

2.3.1 Vorbereitungen vor dem Programmstart

Wie schon im Kapitel 1 erwähnt, sind die KKF-Disketten nicht geschützt. Da zur Programmentwicklung meistens ein Rechner mit Festplatte verwendet wird, sollten alle Files in ein Unterverzeichnis kopiert werden.

Bei den EPROM-Versionen muß anschließend noch das in der Zusatzanleitung angegebene File in ein EPROM gebrannt und in die Platine eingesetzt werden. Nachdem die Stromversorgung sichergestellt und die RS232-Leitung mit dem freien Port des Terminal-Rechners verbunden ist, kann es losgehen. Falls ohne Handshake gearbeitet wird, sollte man sich noch vergewissern, daß die Pin's 4 und 5 bzw. 6 und 20 am 25-poligen RS232-Stecker gekreuzt sind.

2.3.2 Start des KK-FORTH und die Verwendung des Zeileneditors

Das Laden und Starten von Programmen ist bei jedem Rechner anders. Beim PC genügt z.B. die Eingabe des Programmnamens KKF_PC.COM . Für die EMUF-Versionen muß zuerst das Terminalprogramm (TERMINAL.COM) gestartet und mit den Tasten ALT+B bzw. ALT+C die Baudrate und der Port eingestellt werden. Danach ist die EMUF-Platine mit Strom zu versorgen und manchmal noch der Reset-Taster zu betätigen. Falls Sie sich die lästige Initialisierung des Terminalprogrammes ersparen wollen, sollte die aktuelle Einstellung mit ALT+T unter einem neuen Namen speichert und danach immer nur das angepaßte Programm verwendet werden.

In allen Fällen meldet sich das KK-FORTH nach kurzer Initialisierung mit einer Einschaltmeldung, die wie folgt aussehen könnte:

```
KK-FORTH_PC V1.2/0
- (C) Klaus Kohl -
```

Bei einigen Betriebssystemen (PC, MC-32, CP/M ...) können beim Aufruf des Programmes weitere Daten hinter dem Filenamen folgen. Dies wird beim Start automatisch erkannt und dieser Text als erste Eingabezeile interpretiert. Jedoch sollten dabei einige versionsspezifische Einschränkungen und Betriebssystem-Besonderheiten beachtet werden.

Ansonsten wartet der Cursor am Anfang der nächsten Zeile auf die Eingaben. Es können alle Zeichen mit ASCII-Wert größer 31 eingegeben werden. Dabei hat man es hier mit einem komfortablen Zeileneditor zu tun, der auch Zeichen in der Mitte der Zeile verändern kann. Zusätzlich können bei mehrmaligen Drücken der ESC-Taste die letzten Eingabezeilen zurückgeholt und dann geändert werden. Folgende Tasten dienen dabei zur Steuerung:

>	Cursor um ein Zeichen nach rechts
<	Cursor um ein Zeichen nach links
DEL oder CTRL+G	Löschen des Zeichens unter dem Cursor
<-- oder BACKSPACE	Löschen des Zeichens unter dem Cursor
CTRL+X	Gesamte Eingabezeile löschen
^ oder ESC	Zurückholen der letzten Eingabezeile(n)
ENTER	Übernahme der Eingabezeile

Falls diese Tasten nicht verfügbar sind, so wurden andere Tasten oder -kombinationen verwendet. Angaben dazu sind der Zusatzbeschreibung zu entnehmen.

2.3.3 Die Befehle des KK-FORTH

Wir sind jetzt im KK-FORTH und können mit der Untersuchung der Möglichkeiten beginnen. Dazu sollte man zuerst einmal feststellen, welche Befehle das KKF überhaupt kennt. Wie schon oben erwähnt, können wir mit **VOCS** eine Liste der verfügbaren Vokabulare erhalten und dann für die einzelnen Vokabulare mit **WORDS** die Befehlsliste selbst erhalten.

```
vocs FORTH  ok
forth words
  BOOT              'BOOT             COLD              'COLD
  ABORT             'ABORT            BYE               'BYE
  -IDENT            IDENT             SAVESYSTEM        SAVE
  EMPTY             (ERRORHANDLER     'ERROR            ERRORTEXT@
  >ERRORTEXT        STANDARD-IO       (DISC             (INPUT
  (OUTPUT           (MFREE            (MRELOC           (MALLOC
  LWFILL            LFILL             LMOVE             LCMOVE>
  LCMOVE            L2!               L2@               L!
  L@                LC!               LC@               SS@
  DS@               CS@               D>PTR             PTR>D
  QUIT              EVALUATE          INTERPRET         ]
  [                 PARSER            MULTITASK         SINGLETASK
  PAUSE             FORGET            (FORGET           HCLEAR
  REMOVE            'REMOVE           ENDCASE           ENDOF
  OF                CASE              NEXT              FOR
  +LOOP             LOOP              DO                ?DO
  REPEAT            UNTIL             WHILE             BEGIN
  THEN              ELSE              IF                RECURSE
  >RESOLVE          >MARK             <RESOLVE          <MARK
  UNLOOP            ?LEAVE            LEAVE             NEXTBRANCH
  ?BRANCH           BRANCH            UTABLE:           UVECTOR
  IS                DEFER             LABEL             >LABEL
  ALIAS             VOCABULARY        USER              VDOES>
  DOES>             (;CODE            ;                 :
  2CONSTANT         CONSTANT          2VARIABLE         VARIABLE
  VCREATE           CREATE            HEADERS           -HEADERS
  |                 INDIRECT          RESTRICT          IMMEDIATE
  REVEAL            HIDE              ASCII             \NEEDS
  \\                \                 (                 .(
  ."                "                 [']               '
  NUMBER,           NUMBER            2LITERAL          LITERAL
  POSTPONE          [COMPILE]         COMPILE           FIND
  (FIND             -$,               $,                >$,
  ?NAME             -WORD             WORD              LIST
  .S                DUMP              LDUMP             WORDS
  FILES             FILE.             VOCS              ORDER
  ONLYFORTH         FORTH             SEAL              ALSO
  DEFINITIONS       >NEXTTASK         L>NAME            N>LINK
  LINK>             BODY>             NAME>             >LINK
  >BODY             >NAME             CAPACITY          INCLUDE
  LOADFROM          MORE              DELETE            MAKE
  OPEN              (OPEN             CLOSE             -->
  THRU              LOAD              BLOCK             BUFFER
  UPDATE            FLUSH             EMPTY-BUFFERS     SAVE-BUFFERS
  (FILE-NEXT        (FILE-FIRST       (FILE-FREE        (FILE-WRITE
  (FILE-READ        (FILE-POS@        (FILE-POS!        (FILE-SIZE
  (FILE-CLOSE       (FILE-OPEN        (FILE-DELETE      (FILE-CREATE
  (FILE?            NUMBER?           CONVERT           >NUMBER
  D?                U?                ?                 .
  .R                U.                U.R               0U.R
  D.                D.R               #>                SIGN
  #S                #                 HOLD              <#
```

```
HEX             DECIMAL         BINARY          .BLK
.STATE          STOP?           INDENT          .ID
-TYPE           SPACES          SPACE           EXPECT
QUERY           EDITSTRING      STRING          KEY
KEY?            AT?             AT              MAXAT
CLS             BELL            DEL             CR
TYPE            EMIT            EMIT?           ABORT"
ERROR"          NODEFER         NOTFOUND        ?BLK
?OPEN           ?PAIRS          ?ALLOT          ?DEPTH
?STACK          ?ERROR          ERROR           CELLS
CELL+           CHARS           CHAR+           HALIGN
VALIGN          ALIGN           ALIGNED         H,
HC,             V,              VC,             ,A
,               C,              HALLOT          VALLOT
ALLOT           P!              P@              PC!
PC@             >VADDR          UPPER           UPC
BOUNDS          COUNT>0         COUNT           >$
/$              /STRING         -TRAILING       SCAN>
SKIP>           SCAN            SKIP            PUSH
OFF             ON              -!              +!
BLANK           ERASE           FILL            MOVE
CMOVE>          CMOVE           2!              2@
!               @               C!              C@
*/              */MOD           MOD             /
/MOD            M/              M/MOD           *
M*              UM/MOD          UM*             M-
M+              DWITHIN         DUMIN           DUMAX
DMIN            DMAX            DU>             DU<
D>              D<>             D=              D<
D0>             D0<>            D0=             D0<
DABS            ?DNEGATE        DNEGATE         D-
D+              WITHIN          UMAX            UMIN
MAX             MIN             U>              U<
>               <>              =               <
0>              0<>             0=              0<
1+              2+              1-              2-
ABS             ?NEGATE         NEGATE          -
+               ASHIFT          SHIFT           FLIP
DU2/            D2/             D2*             U2/
2/              2*              NOT             XOR
OR              AND             D>S             S>D
2NIP            2DROP           NIP             DROP
CASE?           ?DUP            2TUCK           2OVER
2DUP            TUCK            OVER            DUP
-2ROT           2ROT            2SWAP           -ROT
ROT             SWAP            ROLL            PICK
DCLEAR          'DCLEAR         DEPTH           SP!
SP@             0=EXIT          ?EXIT           EXIT
PERFORM         EXECUTE         J'              J
I'              I               2RDROP          2R@
2R>             2>R             RDROP           R@
R>              >R              RCLEAR          'RCLEAR
RDEPTH          RP!             RP@             CAPS
>HEAP?          LAST            CODE?           NEXT-LINK
INDENT?         FILE-LINK       FILE-FCB        FILE-ID
UFLAG           SFLAG           ERRORHANDLER    DISC
OUTPUT          INPUT           DPL             HLD
BASE            #TIB            TIB             >TIB
SPAN            R#              SCR             >IN
BLK             CONTEXT         CURRENT         STATE
R0              S0              WDP@            SYSVARLEN@
(SYSVAR         SYSVAR          SYSCON          MAXTLEN@
TASKS           TASK0           TASK-LINK       TLEN
TASKADDR@       TASKADDR        TDP             HEAP?
HLEN            HEAP            HDP             VLEN
VHERE           VDP             PAD             HERE
```

```
DP              VOC-LINK        LIMIT           FIRST
#TIB-MAX        #C/L            #L/BLK          #B/BLK
#RW             #WO             #RO             #BRK
#ESC            #CR             #DELOUT         #DELIN
#BL             FALSE           TRUE            NOOP
```

Nur keine Angst, diese 492 Befehle muß man nicht alle kennen, um mit KK-FORTH zu arbeiten. Aber wie man sieht, sind alle schon oben im allgemeinen FORTH-Kurs erwähnten Befehle hier zu finden. Bei vielen, jetzt noch nicht erklärten Wörter ist die Funktion aus dem Namen zu erraten. Der Rest wird dann entweder schon in den folgenden Beispielen verwendet und auch erklärt oder kann im Glossar nachgelesen werden.

Bei der Namensgebung wurde versucht, eine einheitliche Richtlinie zu finden. Jedoch ist dies nicht immer möglich, da leider der FORTH83-Standard bei vielen Funktionen den Namen vorschreibt. Die einzige Ausnahme ist **#BL** , daß das in FORTH83-Programmen verwendete **BL** ersetzt.

- In den Befehlen werden keine Unterstriche verwendet
- Konstanten werden mit # begonnen
- Zeiger beginnen mit >
- Die meist mit NOOP belegten DEFER-Wörter beginnen mit '
- Befehle, die mit ? beginnen, erwarten meist ein Flag
- Befehle, die mit ? enden, liefern ein Flag oder sind Variablen

2.3.4 Das Arbeiten mit FORTH-Screenfiles

Zum Abschluß des FORTH-Grundkurses haben wir noch kurz etwas über die Standardbefehle zur Filebearbeitung gehört. Auch im KK-FORTH ist das Editieren, Abspeichern und Laden von Programmteilen möglich. Dabei wurde in Anlehnung an den Standard folgendes festgelegt:

- Ein FORTH-Programm arbeitet nur mit einem (Screen-)File
- Files können in Blöcke von 1024 Zeichen bearbeitet werden

Dazu hat das KKF einen Diskpuffer mit 1026 Zeichen ab einer Speicheradresse, deren Anfang durch **FIRST** geliefert wird. Die erste Zelle dieses Bereiches enthält die Nummer des momentan in dem Diskpuffer gespeicherten Screens. Der Befehl **UPDATE** setzt das Bit 15 dieses Wertes und markiert damit dieser Screen als geändert. Dadurch wird das KKF gezwungen, vor dem Laden anderer Screens diesen Block zurückzuschreiben. Eine Ausnahme ist der Wert 32767 in **FIRST** , denn er markiert einen leerer Diskpuffer. Dieser Block kann dann natürlich nicht "update"d werden. Es aber trotzdem möglich, diesen Speicherbereich mit **BLOCK** anzufordern und für eigene Zwecke zu mißbrauchen.

Erzeugen, und Vergrößern von Files

Zuerst muß man dem KK-FORTH mitteilen, daß man mit einem File arbeiten will. Existiert dieses File schon (wie z.B. FFT.SCR) so genügt ein:

```
OPEN Filename                    ( z.B.   OPEN FFT.SCR )
```

Es muß immer die Extension des Filenamens angegeben werden. Ist das File verfügbar, so wird dessen Name als neuer FORTH-Befehl in das Dictionary aufgenommen. Ab jetzt genügt dann zum Öffnen des Files die Eingabe des Filenamen.

Sollte mit einem neuen File gearbeitet werden, so muß zuerst mit

```
MAKE Filename                    ( z.B.   MAKE DEMO.SCR )
```

das entsprechende File erzeugt werden. Dies geht natürlich nur auf beschreibbaren Medien wie RAM-Floppy, Disketten oder Harddisks. Gleichzeitig wird wie bei **OPEN** , der Filename als Befehl verfügbar und das File zum Bearbeiten geöffnet. Die Länge des Files wird dabei auf 0 gesetzt und deshalb ein schon vorhandenes, gleichlautendes File gelöscht. Um nun mit diesem File zu arbeiten, muß es auf die gewünschte Länge gebracht werden.

Ein für das KK-FORTH verwendete File sollte immer ein Vielfaches der Screenlänge von 1024 Zeichen haben. Mit dem Befehl

MORE (u --)

kann das File um u*1024 Zeichen verlängert werden. Die zusätzlichen Screens sind für den Editor vorbereitet und deshalb mit Leerzeichen belegt. Ein File kann normalerweise vom KK-FORTH aus nicht verkleinert werden.

Eine dritte Möglichkeit zum Öffnen von Diskfiles bietet der Befehl **(OPEN** .

(OPEN (fcb --)

Dieser Befehl trägt aber den Filenamen nicht in das Dictionary ein. Er sollte nur zum Arbeiten mit Datenfiles verwendet werden, da bei einem mit **(OPEN** geöffneten File der Name nicht extra gespeichert wird und deshalb überschrieben werden könnte. Dies würde während dem verketteten Laden bei Rückkehr zum (OPEN-File zu einer Fehlermeldung führen.

Auch die Ladeanweisung `INCLUDE Filename` öffnet das File und trägt dessen Namen als Befehl ins Dictionary ein. Dabei merkt es sich das aktuelle File. Nach dem Laden wird dann das INCLUDE-File geschlossen und das zuvor bearbeitete File wieder geöffnet. Der Befehl ist schachtelbar und erlaubt damit ein verkettetes Laden ganzer Programme durch Aufruf eines Files. Darin stehen nur eine Reihe von INCLUDE-Anweisungen. Bei einem Fehler bleibt das gerade bearbeitete File offen und kann sofort editiert werden. Jedoch ist die Information, von welchem File aus das aktuelle File geöffnet wurde, unwiederbringlich verloren.

In allen Fällen öffnet das KK-FORTH ein File und merkt sich sowohl den Filename (in **FILE-FCB**) als auch die Handlenummer (in). Da normalerweise keine Ausgabe auf einem Gerät mit der Handlenummer 0 erfolgt (währe aber beim PC möglich), wird der Wert 0 als Flag für ein geschlossenes Screenfile verwendet.

Arbeiten mit Files

Alle im FORTH83-Standard definierten Befehle wie **BUFFER** , **BLOCK** , **UPDATE** , **FLUSH** und **SAVE-BUFFERS** können auf ein geöffnetes File angewandt werden. Jedoch muß man dabei immer bedenken, daß nur ein einziger Diskpuffer im System existiert. Bei Übertragung zwischen zwei Screens oder zwischen zwei Files muß deshalb der FORTH-Speicher (z.B. der Bereich zwischen `HERE` und `VDP @`) als Zwischenablage verwendet werden.

Schließen eines Files

Im KK-FORTH wird immer nur mit einem Screenfile gearbeitet. Beim Öffnen eines anderen Files mit **OPEN** , **(OPEN** , **MAKE** oder **INCLUDE** wird automatisch das aktuelle File geschlossen. Auch das Löschen des Filenameneintrags im Dictionary, ein Neustart mit **COLD** oder das Verlassen des Programmes veranlaßt das Schießen des zugehörigen Files. Man kann mit dem Befehl **CLOSE** auch das File schließen, ohne gleichzeitig ein anderes zu öffnen.

Falls es einmal zu Problemen bei der Speicherung geänderter Screens kommt oder das Terminalprogramm während der Bearbeitung eines Files verlassen wurde, so können die Daten

des Diskpuffers nicht mehr zurückgeschrieben werden. Danach weigert sich das KK-FORTH, das Programm zu verlassen oder das File zu schließen. Falls man auf die Daten des geänderten Screen verzichtet kann, so kann man das FORTH überlisten. Es muß sowohl der Diskpuffer als auch das Flag für ein geöffnetes File löscht werden. Dies sollte aber wirklich nur als Notausstieg verwendet werden.

```
empty-buffers  file-id off
```

Löschen von Files

Manchmal ist es notwendig, ein bestehendes File zu löschen. Der nicht ganz ungefährliche Befehl dazu ist:

```
DELETE Filename
```

Ohne Rückfrage wird das erste File mit gleichlautenden Name gelöscht. Dabei sollte der Filename möglichst exakt angegeben werden, da vom Betriebssystem oft Wildcards ("?" und "*") erlaubt werden.

2.3.5 Der KKF-Screeneditor

Nachdem wir jetzt wissen, wie man Files öffnet und vergrößert, könnte sofort mit der Programmierung begonnen werden. Jedoch hat man hier das Problem, daß niemand freiwillig hergeht und alle Zeichen eines Programmes mit **C!** in die Screenfiles schreibt. Auch die Verwendung anderer Editoren des Betriebssystems ist ausgeschlossen, da daß zugegebenermaßen eigenwillige Fileformat nicht unterstützt wird.

Aus diesem Grunde wird ein für das KK-FORTH ladbarer Editor mitgeliefert. Bei den EMUF-Versionen stehen die beiden Hauptbefehle `L` und `V` schon im EPROM-Kern zur Verfügung, da sie relativ kurz sind und den integrierten Screeneditor des Terminalprogrammes aufrufen. Falls also die Befehlsfolge `' l drop` mit einer Fehlermeldung beantwortet wird, sollte man sich den Editor z.B. mit der Befehlsfolge `Include PC_EDIT.SCR` nachladen.

L	(n --)	(Editiere Screen n)
V	(--)	(Editiere letzte Fehlerposition)

Sobald man eine dieser beiden Befehle eingibt, entsteht nachfolgendes Bild auf dem Display. Natürlich muß man vorher noch (wie hier mit `OPEN G_BIOS.SCR`) das gewünschte File öffnen.

```
Ú--G_BIOS.SCR-----------------------------------------20:01:19-¿
³Screen:000 y=00 x=00  (L:000/C:000)  (O)                      ³
Ã-------------------------------------------------------------´
³\\ PC-Grafik mittels BIOS-Routinen              ( 22.05.91/KK ) ³
³                                                               ³
³System:         KKF_PC V1.2/0                                  ³
³Änderung:       22.05.91                                       ³
³                                                               ³
³                    Hinweise:                                  ³
³                                                               ³
³ - Verwendet INT $10 für die Grafikbefehle                     ³
³ - Benützt einige BIOS-Variablen ab Adresse $0000:$0400        ³
³ - Unterstützt alle Grafikmodes                                ³
³                                                               ³
³                                                               ³
³                                                               ³
³                                                               ³
³                                                               ³
³                                                               ³
À-------------------------------------------------------------Ù
```

Da hier nur das File angesehen werden soll, benötigt man nur die Tastenkombination CTRL+C bzw. CTRL+R um in dem File die nächsten Screens anzusehen. Falls vorhanden, wird mit der ESC-Taste dann der Editor verlassen. Oft werden, wie in der PC-Version, auch noch weitere Tasten unterstützt. Da diese aber vom verwendeten Rechner abhängen, muß ich hier auf die Beschreibung der Tastenbelegung in der speziellen KKF-Beschreibung verweisen.

2.3.6 Schreiben und Testen von Programmen

Das nachfolgende Beispiel soll den vollständigen Ablauf zur Erstellung eines Autostartprogrammes aufzeigen. Um dabei von der Hardware unabhängig zu sein, wird in dem Beispiel nur die Tastatur abgefragt und in Abhängigkeit der gedrückten Taste ein Text ausgegeben. Die dabei verwendete CASE-Struktur wird noch im Kapitel 2.4 besprochen.

System starten und Einschaltmeldung abwarten

Wie das geht, wurde schon oben beschrieben. Für dieses Beispiel wird auch der Editor benötigt und muß deshalb bei einigen Versionen nachgeladen werden. Da hier kein Platzproblem existiert, kann auch das meist mit KKF.COM bezeichnete Programm mit geladenen Assembler, Disassembler, Debugger und Editor verwendet werden.

File öffnen und Programm eingeben

```
Make AUTO.SCR                   ( File anlegen )
2 more                          ( und auf 2 Screens vergrößern )
0  l                            ( Screen 0 editieren )
```

Geben Sie jetzt das folgende Programm ein. Dabei kann mit der Tastenkombination CTRL+C bzw. CTRL+R der Screen gewechselt werden. Screen 0 dient nur als Kommentar und Screen 1 enthält das Programm. Wie der Editor verlassen wird (meist mit ESC) oder welche Tasten noch verwendet werden können, ist in den einzelnen KKF-Beschreibungen nachzulesen.

```
Ú--AUTO.SCR-----------------------------------------18:34:19-¿
³Screen:000 y=00 x=00  (L:000/C:000)  (O)                     ³
Ã-------------------------------------------------------------´
³\\ Autostart-Beispielprogramm                                ³
³                                                             ³
³System:        KK-FORTH V1.2/0                               ³
³Erstellt:      07.06.91  Klaus Kohl                          ³
³                                                             ³
³                   Hinweise:                                 ³
³                                                             ³
³ - Programm erwartet folgende Tasten:                        ³
³     W   =  Ausgabe der Wortliste                            ³
³   1...3 =  Anzeige der gedrückten Taste                     ³
³     Q   =  Zurück zum FORTH-Interpreter                     ³
³                                                             ³
³                                                             ³
³                                                             ³
³                                                             ³
³                                                             ³
À-------------------------------------------------------------Ù
```

```
Ú--AUTO.SCR-----------------------------------------18:40:42-¿
³Screen:001 y=00 x=00  (L:000/C:000)  (O)                    ³
Ã-------------------------------------------------------------´
³\ auto                                                      ³
³  : auto   ( -- ) ( Autostart-Programm )                    ³
³    cr cr cr  ." Bitte Taste ( W, 1..3 oder Q ) drücken" cr  ³
³    BEGIN  key  upc                                         ³
³       CASE  Ascii W  OF  cr words cr                ENDOF  ³
³             Ascii Q  OF  cr  quit                   ENDOF  ³
³             Ascii 1  OF  cr ." Taste 1 gedrückt" cr ENDOF  ³
³             Ascii 2  OF  cr ." Taste 2 gedrückt" cr ENDOF  ³
³             Ascii 3  OF  cr ." Taste 3 gedrückt" cr ENDOF  ³
³       ENDCASE                                              ³
³    REPEAT ;                                                ³
³                                                            ³
³                                                            ³
³                                                            ³
³                                                            ³
³                                                            ³
À-------------------------------------------------------------Ù
```

File laden und danach schließen

```
1 load > ok                    ( File laden )
close  ok                      ( File schließen )
```

Da in dem Befehl **AUTO** wieder zur FORTH-Hauptschleife zurückgekehrt, kann der Befehl auch vor der Speicherung noch getestet werden:

```
auto

Bitte Taste ( W, 1..3 oder Q ) drücken
                               ( Taste 1 drücken )
Taste 1 gedrückt
                               ( Taste 2 drücken )
Taste 2 gedrückt
                               ( Taste 3 drücken )
Taste 3 gedrückt
                               ( Taste W drücken )

  AUTO                AUTO.SCR          FROM                -V
  V                   L     ...         ( Abbruch mit CTRL+X )
                                        ( Taste Q drücken )

.s                                      ( Datenstack testen )
(empty)  ok
```

Falls noch Fehler im Programm sind oder wenn weitere Tasten abgefragt werden sollen, so kann nach dem Löschen des Befehls erneut geladen und getestet werden.

```
Forget auto                    ( Befehl löschen )
auto.scr                       ( File erneut öffnen )
1 l                            ( Befehl mit dem Editor verändern )
...
```

Erstellen eines Autostart-Programmes

```
' auto  is 'abort              ( Warmstart-Zeiger ändern )
Savesystem  AUTO.COM           ( Programm oder Bitimage speichern )
```

Falls das KK-FORTH aus dem EPROM gestartet wurde, ist nun das KKF-Image in ein neues EPROM zu Brennen und dann das im aktuellen Verzeichnis gespeicherte Image AUTO.COM ab einer in der Zusatzdokumentation angegebenen Speicheradresse abzulegen. Beim Einschalten müßte dann sofort das kleine Programm aufgerufen und die Tasten W, Q oder 1...3 abgefragt werden.

Bei den anderen KKF-Versionen (z.B. beim PC) ist in AUTO.COM das vollständige FORTH-System enthalten. Es wird sich deshalb nach dem Laden genauso wie das geänderte EPROM verhalten.

Nach dem Drücken der Taste Q befindet man sich wieder im Interpretermodus. Der neuen Befehl **AUTO** ist aber jetzt ein fester Bestandteil des KK-FORTH. Da auch die Veränderung des Vektors '**ABORT** noch besteht, führt jedes **COLD** oder **ABORT** wieder zur Ausführung von **AUTO** .

2.4 Besonderheiten des KK-FORTH

Mit den folgenden Kapitel werden dann die noch nicht angesprochenen Möglichkeiten des KK-FORTH durchgegangen, wobei immer die Verwendbarkeit im Vordergrund steht. Sollten dabei noch Fragen offenbleiben, so kann im Kapitel 3 die Implementierung dieser Besonderheit nachgelesen werden.

2.4.1 Kontrollstrukturen

Wir haben schon die wichtigsten Kontrollstrukturen beim FORTH-Kurs kennengelernt. Hier beschäftigen wir uns deshalb mit den zusätzlichen Befehlen und mit der Implementation eigener Strukturen im KK-FORTH.

Rekursion in FORTH-Programmen

In anderen Programmiersprachen wie PASCAL oder C werden häufig Befehle definiert, die sich selbst aufrufen. Typisches Beispiel ist die noch folgende Fakultätsberechnung oder die Suche in binären Bäumen.

Da normalerweise ein FORTH-Befehl während der Definition nicht im Dictionary vertreten ist und deshalb nicht aufgerufen werden kann, gibt es **RECURSE** . Dieser Befehl veranlaßt, daß wie beim Aufruf anderer Wörter die Rückkehradresse auf dem Returnstack gespeichert und dann an den Anfang des Befehls gegangen wird.

```
: fakultät   ( n -- u ) ( Fakultät bis n=8 )
   ?dup IF dup 1- recurse * ELSE 1 THEN ;

5 fakultät u. 120  ok
8 fakultät u. 40320  ok          ( Größter erlaubter Wert )
```

Dabei muß man darauf achten, daß weder Datenstack noch Returnstack überladen wird. Das KK-FORTH läßt meistens bis zu 256 Werte auf den Stacks zu.

Die FOR ... NEXT - Schleife

Nicht aus dem FORTH83-Standard stammt die **FOR** ... **NEXT** -Schleife. Sie wurde meines Wissens zuerst von Charles Moore in seinem CM-FORTH für den FORTH-Chip NC4000 definiert. Der Rücksprung von **NEXT** auf das **FOR** wird dabei (genauso wie beim RTX-2000) in einem Takt durchgeführt. Dazu wird bei **FOR** der oberste Datenstackwert mit **>R** zum Returnstack gebracht und kann von dort mit **R@** abgefragt werden. Bei **NEXT** wird dann der Returnstackwert erniedrigt und zurückgesprungen, falls er ungleich 0 war. Bei 0 wird der Returnstack wieder bereinigt und dann das Programm hinter dem **NEXT** fortgesetzt.

```
: countdown  10 FOR  r@ .  NEXT ;  ok
countdown 10 9 8 7 6 5 4 3 2 1 0  ok
```

Die CASE-Struktur

Eine weniger bekannte Fallunterscheidung ist die CASE-Struktur. Es gibt leider sehr viele unterschiedliche Realisierung im FORTH. Die im KK-FORTH implementierte CASE-Struktur nach Charles hat dabei den größten Verbreitungsgrad. Ihr wird ein 16Bit-Wert bei **CASE** übergeben und der Reihe nach mit mehreren Werten verglichen. Nur bei Übereinstimmung verschwindet der übergebene Wert vom Datenstack. Nach Ausführung der Befehle zwischen dem entsprechenden **OF** ... **ENDOF** wird das Programm hinter **ENDCASE** fortgesetzt. Es ist zu beachten, daß der Befehl **ENDCASE** den noch vorhandenen Vergleichswert vom Datenstack entfernt.

Typisches Beispiel für eine CASE-Abfrage ist, wie beim Autostart-Beispiel die Tastenabfrage. Falls dort noch alle anderen Tasten mit einer Fehlermeldung quitiert werden sollen, so kann dies zwischen dem letzten **ENDOF** und **ENDCASE** geschehen:

```
3            Ascii 2  OF  cr ." Taste 2 gedrückt" cr  ENDOF     3
3            Ascii 3  OF  cr ." Taste 3 gedrückt" cr  ENDOF     3
3            cr ." Taste '"  dup emit  ." ' gedrückt" cr        3
3       ENDCASE                                                 3
3    REPEAT ;                                                   3
```

Aufbau der Kontrollstrukturen

Alle Befehle zu Kontrollstrukturen sind sowohl **IMMEDIATE** als auch **RESTRICT** . IMMEDIATE-Befehle werden, wie schon oben erklärt, auch in :-Definitionen ausgeführt. "RESTRICT" bedeutet dagegen, daß dieser Befehl nur innerhalb von :-Definitionen aufgerufen werden darf.

Durch diese Kombination ist es den Befehlen möglich, ohne weitere Statusabfragen eigene Teile zu kompilieren und dabei auch Parameter auf dem Datenstack abzulegen. Diese Parameter setzen sich zusammen aus (Rücksprung-)Adressen und Zusatzflags. Die Flags werden dann von anderen Kontrollbefehlen auf Richtigkeit überprüft und erst dann die Adressen verarbeitet.

Folgende Strukturen sind im KK-FORTH definiert. Alle in geschweiften Klammern eingeschlossenen Befehle können dabei fast beliebig oft oder auch überhaupt nicht innerhalb der entsprechenden Kontrollstruktur auftreten. Kontrollstrukturen dürfen ineinander geschachtelt sein, wobei die Punkte ("...") den Einsetzpunkt dafür darstellt. Es dürfen auch beliebig viele FORTH-Befehle an Stelle von ... eingesetzt werden. Bis auf die **(?)DO** ... **(+)LOOP** - und **FOR** ... **NEXT** -Schleifen können alle Wörter auch innerhalb von Schleifen mit **EXIT** verlassen werden. Dabei kann aber (wie bei **CASE**) noch ein Parameter auf dem Datenstack verbleiben. **DO** ... **LOOP** -Schleifen und ihre Verwandten legen drei und die **FOR** ... **NEXT** -Schleife einen 16Bit-Werte auf dem Returnstack ab, die vor **EXIT** zuerst mit **UNLOOP** bzw. **RDROP** entfernt werden müßten.

```
Flag  IF .. THEN
Flag  IF .. ELSE .. THEN
BEGIN ... { Flag  WHILE } ... Flag  UNTIL
BEGIN ... { Flag  WHILE } ... REPEAT
Ende+1 Anfang  DO ... { LEAVE } ... { Flag  ?LEAVE } ... LOOP
Ende+1 Anfang  DO ... { LEAVE } ... { Flag  ?LEAVE } ... n  +LOOP
Ende+1 Anfang  ?DO ... { LEAVE } ... { Flag  ?LEAVE } ... LOOP
Ende+1 Anfang  ?DO ... { LEAVE } ... { Flag  ?LEAVE } ... n  +LOOP
Ende  FOR ... NEXT
Wert  CASE ... { n  OF ... ENDOF } ... ENDCASE
```

Grundbefehle zu Kontrollstrukturen

Manchmal fällt es schwer, sich die Bedingung zu merken, unter der eine Schleife verlassen wird. Deshalb hier einen kleinen Einblick in das FORTH. Es sind im FORTH83-Standard nur zwei Sprungbefehle zum Fortsetzen eines Programmes an anderer Stelle definiert:

BRANCH Bedingungslos, daß heißt ohne Übergabe von Flags, springt der Befehl **BRANCH** . Er wird von **ELSE** , **ENDOF** und **REPEAT** verwendet.

?BRANCH Als bedingter Sprung erwartet **?BRANCH** ein Flag auf dem Datenstack. Ist dieser Wert 0, so wird zu einem anderen Programmteil verzweigt. Verwendet wird **?BRANCH** von **IF** , **WHILE** und **UNTIL** .

Im KK-FORTH ist dagegen neben den nicht sichtbaren Runtime-Routinen zu **LOOP** , **+LOOP** ein weiterer, auch den Anwender zugängiger Befehl definiert.

NEXTBRANCH Dieser Befehl wird von **NEXT** verwendet und springt nach Erniedrigung des obersten Returnstackwertes zurück, falls der Wert ungleich 0 war. Bei 0 wird statt dessen der Wert vom Returnstack entfernt und das Programm dahinter fortgesetzt.

Auffallend in dieser Zusammenfassung ist, daß **BEGIN** , **CASE** , **ENDCASE** und **THEN** keine Sprungbefehle erzeugen, sondern lediglich als Vor- bzw. Rückwärtsreferenzen dienen oder Sprungadresse korrigieren.

Verwendung der Grundbefehle für eigene Strukturen

Alle Sprungbefehle brauchen zur Ausführung noch eine Adreßangabe. Jedoch ist die Implementierung von Sprungbefehlen in den einzelnen KKF-Versionen so unterschiedlich, daß keine Angaben darüber hier gemacht werden kann. Statt dessen gibt es eine Regel und zusätzliche Befehle zur Ablage der Adresse:

Da nicht definiert ist, ob die Sprungbefehle "IMMEDIATE" sind, dürfen sie nur mit der Befehlsfolge `POSTPONE ...` kompiliert werden. Die Vor- oder Rückwärtssprünge werden dann mit **>MARK** oder **<MARK** markiert und danach mit **>RESOLVE** oder **<RESOLVE** aufgelöst.

Dadurch ergeben sich folgende Befehlskombinationen:

Vorwärtssprung:	`postpone branch >mark`
Auflösung:	`>resolve`
Rückwärtssprung:	`<mark`
Auflösung:	`postpone branch <resolve`
Bedingter Vorwärtssprung:	`postpone ?branch >mark`
Auflösung:	`>resolve`
Bedingter Rückwärtssprung:	`<mark`
Auflösung:	`postpone ?branch <resolve`
FOR-Schleifenanfang:	`postpone >r <mark`
NEXT-Schleifenende:	`postpone nextbranch <resolve`
OF-Strukturteil:	`postpone ofbranch >mark`
ENDOF:	`postpone branch >mark swap <resolve`

In der letzten Zeile wird durch das **SWAP** verraten, daß **<MARK** und **>MARK** nur eine Adresse auf dem Datenstack ablegen. **>RESOLVE** und **<RESOLVE** nehmen dann diese Adresse und die aktuelle Dictionaryposition **HERE** zur Ermittlung des zu kompilierenden Wertes.

Im KK-FORTH werden neben den Adressen noch Flags verwaltet. Welche Flags für die einzelnen Strukturen notwendig sind, ist auch im Glossar oder im Kapitel 3 nachzulesen. Da ein optimierender Kompiler daran gehindert werden muß, Befehle über Kontrollwörter wie **ELSE** hinweg zusammenzufassen, wird die Variable **CODE?** durch die Befehle **<MARK** bis **>RESOLVE** auf 0 gesetzt und damit die Optimierung abgeschaltet.

2.4.2 Speicherbelegung

Bis jetzt haben wir im gesamten Handbuch nocht nichts darüber erfahren, wohin überhaupt die Eingaben gehen, wo die Stacks aufbewahrt werden und wohin man Programme kompiliert. Um mit KKF optimal arbeiten zu können, sollte man zumindest den groben Aufbau kennen.

Das KK-FORTH der Version 1.2 geht davon aus, daß dem FORTH-Kern nur ein Speicher von bis zu 64KByte zugängig ist. In diesem Speicher wird dann je nach Prozessor und verwendeter

Hardware sowohl das Dictionary (im EPROM und/oder am RAM-Anfang) als auch der Variablen-, Heap- , Task-, Disketten- und Arbeitsbereich (am Ende des RAM's) abgelegt. Die folgende Aufteilung entstammt dem KKF_PC V1.2/0, entspricht jedoch mit wenigen Änderungen auch anderen KKF-Versionen.

Adresse	Zeiger	Konstanten	Speicherbereiche
$0000		LIMIT	Ende+1
-----	--------	--------	----------
			Kopie der Systemvariablen
$FFB0		(SYSVAR	----------
			Anwender-Arbeitsspeicher (hier leer)
$FFB0	UWORK		----------
			System-Arbeitsspeicher
$FD20	SWORK		----------
			Diskpuffer
$F91E		FIRST	----------
			Taskbereich
$F26C	TDP		----------
			Heap-Bereich
$F26C	HDP	HEAP	----------
			Variablenbereich
$F228	VDP		----------
			freier Programmspeicher
$418F	DP	HERE	----------
			Dictionary
			Systemvariablen
$0110		SYSVAR	----------
			Systemkonstanten
$0100			----------

PC-Programme mit der Endung .COM werden immer ab der Speicheradresse $0100 geladen und dann gestartet. Deshalb beginnt das KKF auch erst an dieser Adresse. Da aber 64KByte zur Verfügung stehen, bleiben dem Anwender hier noch $B099 ($F228 - $418F = 45209) Bytes zu freien Verwendung.

Bei den EMUF-Versionen für Z80 wird meistens ein 32KByte-EPROM mit dem FORTH-Kern und den Applikationen am Speicheranfang und ein 32KByte-RAM für das Programm und die Arbeitsbereiche am Speicherende eingesetzt. Hier beginnt dann der freie Programmierspreicher erst bei $8000 und sowohl die Systemvariablen als auch die (NEXT-Routine wird an das RAM-Ende ausgelagert. Beim Start des KK-FORTH werden dann noch $50 Bytes am RAM-Anfang für die Kopie des SYSVAR-Bereiches reserviert. Diese Kopie dient dann auch als Header für Zusatzimages z.B. bei Autostart-Programmen.

Versteckt im Taskbereich sind dann beide Stacks und der Arbeitsbereich für Befehlseingabe und Zahlenausgabe. In den gespeicherten Programmfiles bleiben aber von den hier insgesamt benötigen 1,4Kbyte nur noch die 256 Bytes des USER-Bereiches.

Adresse	Zeiger	Konstanten	Speicherbereiche
$F91E		FIRST	----------
			Sicherheitsbereich
$F918	R0		----------
			Returnstack mit bis zu 256 Werte
			Eingabebereich für QUERY
$F6C8	>TIB	TIB	----------
			USER-Bereich mit 256 Bytes
$F5C6	TASKADDR	TASKADDR@	----------
			Sicherheitsbereich
$F5C0	S0		Datenstack mit bis zu 256 Werten
			String-Speicherbereich
$F3B4		PAD	----------
			Bereich für Zahlenstring-Erzeugung
			Stringadresse für WORD
$F26C	TDP	WDP@	----------

In den meisten FORTH-Systeme wird mit **WORD** der nächste String aus dem Eingabepuffer geholt und nach **HERE** gebracht. Da aber im KK-FORTH eine vollständige Trennung von Dictionary und RAM-Speicher geplant war, wurde für jeden Task ein eigener Arbeitsspeicher

definiert, dessen Anfangsadresse mit dem Befehl WDP@ abgefragt werden kann. **HERE** muß deshalb meistens durch **WDP@** ersetzt werden.

Da hier nur ein Task definiert ist, stimmen die WDP@-Adresse mit der Anfangsadresse des Taskbereiches überein. Die hier definierte Größe der einzelnen Bereiche kann durch direkte Manipulation der entsprechenden Werte im Taskbereich verändert werden. Sie wird aber erst nach der Speicherung und dem Neustart des FORTH aktiviert. Mehr dazu finden Sie bei der Beschreibung des Task-Bereiches (Kapitel 3.2.3).

2.4.3 Die Verwendung des Variablenbereiches

In normalen FORTH-Systemen werden Variablen, Zeiger für DEFER-Befehle und Vokabular-Verkettungen im Dictionary aufbewahrt. Da aber das KK-FORTH auch in ROM-Version ausgeliefert wird, mußten hier einige Änderungen eingebaut werden. Jedoch sind diese Änderungen so gering, daß man unter Beachtung einer Besonderheit praktisch keine Probleme damit bekommt.

Variablen

Bei Variablen werden im Dictionary neben dem Befehlsheader und der Codefeldadresse nur der Offset in den Variablenbereich aufbewahrt. Addiert man zum Offset den aktuellen Wert der Systemvariable **VDP** , so erhält man die eigentliche Speicheradresse der Variable.

In der Praxis ist aber deswegen keine Programmänderung notwendig, da der Aufruf einer Variable die richtige Adresse liefert. Da aber bei Definition weiterer Variablen und bei Veränderung der Heap- oder Taskgröße diese Adresse verändert wird, dürfen die gelieferten Adressen nicht als Literals oder Konstanten verwendet werden.

In Assemblerprogrammen sollte immer nur der Offset der Variable (`Variablenadresse VDP @ -`) gespeichert und dann zum aktuellen Inhalt der Systemvariablen **VDP** (hat feste Speicheradresse) addiert werden. Bei Interruptprogrammen muß sichergestellt sein, daß nicht gerade ein anderer Task die Speicheraufteilung des Systems verändert.

Vokabulare

Selbst bei Vokabulare, die im ROM definiert sind, können weitere Befehle angehängt werden. Dies wird dadurch erreicht, daß der Zeiger auf das oberste Wort des Vokabulars im Variablenbereich steht und im Dictionary nur die Vokabular-Verkettung und der Offset in den Variablenbereich steht. Auf die Auslagerung der Vokabular-Verkettung wurde verzichtet, da kaum jemand Befehlsteile aus dem EPROM entfernen will.

Damit ergibt sich folgende Verkettung aller Vokabulare und Befehle im KKF_PC:

```
                PRINTER     DISAM     Assembler     FORTH
VOC-LINK  -->   $4FCA   --> $41B1 -->    $2072    --> $0000    Verkettung
                $007A       $0052        $0044        $001E    Offset
```

Der Wert in der Speicheradresse `VDP @ Offset +` ist die Linkfeldadresse des obersten Wortes im angegebenen Vokabular. Mit **L>NAME** kommt man zur Namensfeldadresse. Den Vokabularname bekommt man durch die Befehlsfolge `BODY> >NAME` aus der angegebenen Verkettungsadresse.

DEFER-Befehle

Als Letztes muß man noch die veränderbaren Befehle im Variablenbereich unterzubringen. Wie auch bei Variablen wird dann im Befehl selbst nur noch ein Offset in den Variablenbereich untergebracht und dann die Codefeldadresse des auszuführenden Wortes im Variablenbereich

gespeichert. Die Veränderung der Ausführungsroutine sollte sowieso nicht selbst, sondern immer mit **IS** durchgeführt werden:

```
' meinprg  is  'error            ( Restart des Programmes nach Fehler )
```

Eigene Datenstrukturen im Variablenbereich

Nachdem Programme auch vollständig im EPROM ablaufen können, dürfen die begehrten **CREATE** ... **DOES>** -Strukturen nur unveränderbare Daten speichern. Um aber auch so Datentypen wie Vektoren oder Felder zu verwalten, wurden ähnliche Möglichkeiten auch für den Variablenbereich eingeführt.

Im KK-FORTH existiert ein Befehlssatz zur Manipulation des Variablenbereiches. Er ist bis auf das vorangestellte "V" mit den Dictionarybefehlen identisch, verwaltet aber den Variablenbereich.

VCREATE	(name ; -- addr ; c: --)	Liefert aktuelle Variablenadresse
VDOES>	(-- addr ; c: --)	Es folgt Ausführungsteil
VALLOT	(n --)	Reserviert n Bytes
V,	(n --)	Speichert n als 16Bit-Wert
VC,	(char --)	Speichert char als 8Bit-Wert

Die Definitions eines Vektors mit 3 16Bit-Werten sieht ähnlich wie in anderen FORTH-Versionen aus:

```
0 Constant x
1 Constant y
2 Constant z
: 3D-Vector   ( Definition eines Vektors )
   VCreate  0 v,  0 v,  0 v,   ( mit 0 vorbelegen )
   VDoes>   swap  2*  +  ;     ( n -- addr )

3D-Vector test
: test3*   ( Multiplikation des Vektors )
   x test @  3 *  x test !
   y test @  3 *  y test !
   z test @  3 *  z test ! ;
```

Bei der Erzeugung des Datenfeldes muß man beachten, daß zur Allokierung des Variablenspeichers die Variablen nach unten geschoben werden. Falls man einen Teil des Variablenbereiches nach der Definition löschen will, so sollte der aktuelle Offset geholt (steht in **VLEN**) und erst nach Reservierung des Speichers den Inhalt von **VDP** addiert werden.

```
VCreate meinfeld  vlen @  $100 vallot  vdp @ +  $100 erase
```

Man kann aber auch vom Ende des Variablenbereiches aus rechnen:

```
Vcreate meinfeld  $100 vallot  vhere $100 - $100 erase
```

Man muß bei einigen Prozessoren und KKF-Versionen darauf achten, daß 16Bit-Zugriffe nur ab geraden Speicheradressen möglich sind. Dies wird in KK-FORTH durch **VALIGN** in eigenen Programmen unterstützt und beim Anlegen neuer Variablen oder beim Verschieben des Variabenbereiches automatisch beachtet. Es kann dann vorkommen, daß zwischen Ende des Variablenbereiches und dem Anfang des Heap ein unbenutztes Byte steht.

2.4.4 Der HEAP und INDIRECT-Befehle

Manchmal ist es sinnvoll, dem Anwender eines Programmes nicht alle Befehle zur Verfügung zu stellen oder Teile von FORTH (wie z.B. den Assembler) für die Programmerstellung zu Laden und danach wieder zu Entfernen. Ein spezielles Konzept, HEAP genannt, dient bei der Kompilierung

zur Ablage der Wortheader. Der Vorteil der Platzeinsparung durch Löschen von Befehlsnamen und deren Verkettungszeiger ist bei einem 64KByte-Programmspeicher nicht zu verachten. Beispielsweise benötigt eine Variable nur noch 6 Bytes an Stelle den sonst beanspruchten 10..40 Bytes (je nach Namenslänge).

Der Heap liegt im Speicherbereich zwischen dem Variablen- und dem Taskbereich. Beim Ablegen eines Befehlsheader oder bei Reservierung des HEAP-Speichers wird der Variablenbereich mit allen Werten nach "unten" (zu den niedrigeren Speicheradressen) verschoben. **HCLEAR** , **-n HALLOT** oder **FORGET** (bzw. **REMOVE**) gibt den HEAP-Speicher frei und verschiebt auch den Variablenbereich wieder nach "oben".

Selbst wenn der Befehlsname auf dem Heap gespeichert wird, ist der auszuführende Programmcode im Dictionary abgelegt. Deshalb können derartige Befehle auch nach Löschen des Befehlsnamens (mit **HCLEAR** oder **SAVE**) weiterhin funktionsfähig bleiben. Sie sind nur nicht mehr direkt über Tastatur aufrufbar oder kompilierbar. Solange der Header noch existiert, wird durch ein Flag in der Namensfeldadresse dem FORTH-Kompiler angezeigt, daß es sich bei der im nachfolgenden Codefeld abgelegten Adresse nicht um die Codeadresse, sondern um einen Zeiger auf die eigentliche Codefeldadresse handelt.

Headerlose Worte werden mit folgenden Befehlen verwaltet:

|	(--)	Der nächste Befehl ohne Header
-HEADERS	(--)	Alle nachfolgenden Befehle ohne Header
HEADERS	(--)	Alle nachfolgenden Befehle mit Header
HALLOT	(n --)	Heap um n Bytes verändern (+=größer)
HEAP	(-- addr)	Liefert Adresse des Heap-Speichers
HCLEAR	(--)	Löschen des Heap mit allen Wörtern

Da ein Assembler nur Opcodes in CODE-Definitionen ablegt und deshalb bei der Ausführung der so erzeugten Befehle nicht mehr benötigt wird, kan er vollständig in den Heap geladen werden. Dazu muß man ausreichend Platz im Heap reservieren und während des Laden des Assemblers den Dictionarypointer **DP** dorthin stellen. Danach wird der Rest des Programmes wieder an gewohnter Position abgelegt.

```
here                        ( alte Position merken )
$1400 hallot                ( ca. 5KByte reservieren )
heap dp !                   ( Dictionarypointer ändern )
Include HD64180.ASM         ( Assembler laden )
dp !                        ( alte Dictionarypointer )
...                         ( Hauptprogramm laden )
```

Dabei dürfen auch im Assembler headerlose Worte erzeugt werden. Die Header werden dann unterhalb des Heap abgelegt. Es muß aber darauf geachtet werden, daß der reservierte Platz ausreicht und daß die richtige Dictionaryposition danach wieder gesetzt wird. Man sollte nach der Programmerstellung immer den gesamten Heap mit **HCLEAR** oder **SAVE** löschen.

Da die Zeiger oder Programme im Heap nicht verändert werden können, muß der Heap beim Anlegen zusätzlicher Taskbereiche immer gelöscht sein. Zusätzlich sollte man bei Verwendung des Heap zur Ablage eigener Daten oder 16Bit-Werte mit **HALIGN** dafür sorgen, daß eine gerade Adresse für den Wortzugriff verwendet wird. Falls kein Align notwendig ist, so wird weder Speicher noch Prozessorzeit durch **HALIGN** verschwendet.

2.4.5 Ein-/Ausgabe im KK-FORTH

Die Grundbefehle für Ein- und Ausgabe wurden schon bei der Einführung beschrieben. Jedoch gibt es für das KK-FORTH noch mehr Befehle.

EMIT?	(-- f)	Liefert Flag, ob Ausgabe möglich ist
EMIT	(char --)	Ausgabe eines Zeichens

TYPE	(addr len --)	Ausgabe eines Strings
CR	(--)	Cursor zum Anfang der nächster Zeile
DEL	(--)	Ein Zeichen zurück (mit Löschen)
BELL	(--)	Ton ausgeben
CLS	(--)	Bildschirm löschen
MAXAT	(-- x y)	Anzahl der Zeichen und Zeilen
AT	(x y --)	Cursor setzen (0/0 oben links)
AT?	(-- x y)	Aktuelle Position abfragene
KEY?	(-- f)	Liefert Flag, ob Taste gedrückt wurde
KEY	(-- char)	Holt ein Zeichen (wartet dabei)
STRING	(addr len --)	String ohne Ausgabe holen
EDITSTRING	(addr maxlen pos len -- pos2 len2)	String editieren
QUERY	(--)	Nächste Eingabezeile holen

Neu ist hier vor allem der Befehl **EDITSTRING** . Er ist ein sehr mächtiges Werkzeug für Menüprogramme, da damit ein vorgegebener String editiert werden kann. Dazu wird der Cursor an die gewünschte Bildschirmposition gebracht und dann der Befehl mit folgenden Parametern aufgerufen:

addr	Speicheradresse des Strings
maxlen	Größe des Eingabefeldes
pos	Position des Cursors im Feld (ab 0 gerechnet)
len	Länge des vorhandenen Strings

Es sind alle druckbaren Zeichen und einige schon bei der Eingabe beschriebenen Steuertasten zulässig. Zurückgeliefert werden nach Betätigung der ENTER-Taste dann die aktuelle Cursorposition und die Gesamtlänge des eingegebenen Strings.

Da dieser Befehl auch durch **QUERY** für die Eingabe der Befehlszeilen verwendet wird, hat er eine zusätzliche Besonderheit: Im KK-FORTH gibt es am RAM-Ende ein Speicherbereich für bis zu 8 Eingabezeilen. Falls das Bit 2 der Systemvariable **SFLAG** auf 0 steht, können in **EDITSTRING** mit der ESC-Taste die letzten Eingaben zurückgeholt und dann geändert werden. Natürlich ist dazu die Speicherung jeder Eingabe notwendig. Falls diese Option nicht genutzt werden soll, so muß nur das Bit 2 von **SFLAG** auf 1 gesetzt werden. Die Freigabe des nicht mehr benötigten Zeilenpuffers ist nur bei Neustart des geänderten KK-FORTH (oder eines Zusatzimage) möglich, wobei die im SYSVAR-Bereich gespeicherte Länge des SWORK-Bereiches auf 0 gesetzt sein muß. Da aber der Wert von **SFLAG** nicht in die Kopie der Systemvariable zurückgeschrieben und deshalb auch nicht gespeichert wird, muß er direkt verändert werden:

```
sflag  sysvar -  (sysvar +        ( Adresse ermitteln )
dup @  4 or  swap !               ( Bit 2 setzen )
```

Bei eigenständigen KKF-Systemen wird in Programmen statt der seriellen Schnittstelle oft eigene Hardware wie LCD-Anzeige oder Tastatur verwendet. Um trotzdem die ganzen Möglichkeiten des KK-FORTH zu nutzen, können alle Ein-/Ausgaben so umgeleitet werden, daß sie die neuen Schnittstellen verwenden. Die dazu nötigen Veränderungen der USER-Variablen INPUT und OUTPUT werden in Kapitel 3.5 beschreiben.

2.4.6 Das Fileinterface

Es gibt zwei Arten von Filebehandlungen im KK-FORTH:

Die erste Befehlsgruppe beruft sich auf den FORTH83-Standard. Von einem geöffneten File können sogenannte Screen's mit einer Länge von jeweils 1024 Bytes in den Speicher geladen, verändert und wieder gespeichert werden. Diese Screens können sowohl als reine Datenspeicher als auch für Programme verwendet werden.

Die zweite Befehlsgruppe ist eine direkte Anwendung der MSDOS-Betriebssystemaufrufe und erlaubt die Manipulation von Files über sogenannte Handlenummern. Dabei überläßt das KK-FORTH die Verwaltung der Files vollständig dem Anwender und dem Betriebssystem bzw. den eigenen Befehlen des Vektors **DISC** . Es werden deshalb keine Fehlerbehandlungen durchgeführt, sondern bei jedem Befehl die entsprechende Fehlernummer zurückgeliefert. Bei Terminal-Versionen des KK-FORTH wird das Fileinterface dabei transparent für den Anwender durch das Terminal durchgeführt.

Standardbefehle des Fileinterface

Für das Bearbeiten von Screenfiles kennt der FORTH83-Standard folgende Befehle:

BLOCK	(n -- addr)	Liefert Speicheradresse des Blockes
BUFFER	(n -- addr)	Liefert Speicheradresse, ladet aber nichts
UPDATE	(--)	Markiert aktuellen Block als geändert
FLUSH	(--)	Schreibt geänderte Blöcke zurück
EMPTY-BUFFERS	(--)	Löschen des Blockpuffers ohne Sicherung

Leider sind die Befehle zum Anlegen, Öffnen, Vergrößern, Schießen und Löschen von Files nicht im FORTH83-Standard definiert. In Anlehnung an das MSDOS und dem F83 wurden deshalb folgende Wörter verwendet.

MAKE <File>	(--)	File neu Anlegen und Öffnen
OPEN <File>	(--)	File öffnen (mit Dictionaryeintrag)
(OPEN	(fcb --)	File öffnen
MORE	(n --)	File um n Screens vergrößern
CLOSE	(--)	Aktuelles File schließen
DELETE <File>	(--)	File löschen
L	(n --)	Screen n editieren
V	(--)	Position des letzten Fehlers anzeigen

Dabei wird automatisch, bis auf **(OPEN** , beim Anlegen oder Öffnen eines Files gleichzeitig ein gleichlautender Befehlsname angelegt. Dieser dient dem KK-FORTH als Speicher des Filenamens und wird für das verkettete Laden von Files benötigt. Wird der neue Befehl dann aufgerufen, so öffnet er automatisch das zugehörige File.

Das KKF verfügt nur über einen Diskpuffer (Adresse FIRST+2) mit 1024 Zeichen. Falls einmal Screens von einem Block auf einen anderen Block übertragen werden, so muß ein externer Speicher (oder der FORTH-Speicher) verwendet werden. Die aktuelle Handlenummer des Files ist in der Variablen **FILE-ID** gespeichert und dient (bei Werten ungleich 0) gleichzeitig als Flag, daß ein File geöffnet ist. Damit im Normalfall nur ein File für das KKF geöffnet ist, wird auch beim verketteten Laden (z.B. beim INCLUDE-Befehl in einem Screen) vorher das alte File geschlossen. Um es dann wieder zu Öffnen, wird der in der Variable **FILE-FCB** gespeicherte Zeiger auf den Filenamen gemerkt.

Die Befehle **L** und **V** sind bei den Terminal-Versionen standardmäßig vorhanden und rufen den Editor des Terminal's auf. Ansonsten muß er erst durch das Laden des Screeneditors verfügbar gemacht werden. Auf der Diskette wird meist ein vollständiges FORTH-System mit Editor, Assembler und weiteren Tools mitgeliefert.

Zusätzliche Filebefehle

Für manche Programme ist das gleichzeitige Bearbeiten mehrerer Files notwendig. Deshalb stehen viele dazu benötigte Routinen dem Anwender schon mit dem KKF-Kern zur Verfügung. Die einzige Einschränkung ist dabei, daß normalerweise nur Files des aktuellen Verzeichnis behandelt werden.

Alle Befehle (siehe Kurzglossar unter Fileverwaltung und bei **DISC**) beginnen mit (FILE... und liefern neben den gewünschten Parameter auch ein Fehlerflag. Ist das Fehlerflag ungleich 0, so werden keine weiteren Daten zurückgeliefert. In eigenen Programmen ist darauf zu achten, daß nur eine limitierte Anzahl von Files gleichzeitig geöffnet sein dürfen und daß jedes geöffnete File vor dem Verlassen des Programmes wieder geschlossen wird.

Bei der Übertragung von Daten zwischen Speicher und dem File ist die Länge auf 65535 Bytes pro Befehl begrenzt. Da einige Prozessoren eine segmentierte Speicherverwaltung verwenden, ist auch hier evtl. eine Begrenzung der Datengröße zu beachten. Bei den 8086-Prozessoren führt z.B. das Laden von $1000 Bytes ab der Pointer-Adresse $1234:F800 zur Überschreibung des Segment-Anfangs.

Auch die Fileverwaltung wurde über eine USER-Variable vektorisiert. Dadurch können bei EMUF's mit eigener Hardware an Stelle der Terminal-Schnittstelle auch diese Diskettenlaufwerke und Harddisks angesteuert werden. Nach Änderung von DISC werden auch die FORTH-Screenfiles über diese neue Schnittstelle verwaltet. Kapitel 3.5 und das auf Diskette mitgelieferte I/O-File zeigt die aktuelle Implementation und wie eine eigene Definition durchzuführen ist.

2.4.7 Umleitung der Fehlerbehandlungsroutine

Normalerweise ist die Fehlerbehandlung in FORTH recht dürftig. Als Standardbefehle stehen nur **ABORT** und **ABORT"** zur Verfügung. Dabei der Name "Fehlerbehandlung" falsch gewählt, da die einzige Aktion der Rücksprung zum Interpreter **QUIT** nach Löschen des Datenstacks und evtl. noch vorher eine Ausgabe des zuletzt eingegebenen Befehls mit zugehörigem Kommentar ist.

Im KK-FORTH wurde dieser Punkt gründlich überarbeitet. Es gibt mehrere Befehle zum Testen auf Fehler. Auch die Fortsetzung eigener Programme nach dem Auftreten eines Fehlers ist jetzt möglich.

Die USER-Variable ERRORHANDLER

Als Information steht einer Fehlerbehandlungsroutine eine 16Bit-Fehlernummer zur Verfügung. Diese Nummer wird an den FORTH-Befehl weitergegeben, dessen Codefeldadresse in der USER-Variable **ERRORHANDLER** steht. Da weder der Daten- noch der Returnstack gelöscht wird, kann durch deren Untersuchung die Quelle des Fehlers ermittelt werden.

Die Standard-Fehlerbehandlung

Beim Start zeigt dieser Fehlervektor auf die Routine **(ERRORHANDLER** . Diese Routine verwerten die angegebene Fehlernummer. Tritt der Fehler während der Eingabe von Tastatur auf und ist Bit 15 der Fehlernummer gelöscht, so wird weder der Datenstack gelöscht noch der Systemstatus (kompilieren oder interpretieren) geändert. Dadurch kann ein Eingabefehler sehr einfach durch Zurückholen der letzen Eingabezeile, entfernen des Textes vor dem Fehler und Korrektur des Befehls erfolgen.

Da die Liste der Fehlernummern sehr umfangreich ist, wurde sie als Übersicht in den Anhang ausgelagert und wird in Kapitel 3.6 noch ausführlich beschreiben. Deshalb hier nur noch einige Anmerkungen zu Fehlernummern:

1. Bit 15 der Fehlernummer gibt der Routine **(ERRORHANDLER** an, daß in jedem Fall der Datenstack gelöscht werden soll und der Interpretermodus zu aktivieren ist.

2. Es gibt zwei Fehlernummern mit besonderer Bedeutung:
   $0000 Flag, daß kein Fehler aufgetreten ist
   $7FFF Der Fehlertext liegt als CFA auf dem Datenstack

3. Die Fehlernummern sind in Gruppen aufgeteilt:
   $0001 ... Betriebssystem-Fehlermeldungen
   $7E01 ... Wird von **ERRORTEXT@** für FORTH-Meldungen verwendet
   $7F01 ... FORTH-Fehlermeldungen

Einfache Fehlerbehandlung in Anwenderprogrammen

Die Umleitung der Fehlerbehandlung geschieht z.B. mit:

```
: usererror  standard-io ."  Error : " errortext@ type  quit ;
' errtrap  errorhandler !
```

Diese einfache Art der Fehlerbehandlung wird in etwas erweiterter Form im KKF-Kern verwendet und startet nach der Ausgabe der Fehlermeldung über **QUIT** wieder den FORTH-Interpreter. In eigenen Programmen wird natürlich statt **QUIT** der eigene Hauptbefehl eingefügt und dadurch das Programm neu gestartet oder fortgesetzt.

Errortrapping für Menüprogramme

In Kapitel 4.6.1 wird ein Beispielprogramm beschrieben, in dem die Fehlerbehandlung durch eine eigene Routine abgewickelt wird. Wenn man das DEMO-Programm wegläßt und nur die Grundbefehle in eigenen Menüprogrammen verwendet, so muß man nicht mehr bei einem Hauptbefehl anfangen. Statt dessen lassen sich in jedem Untermenü eigene Einsprünge definieren. Dabei funktioniert dieses Verfahren auch dann, wenn Programmteile in Module ausgelagert wurden und bei Fehler zuerst noch diese Module geladen werden müssen.

2.4.8 Nutzung der DEFER-Befehle des KK-FORTH

Eine weitere Möglichkeit zum Eingreifen in das KK-FORTH und zur Erstellung flexibler Programme bietet der Definitionsbefehl **DEFER** .

```
Defer msg   ( -- )
```

DEFER erwartet den Namen eines neuen Befehls. Dieser Name wird in das KK-FORTH aufgenommen, liefert aber bei Aufruf einen Fehler. Erst durch spätere Definition des entsprechenden Aktionswortes und der Aktivierung des DEFER-Wortes ist die Definition des Befehls abgeschlossen.

```
: (msg    ." Forwärts-Definition abgeschlossen ;
' (msg IS msg
```

Aktiviert werden die DEFER-Befehle mit IS. Die Codefeldadresse des auszuführenden Befehls (liefert ') wird in das DEFER-Wort (hier **msg**) eingesetzt. Dadurch können z.B. Vorwärtsreferenzen oder beliebige Umdefinition von bestimmten Befehlen realisiert werden.

Fehler entstehen dann, wenn ein nicht initialisiertes DEFER-Wort aufgerufen wird oder **IS** auf ein Befehl angewendet wird, der nicht mit **DEFER** angelegt wurde.

Im KK-FORTH gibt es eine Reihe von DEFER-Befehlen. Sie teilen sich in vier Gruppen auf:

- Initialisierungsroutinen:
  `'BOOT  'COLD  'ABORT  'ERROR  'BYE`
- Korrekturroutinen zum Schutz eigener Datenstrukturen:
  `'REMOVE  'DCLEAR  'RCLEAR`
- Umleitung von Standarddefinitionen:
  `NUMBER?  NUMBER  NUMBER,`
  `FIND  CREATE  ,A  ,C`

```
      ERR0RTEXT@   .STATE        .BLK
 - Vom System selbst veränderbare Definitionen
      PARSER       PAUSE
```

In der ersten Gruppe ist vor allem das DEFER-Wort **'ABORT** interessant. Durch Umsetzung auf eigene Routinen wird beim Start des KK-FORTH oder bei **COLD** zuerst das angegebene Befehlswort ausgeführt. Autostartprogramme lassen sich so einfach realisieren. Die Verwendung von **'COLD** ist unten noch beschrieben.

Bei Aufbau eigener Datenstrukturen wird häufig der Daten- und Returnstack oder Speicherbereiche im Dictionary verwendet. Die drei angegebenen DEFER-Befehle sind mit **NOOP** vorbesetzt und können zur Einbindung eigener Löschroutinen verwendet werden. Erst nach Aufruf dieser Befehle werden die Stacks (bzw. der Speicher) freigegeben.

Die dritte Gruppe von DEFER-Wörtern stehen beim Systemstart auf ihre Standardwörter (ohne Header unmittelbar hinter der DEFER-Definition). Sie können dann zur Erweiterung des Systems dauerhaft umgeleitet werden. Als Anwendung dazu ist eine eigene Interpreter-Routine oder die Realisierung der Headerverwaltung im externen Speicher. Häufig mißbraucht wird der Vektor .BLK, der beim Laden von Blöcken die aktuelle Blocknummer ausgibt und deshalb auch für Statuszeilen nützlich ist.

2.5 Literaturhinweise

Natürlich konnten in den letzten paar Seiten nicht alle Aspekte der FORTH-Programmierung berücksichtigt werden. Es sollte lediglich eine kleine Einführung sein und den Programmierer mit der Anwendung des KK-FORTH vertraut machen. Erst das systematische Studieren der einzelnen FORTH-Befehle und das Arbeiten damit führt zu einem Punkt, an dem man wirklich die Vorzüge des FORTH erkennt.

2.5.1 Einführungen

Parallel zur Einführung sind besonders die beiden Bücher "Programmieren in FORTH" und "In FORTH denken" von Leo Brodie zu empfehlen. Dabei ist "Programmieren in FORTH" ein Werk, daß jedem FORTH-Programmierer, ob Anfänger oder Profi, vorgeschlagen werden kann. Das zweite Buch dagegen behandelt die Philosophie, die die Grundlage der FORTH-Programmierung bildet. Dabei werden auch stilistische Probleme wie Namensgebung und Screenformatierung besprochen.

Für einen Kenner ist dagegen "FORTH-83" von Roland Zech zu empfehlen. Es behandelt schon relativ komplexe Probleme wie Einbindung von Strings in FORTH.

[E01] Leo Brodie:
Programmieren in FORTH
(Hanser Verlag; ISBN 3-446-14070-0)

[E02] Leo Brodie:
In FORTH denken
(Hanser Verlag; ISBN 3-446-14334-3)

[E03] Ronald Zech:
Die Programmiersprache FORTH
(Franzis-Verlag; ISBN 3-7723-7261-9)

[E04] Ronald Zech:
FORTH-83
(Franzis-Verlag; ISBN 3-7723-8621-0)

[E05] C. K. McCabe
Programmieren mit FORTH (auch für volksFORTH)
(Vieweg; ISBN 3-528-04346-6)

[E06] Hans-Walter Beilstein
Wie man in FORTH programmiert
(Vogel-Buchverlag Würzburg ISBN 3-8023-0165-X)

[E07] P. Kail
FORTH (Einführung u. vollst. Programmierkurs)
(Oldenbourg Verlag; ISBN 3-486-2027-6)

2.5.2 Standards und FORTH-Versionen

Natürlich gibt es auch für die diversen Standard's entsprechende Unterlagen. Die folgende Liste ist nach Gruppen unterteilt. Dabei bildet das F.I.G.-FORTH die älteste FORTH-Gruppe. Das am Ende erwähnte BASIS-Dokument stellt dagegen den aktuellen Stand der Verhandlungen zum ANSI-FORTH dar.

[F01] FORTH-INTEREST-GROUP:
a.) FIG-FORTH Installation manual
b.) fig-FORTH 6502 Release 1.1
c.) fig-FORTH Z80 Ver. 1.1
d.) fig-FORTH 6809 Rel. 1.1
... (weitere Prozessoren verfügbar)
(FORTH INTEREST GROUP
P.O. Box 1105
San Carlos, Ca. 94070)

[F02] MVP-FORTH Series
Volume 2: MVP-FORTH Source Listings
8080 (CP/M), IBM-PC, Apple II
(Mountain View Press, Inc.)

[F03] FORTH Standards Team:
FORTH-83 STANDARD
(Mountain View Press. Inc.)

[F04] G.U. Vack
Der Standard FORTH-83
(Kammer der Technik Suhl)

[F05] Rockwell International
RSC-FORTH User Manual
(für Prozessoren 65F11 und 65F12)

[F06] C. H. Ting
Inside F83
(Offete Enterprises, Inc.)

[F07] FORTH-Gesellschaft e.V.
volksFORTH (für PC, CP/M, Atari u. C64)
(FORTH-Gesellschaft e.V.; Postfach 1110; 8039 Unterschleißheim)

[F08] Aumiller/Luda
Atari ST 32FORTH-Compiler
Atari ST Programmieren mit FORTH
(Markt&Technik-Verlag)

[F09] ANS ASC X3/X3J14 Technical Committee
X3J14 BASIS Document Revision: BASIS 15 (evtl. schon 17)
(111 N. Sepulveda Blvd., Suite 300, Manhattan Beach, CA 90266)

2.5.3 Applikationen

In den Applikationen findet man Lösungen zu Problemen, die andere Programmierer schon zigmal gelöst haben. Darüber hinaus sind auch Proceedings von amerikanischen und europäischen Treffen erhältlich. Abgerundet werden diese Informationen durch umfangreiche Unterlagen zu bestimmten FORTH-Themengebieten ound Diplomarbeiten.

[A01] S. D. Roberts:
FORTH Applications
(Hofacker Gmbh; ISBN 3-88963-061-8)

[A02] Wilhelm-Pieck-Universität - Rostock
Applikation von FORTH

[A03] Dick Pountain
Object-oriented FORTH
(Implementation of Data Structures)
(Academic Press Inc. Ltd. - London)

[A04] Michael Stenzel
Entwicklung einer digitalen Oszilloskopkamera
(Diplomarbeit am Lehrstuhl Prof. S. J. Skorka)

[A05] Kammer der Technik - Suhl
Tagungsband 87 - FORTH 1988

[A06] Proceedings zu euroFORML
(Sammlung der Vorträge der euroFORML-Konferenz)
(Kopierservice der FORTH-Gesellschaft e.V.)

[A07] Institute For Applied FORTH Research
The Journal Of FORTH Application And Research
(70 Elmwood Avenue, Rochester, New York 14611 USA)

[A08] Jahrestagungen der FORTH-Gesellschaft e.V.
(FORTH-Gesellschaft e.V.)

[A09] MicroProcessor Engineering Ltd
Floating Point & String Listing
(21 Hanley Road Shirley, Southampton SO15AP)

2.5.4 Zeitschriften

Wie auch in anderen Programmiersprachen gibt es bestimmte Zeitschriften, die in unregelmäßiger Folge FORTH-Applikationen bringen. Speziell für FORTH gibt es in Deutschland die "Vierte Dimension", die Clubzeitschrift der FORTH-Gesellschaft e.V. . Aber schon vorher gab es die "FORTH DIMENSION", die Clubzeitschrift der F.I.G. . Auch andere Zeitschriften bringen in unregelmäßiger Folge FORTH-Artikel. Besonders hervorzuheben ist dabei Dr. Dobb's, da sie so viel Stoff zusammenbrachte um gleich zwei spezielle Bücher damit zu füllen.

[Z01] FORTH INTEREST GROUP
FORTH DIMENSION (Clubzeitschrift; 1/4-jährlich)

[Z02] FORTH-Gesellschaft e.V.
4. Dimension (Clubzeitschrift; 1/4-jährlich)

[Z03] Dr. Dobbs Journal
FORTH-Spezialausgabe früher einmal im Jahr

[Z04] Dr. Dobb's
Toolbook of FORTH
(M&T-Verlag)

[Z05] Dr. Dobb's
Toolbook of FORTH Volume II
(M&T-Verlag)

[Z06] BYTE
Vereinzelte FORTH-Artikel
(McGraw-Hill Publication)

Kapitel 3

Beschreibung

In diesem Kapitel findet man weitere Informationen zum KK-FORTH. Neben den wichtigen Tabellen zum SYSCON-, SYSVAR- und Task-Bereich sollten vor allem der Ablauf der Systeminitialisierung und die Informationen zu Fehlermeldungen beachtet werden.

3.1 Allgemeines zum KK-FORTH

Bei dem KK-FORTH handelt es sich um eine FORTH83-Implementation mit vielen Erweiterungen. Für Adressen werden 16 Bit verwendet und erlaubt damit einen direkten Zugriff auf 64KByte. Da das KKF ROM-fähig ist, sind einige Systemteile wie Variablen, Vokabulare, Konstanten, Task- und Diskpuffer aus dem Dictionary herausgenommen.

Bei den vollständig im RAM laufenden KKF-Versionen werden neue Befehle unmittelbar an das Dictionary angehängt. Bei SAVESYSTEM <Filename> wird sowohl der alte FORTH-Kern als auch die zusätzlichen Befehle wieder auf Diskette zurückgeschrieben und stehen dann beim Start dieses neuen Files sofort zur Verfügung.

Bei ROM-Versionen des KK-FORTH wird ein fester Kern in das EPROM gebrannt. Beim Start sucht dann das FORTH nach zusätzlichen Befehlen ab einer bestimmten Speicheradresse. Fehlen sie, so meldet sich der KKF-Kern über die angegebene (meist serielle) Schnittstelle.
Nachdem man ein Programm erstellt hat, kann mit SAVESYSTEM <Filename> der zusätzliche Dictionaryteil zusammen mit Variablen- und Taskbereich gespeichert und an das Kernimage des KKF angehängt werden.

Wird dieses Zusatzprogramm beim Start gefunden, so kopiert es das KKF zuerst an die gewünschte Speicheradresse und bindet es in das System ein. Durch Veränderung eines DEFER-Wortes kann auch sofort ein bestimmter Befehl gestartet werden.

3.2 Speicheraufteilung

In üblichen FORTH-Versionen besteht das System aus folgenden Teilen:

- Dictionary mit
  - + ORIGIN-Bereich mit Starteinstellungen
  - + Befehlswörtern
  - + Datenfelder und Variablen
- Arbeitsspeicher (meist ab Dictionary-Ende)
- Taskbereich mit
  - + Datenstack
  - + Umschaltroutine für Multitasker
  - + USER-Variablen
  - + Eingabebereich
  - + Returnstack
- Pufferbereich für Blockoperationen

Das KK-FORTH ist für den Einsatz in ROM-Systemen konzipiert worden. Deshalb gibt es einen etwas abweichenden Aufbau.

- Systemkonstanten
  - + Sprung auf Kaltstart-Routine
  - + Angaben zur Speicheraufteilung
- Systemvariablen oder Startwerte des SYSVAR-Bereiches
  - + Header
  - + Informationen zur aktuellen KKF-Version
  - + Systemvariablen
- Dictionary
- Freier Arbeitsspeicher
- Variablenbereich
- Heapbereich
- Taskbereich mit
  - + Arbeitsspeicher WDP@ und PAD
  - + Datenstack
  - + USER-Variablen
  - + Puffer für Eingabezeile
  - + Returnstack
- Diskpuffer (Screennummer + 1024 Zeichen)
- System-Arbeitsspeicher mit Zeilenpuffer
- (Kopie der) Systemvariablen

Im EPROM oder beim File auf Diskette werden die Variablen-, Heap- und Taskbereiche unmittelbar nach dem Dictionary abgelegt. Dabei werden aus dem Taskbereich nur jeweils die USER-Variablen der einzelnen Tasks gespeichert. Dadurch wird erreicht, daß das gesamte KKF-Image in 16KByte untergebracht werden kann.

Bei einigen KKF-Versionen wie z.B. beim 84C015-EMUF folgt am Speicherende hinter den Systemvariablen noch die Kernroutine zur Ausführung des nächsten FORTH-Befehls. Sie wurden nur deswegen in's RAM verlegt, um den Debugger die Bearbeitung eines Programmes im Einzelschrittmodus zu ermöglichen. Bei Z80-kompatiblen Prozessoren ist dies ohne Verlust an Geschwindigkeit möglich. Bei Prozessoren der 8086-Serie (V20, 8088...80486) ist diese Routine so kurz, daß sie in alle Assembler -Routinen direkt verwendet wurde. Um bei der Programmentwicklung trotzdem den Debugger nützen zu können, ist eine modifizierte KKF-Version beigefügt. Dieses etwas langsamere KK-FORTH legt die sogenannte NEXT-Routine (nicht zu verwechseln mit der **FOR** ... **NEXT** -Kontrollstruktur) in das RAM. Die Variable **NEXT-LINK** enthält die Anfangsadresse der Routine. Bei dem Wert 0 in **NEXT-LINK** ist der Debugger nicht verwendbar.

3.2.1 Systemkonstanten

Am Anfang des KK-FORTH sind nach dem Sprungvektor (belegt 2 Zellen) und 2 reservierten Zellen weitere 4 Zellen mit Informationen zur FORTH-Version angelegt. Die folgende Adreßangaben gehen davon aus, daß es sich um einen Byte-adressierten Speicher handelt. Jedoch sollte man statt SYSCON+8 im Programm die Formel `SYSCON 4 CELLS +` einsetzen.

SYSCON	**Sprung auf BOOT-Routine**
SYSCON + 4	**reserviert**
SYSCON + 8	**Speicheradresse des SYSVAR-Bereiches**
SYSCON + 10	**Anfangsadresse des Zusatzimage**
SYSCON + 12	**Anfangsadresse des RAM's**

SYSCON + 14 Endadresse + 1 des RAM's

In den RAM-Versionen des KK-FORTH werden die Angaben zur Imageadresse und zum RAM-Anfang nicht verwendet. Falls nicht schon durch das Betriebssystem (z.B. im CP/M) die Endadresse des RAM-Speichers vorgegeben wird, verwendet KK-FORTH die Angabe in SYSCON+14 zur Berechnung der Speicheradressen.

Bei den EPROM-Versionen ist wegen der Auslagerung des SYSVAR-Bereiches an das Speicherende keine Änderung von SYSCON+14 möglich. Es können aber abhängig von der Systemkonfiguration die Anfangsadresse des Zusatzimage und die Anfangsadresse des RAM's angegeben werden.

3.2.2 Systemvariablen

Zum Arbeiten und zur Speicherung des Programmes benötigt das KK-FORTH einen festen Speicherbereich mit den Systemvariablen. Dieser Speicherbereich befindet sich bei RAM-Versionen des KK-FORTH unmittelbar nach den Systemkonstanten im FORTH-Kern und bei ROM-Versionen am Ende des Speichers und hat folgenden Aufbau:

SYSVAR Kennung für Zusatzimage ($4b/$6f/$68/$6c)

SYSVAR + 4 Prozessorkennung

Bit 8-15: Hauptgruppe
Bit 0-7: Untergruppe

$00xx: Virtueller FORTH-Versionen und FORTH-Chips

$0001:	Virtueller FORTH-Prozessor	
$0011:	NC4000	(NC-Entwicklungsboard)
$0021:	RTX-2000	(FG-Board)
$0022:	RTX-2001	
$0023:	RTX-2001A	
$0031:	FRP-1600	
$0041:	SC-32	

$01xx: 80x88/86-Serie

$0111:	8088	(PC)
$0112:	80188	
$0113:	??? (es gibt kein 80286SX)	
$0114:	80386SX	
$0115	80486SX	
$0121:	8086	
$0122:	80186	
$0123:	80286	(AT)
$0124:	80386	
$0125:	80486	
$0131:	V20	
$0132:	V25	
$0133:	V30	
$0134:	V33	
$0135:	V35	
$0136:	V40	
$0137:	V50	
$0138:	V60	
$0139:	V70	

$02xx: 8080/Z80-Familie

$0211:	8080	
$0212:	8085	
$0221:	Z80	(CP/M-Systeme)
$0222:	84011 oder 84C11	
$0223:	84013 oder 84C13	
$0224:	84015 oder 84C15	

$0231:	64180	
$0232:	64181	
$0241:	Z280	

$03xx: 65xx-Familie

$0311:	6502	(6502-EMUF)
$0312:	6504	(6504-EMUF)
$0312:	6509	(C64)
$0313:	6510	(CBM600/CBM700)

$04xx: 68xxx-Familie

$0411	68008
$0421	68000
$0422	68010
$0423	68020
$0424	68030
$0425	68040
$0431	68070
$0441	68302
$0442	68331
$0443	68332
$0444	68340

$05xx: Transputer-Serie

$051x:	T200-Serie
$052x:	T400-Serie
$053x:	T800-Serie
$054x:	T9000-Serie

$06xx: 8031-Serie

$07xx Z8- und Super8-Serie

$071x	Z86C00-Serie (ROM-Entwicklungssystem)
$072x	Z86C11-Serie (UART und externes ROM/RAM)
$073x	Super8-Serie

$08xx: 68xx-Serie

$081x	6800/6801/6803-Serie
$082x	6802
$083x	6804/6805-Serie
$0841	6809
$085x	68HC11-Serie

$09xx: 80C166-Serie

$0911	80C166

$0Axx: 320xx-Serie

$0A01:	32010
$0A02:	32020
$0A03:	32030

SYSVAR + 6 Hardwarekennung/Betriebssystem

Bit 8-15: Hauptgruppe

Bit 0-7: Untergruppe

$00xx: Virtuelles System

$01xx: Terminal-IO über PC

$0111	KKF_8415	84C15-EMUF über SIO2 (Polling)
$0121	KKF_V20S	V20-EMUF über SIO1 (Polling)
$0122	KKF_V20D	Debugger-Version von KKF_V20S
$0123	KKF_V20M	V20-EMUF mit MSDOS-Terminal
$0131	KKF_FG	FG-Board über Port 7 (Bit 0/1)
$0132	KKF_A4	Analogkarte A4 über BOOT/EI1
$0141	KKF_65EM	6502-232 EMUF

$02xx: MSDOS/DRDOS

$0211	KKF_PC	MSDOS V2.11
$0212		MSDOS V3.21
$0213		MSDOS V3.3
$0214		MSDOS V4.01
$0215		MSDOS V5.0

$0221		DRDOS 3.40
$0222		DRDOS 3.41
$0223		DRDOS 5.0
$0231		WINDOWS 2.1
$0232		WINDOWS 3.0
$03xx: Commodore-Betriebssysteme		
$031x		Alte Versionen (PET ...)
$032x		CBM8000er-Serie
$0331	KKF_C700	RAM-Version für CBM700-Serie
$0332	KKF_C7E	EPROM-Version für CBM700-Serie
$0341	KKF_C64	Commodore C64
$0342	KKF_C128	Commodore C128
$035x		Amiga-Serie
$04xx: ATARI-Betriebssysteme		
$0411		ATARI 400
$0412		ATARI 800
$042x		KKF_ST ATARI ST
$043x		ATARI TT
$05xx: CP/M-Betriebssystem		
$0511	KKF_CPM	CP/M 2.2
$0512	KKF_CPM3	CP/M 3.0

SYSCON + 8 Versionsnummer (z.B. 1.2/0)

Bit 12-15: Hauptversion
- $01 16Bit-Version in 64KByte
- $02 Targetkompiler-Versionen des 16Bit-FORTH
- $03 32Bit-Versionen
- $04 Targetkompiler-Versionen des 32Bit-FORTH

Bit 8-11: Unterversion
- $01: Arbeitsversion ohne getrennten Variablenbereich
- $02: ROM-Fähige KKF-Version (entspricht dieser Beschreibung)

Bit 0-7: Revisionsnummer

SYSCON + 10 Attribut-Wort

Bit 14/15: Art
- %00 RAM-Version
- %01 ROM/RAM-Version

Bit 12/13: Speicheraufbau
- $00 nur 64KByte verfügbar
- $01 segmentierte Verwaltung (seg:addr)
- $02 lineare Verwaltung (32Bit-Adresse)

Bit 10/11: Floatingpoint-Unterstützung
- $00 fehlt
- $01: Software
- $02: Hardware
- $03: beides

Bit 9: 1=Fileinterface über Betriebssystem

Bit 8: 1=Terminal über Betriebssystem

Bit 6/7: Returnstack-Verwaltung
- $00: Stack im Programmspeicher
- $01: Stack im getrennten "Segment"
- $02: Hardware-Stack

Bit 4/5: Datenstack-Verwaltung

$00..$03 wie Bit 6/7

Bit 3: 1=Align der Speicherzugriffe bei 32Bit notwendig

Bit 2: 1=Align der Speicherzugriffe bei 16Bit notwendig

Bit 0/1: Art der Ablage von 32Bit-Werten
- %00: d0 d1 d2 d3
- %01: d1 d0 d3 d2
- %10: d2 d3 d0 d1 (wie 80x86/88 oder Z80)
- %11: d3 d2 d1 d0 (wie 68xxx oder RTX)

SYSVAR + 12	**Zeiger auf Kopie der Systemvariablen**
SYSVAR + 14	**VOC-LINK**
SYSVAR + 16	**DP**
SYSVAR + 18	**VDP**
SYSVAR + 20	**VLEN**
SYSVAR + 22	**HDP**
SYSVAR + 24	**HLEN**
SYSVAR + 26	**TDP**
SYSVAR + 28	**TASK-LINK**
SYSVAR + 30	**TASK**
SYSVAR + 32	**Enthält Adresse von TASK0**
SYSVAR + 34	**TASKS**
SYSVAR + 36	**TLEN**
SYSVAR + 38	**TMAXLEN**
SYSVAR + 40	**Enthält Adresse für FIRST**
SYSVAR + 42	**Enthält Größe des Diskpuffer**
SYSVAR + 44	**SWORK (Adresse des System-Arbeitsspeichers)**
SYSVAR + 46	**SLWEN (Länge des System-Arbeitsspeichers)**
SYSVAR + 48	**UWORK (Adresse des Anwender-Arbeitsspeichers)**
SYSVAR + 50	**UWLEN (Länge des Anwender-Arbeitsspeichers)**
SYSVAR + 52	**Enthält Länge einer gespeicherten Eingabezeile**
SYSVAR + 54	**Enthält die Anzahl der zu speichernden Zeilen**
SYSVAR + 56	**Enthält Backup der USER-Variable INPUT**
SYSVAR + 58	**Enthält Backup der USER-Variable OUTPUT**
SYSVAR + 60	**Enthält Backup der USER-Variable DISC**
SYSVAR + 62	**Reserviert für Baudraten-Angaben**
SYSVAR + 64	**Reserviert für versionsabhängige Daten**
SYSVAR + 66	**Reserviert für versionsabhängige Daten**
SYSVAR + 68	**frei**
SYSVAR + 70	**frei**

SYSVAR + 72 **frei**

SYSVAR + 74 **frei**

SYSVAR + 76 **SFLAG**

Bit 0=1:	Keine Start-Befehlszeile mehr verfügbar
Bit 1=1:	Keine Warnung durch **CREATE** ausgeben
Bit 2=1:	Keine Zeilenpuffer verwenden
Bit 3=1:	Schnittstelle wurde schon initialisiert

SYSVAR + 78 **UFLAG**

Frei für Anwender, wird dem Betriebssystem zurückgeliefert

3.2.3 Der Taskbereich

Jeder Task hat einen eigenen Speicher mit Ein-/Ausgabe-Bereich, beiden Stacks und den USER-Variablen. Da sowohl die Größe der einzelnen Bereiche als auch die Adressen veränderbar sind, werden mit den USER-Variablen auch Informationen über die Länge der einzelnen Bereiche abgelegt. Da diese Daten jedoch für das normale Arbeiten nicht benötigt werden, sind keine eigenen Namen dafür eingerichtet worden.

WDP@	Anfang des Arbeitsbereiches für diesen Task - Ablagebereich für WORD
…	
	- Arbeitsbereich für <# bis #>
PAD	Anfang des Stringbereich
…	
S0 @	Ende des Datenstacks
…	(danach einige Bytes frei)
TASKADDR@	Anfangsadresse des USER-Bereiches - Datenspeicher und Programme für Taskwechsel
TASKADDR@ + 16	WDP : Enthält Adresse des 1. Arbeitsbereiches
TASKADDR@ + 18	Enthält Länge des 1. Arbeitsbereiches (Adressen)
TASKADDR@ + 20	Enthält Anzahl der Datenstackwerte
TASKADDR@ + 22	**S0** : Enthält Startadresse des Datenstacks
TASKADDR@ + 24	Enthält Länge des 1. Sicherheitsbereiches
TASKADDR@ + 26	Enthält Länge des 2. Arbeitsbereiches (Adressen)
TASKADDR@ + 28	Enthält Anzahl der Returnstackwerte
TASKADDR@ + 30	**R0** : Enthält Anfangsadresse des Returnstacks
TASKADDR@ + 32	Enthält Länge des 2. Sicherheitsbereiches
TASKADDR@ + 34	**STATE**
TASKADDR@ + 36	**CURRENT**
TASKADDR@ + 42	CONTEXT-Arbeitsbereich (Offset, 6 Einträge)
TASKADDR@ + 56	**BLK**
TASKADDR@ + 58	**>IN**
TASKADDR@ + 60	**SCR**
TASKADDR@ + 62	**R#**
TASKADDR@ + 64	**SPAN**
TASKADDR@ + 66	**>TIB**
TASKADDR@ + 68	**#TIB**
TASKADDR@ + 70	**BASE**
TASKADDR@ + 72	**HLD**
TASKADDR@ + 74	**DPL**
TASKADDR@ + 76	**INPUT**
TASKADDR@ + 78	**OUTPUT**
TASKADDR@ + 80	**DISC**
TASKADDR@ + 82	**ERRORHANDLER**

TASKADDR@ + MAXTLEN@	Eingabepuffers für **QUERY** (= TIB - 2)
...	
R0 @	Ende des Returnstacks (danach einige Bytes frei)

Da man während der Entwicklung eines Programmes sehr häufig zu viele Werte vom Datenstack entfernt, wurde ein kleiner Sicherheitsbereich (Nummer 1) eingerichtet. Während der Befehlseingabe wird sowieso nach jedem Befehl der Stack getestet. In eigenen Programmen kann dann **?STACK** eingefügt werden.

Der zweite Sicherheitsbereich über dem Returnstack ist ein Schutz vor Programmierer, die einige **RDROP**'s zuviel im Programm haben. Es wird dann eine Routine angesprungen, die den Returnstack-Unterlauf bemängeln.

Beide Sicherheitsbereiche helfen aber nciht mehr, wenn mehr als drei Werte zuviel von den Stacks entfernt oder zu viele Daten auf den Stacks abgelegt wurden. Selbst wenn sich der Interpreter wieder meldet, kann keine Garantie auf die volle Funktionsfähigkeit des KK-FORTH mehr gegeben werden.

Der erste Arbeitsbereich enthält den WORD-Arbeitsspeicher **WDP@** , den Stringbereich um **PAD** und den Datenstack. Er sollte bei byteadressierten Prozessoren eine Größe von 84 (über **WDP@**) + 256 (über **PAD**) + 2*256 (Datenstack) Bytes besitzen.

Im zweiten Arbeitsbereich wird der USER-Bereich, der Termina-Eingabepuffer und der Returnstack untergebracht. Er wird im KKF-Kern meistens auf ein Größe von $100 (USER-Bereich) + 82 (Eingabepuffer) + 2*256 (Returnstack) Bytes eingerichtet.

3.2.4 Aufbau der Befehle

Obwohl der Aufbau des Befehls ab der Codefeldadresse sehr von der KKF-Version abhängt, kann der grobe Aufbau schon hier beschrieben werden. Für die Ermittlung der einzelnen Adressen gibt es die Befehle **L>NAME** , **N>LINK** , **>LINK** , **>NAME** , **>BODY** , **LINK>** , **NAME>** und **BODY>** .

Ein Befehl besteht aus:

Linkfeldadresse	Verkettung zur LFA des nächsten Befehlsnamen
Namensfeldadresse	Filename mit Längenangabe und 3 Flags
Codefeldadresse	Zeiger oder Adresse der Assemblerroutine
Parameterfeldadresse	Enthält Daten des Befehls

Die einzige Besonderheit ist die Verwendung der höchsten drei Bits der Namensfeldadresse. Hier werden die Flags für die Art des Befehls aufbewahrt. IMMEDIATE-Befehle werden auch in :-Definitionen ausgeführt und RESTICT-Befehle sind nur in :-Definitionen zulässig. Bei INDIRECT-Befehlen enthält die Codefeldadresse nur einen Zeiger auf die tatsächliche Codefeldadresse.

Bit 7=1	Befehl ist IMMEDIATE
Bit 6=1	Befehl ist RESTRICT
Bit 5=1	Befehl ist INDIRECT

3.3 Ablauf der Systeminitialisierung

Gestartet wird das KK-FORTH durch einen Sprung an den Anfang des Programmes oder des EPROM's. Dort steht dann ein Sprung zur eigentlichen Initialisierungsroutine, die folgende Handlungen durchführt:

BOOT

- Register initialisieren
  - 8086: CS --> DS, ES und SS; Stack vor (SYSVAR-Bereich
  - Z80: Stack vor SYSVAR-Bereich
- Endadresse des Kern-Image errechnen (DP+VLEN+HLEN+Tasks)
- Speicheraufteilung errechnen und Systemvariablen setzen
  - * (SYSVAR - UWLEN --> UWORK
  - * - SWLEN --> SWORK
  - * - BLEN --> FIRST
- Alle USER-Bereiche kopieren und initialisieren
- Heap-Bereich kopieren
- Variablenbereich kopieren
- SYSVAR zum (SYSVAR-Bereich kopieren
- Register für FORTH initialisieren
  - * TASK0 als aktueller Task
  - * Daten- und Returnstack leeren
- evtl. noch verwendete System-Flags und -Zeiger retten
- FORTH initialisieren
  - * Nur einen Task zulassen (**SINGLETASK**)
  - * Variablen des Fileinterface zurücksetzen
  - * Schnittstellen initialisieren
- '**BOOT** aufrufen (mit **NOOP** vorbelegt)
- '**COLD** aufrufen (mit **NOOP** vorbelegt)
- Mit Aufruf von ABORT zum FORTH-Kern springen
  - * Daten- und Returnstack löschen
  - * Flag für Header, **STATE** und **INDENT?** zurücksetzen
  - * '**ABORT** aufrufen (mit **IDENT** vorbelegt)
  - * Durch Aufruf von **QUIT** den Interpreter starten

Wie man im Ablauf sieht, werden beim Einschaltvorgang die Variablen-, Heap- und Taskbereiche vom Kern getrennt an das Speicherende gebracht. Da dies nur einmal geschehen darf, ist der Header des Befehls Befehl **BOOT** so verändert, daß er nur **COLD** ausführt. Diese "kleine" Initialisierung wirkt ähnlich wie **BOOT** , verwendet aber zum Löschen der Befehle **REMOVE** . Da aber der Variablenbereich nicht verändert wird, muß man darauf achten, daß alle DEFER-Wörter ihren Wert behalten und bei einem nicht mehr vorhandenen Befehl auch einen Absturz verursachen kann.

COLD

- Register initialisieren (wie bei **BOOT**)
- TASK0 ist aktueller Task
- Daten- und Returnstack löschen
- FORTH initialisieren (wie oben)
- Alle Befehle seit dem letzten SAVE entfernen
- '**COLD** aufrufen (mit **NOOP** vorbelegt)
- Mit Aufruf von **ABORT** zum FORTH-Kern springen

Für die Programmentwicklung ist es oft günstig, ab einer aktuellen Konfiguration nach **SAVE** weiterzuarbeiten. Nach Korrektur eines Fehlers genügt ein **EMPTY** zur Zurückstellung des Systems wie durch **COLD** . Es wird aber weder der Datenstack noch das aktuelle File verändert.

3.4 DEFER-Befehle des KK-FORTH

Obwohl das KKF auch in ein EPROM gebrennt wird und von dort aus läuft, können viele Systemeigenschaften verändert werden. Dabei unterscheidet man folgende Gruppen von DEFER-Befehlen:

Initialisierungsroutienen

Die hier beschriebenen Befehle dienen zur Einfügung eigener Initialisierungsroutienen oder zum Start eigener Programme an Stelle des FORTH-Interpreter/Kompilers.

'BOOT (--)

Der mit **NOOP** vorbelegte Vektor ist für den Einbau eigener Initialisierungsroutinen nach dem Start des FORTH vorgesehen. '**BOOT** wird nur einmal nach Initialisierung der Ein-/Ausgabe, des Fileinterfaces und der FORTH-Variablen vor der Ausführung von '**COLD** (siehe Kapitel 3.3) aufgerufen. Sinnvoll ist hier die Einbindung von Portinitialisierung oder Zurücksetzung von eigenen Variablen für Fileverwaltung (z.B. RAM-Floppy).

'COLD (--)

Das ebenfalls mit **NOOP** vorbelegte '**COLD** wird sowohl beim Systemstart als auch durch **COLD** aufgerufen. Die hier eingebundenen Routinen sollen wieder einen Zustand wie nach dem Einschalten erzeugen.

'ABORT (--)

Beim Systemstart, **COLD** oder nach **ABORT** wird vor dem Aufruf der Interpreter-/Kompilerschleife QUIT noch '**ABORT** ausgeführt. Im KKF-Kern zeigt dieser Vektor auf den Befehl **IDENT** , der dann die Einschaltmeldung ausgibt. Falls Autostart-Programme erstellt werden, so ist hier der günstigste Punkt zur Angabe des eigenen Befehls.

'ERROR (--)

Nach einem Fehler wird vor dem Aufruf des **QUIT** noch '**ERROR** ausgeführt. Falls die Standard-Fehlerbehandlung durch eigene Programme genutzt wird, kann hier trotzdem ein neuer Rücksprung ins Programm definiert werden. Man sollte aber dann den Returnstack löschen.

'BYE (--)

Bevor das FORTH-Programm entgültig verlassen wird, erfolgt über den DEFER-Vektor '**BYE** im KKF-Kern noch die Ausgabe der Abschiedsmeldung durch **-IDENT** . '**BYE** wird durch **BYE** nach Schließen des aktuellen Files, aber noch vor der Rückgabe der Systemresourcen aufgerufen.

Schutz eigener Datenstrukturen

Oft werden durch Programme im Speicher oder auf den beiden Stacks eigene Datenstrukturen erzeugt und verwaltet. Um auch beim Veränderung dieser Speicherbereiche diese Strukturen nicht zu Zerstören oder um zumindest die Flags zurückzusetzen, können in den wichtigen Kernroutinen **REMOVE** , **DCLEAR** und **RCLEAR** eigene Befehle eingebaut werden. Da der Speicher bzw. die beiden Stacks erst nach dem Aufruf gelöscht werden, sind noch alle Daten gültig.

'REMOVE (min max -- min max)

REMOVE ist die von allen FORGET-Befehlen verwendete Kernroutine, die nach der

Korrektur der Verkettungszeiger **DP** auf min und **HDP** auf max setzt. Es sind noch alle Verkettungszeiger beim Aufruf von **'REMOVE** gültig.

'DCLEAR (... -- ...)

Vor dem Löschen des Datenstacks durch **DCLEAR** wird noch **'DCLEAR** aufgerufen. **'DCLEAR** darf auch den Datenstack verändern, da er danach sowieso gelöscht wird.

'RCLEAR (-- ; R: ... -- ...)

Wie z.B. durch das Errortrapping werden häufig neben den Rücksprungadressen auch eigene Informationen auf dem Returnstack abgelegt. Es kann durch **'RCLEAR** z.B. die Variablen für den Rücksprung so verändert werden, daß wieder das Hauptprogramm angesprungen wird.

Umleitung von Standard-Definitionen

Einige im KKF-Kern verwendete Definitionen sind ebenfalls als DEFER-Befehle verfügbar. Damit aber kein zusätzlicher Platz verbraucht wird, haben die eigentlichen Definitionen keine eigenen Befehlsnamen, sondern folgen unmittelbar hinter der Definition. Die Adresse ermittelt man z.B. durch folgende Formel:

```
' Create >body cell+ aligned
```

NUMBER? (csa -- d n>0 | n -1 | addr 0)

Zur Umwandlung eines Zahlenstrings wird **NUMBER?** verwendet. Da hier auch ein günstiger Punkt zur Erkennung von Floatingpointroutinen ist, wurde dieser Befehl als DEFER-Wort angelegt. Es sollte dann neben dem entsprechenden Wert auch ein Flag mit Werten zwischen -32768 und -2 auf dem Stack abgelegt werden. Falls dies geschieht, müssen auch die beiden nachfolgenden Vektoren verändert werden.

NUMBER (d n>0 | n -1 -- d | n)

Dieser Befehl dient zur Interpretation des durch NUMBER? erkannten Zahlenwertes. Da im Kern dazu keine weiteren Veränderungen notwendig sind, zeigt dieser DEFER-Vektor auf DROP .

NUMBER, (d n>0 | n -1 --)

Bei der Kompilierung muß zwischen 16Bit- und 32Bit-Wert unterschieden werden. Anhand des Flags wird dann entweder **LITERAL** oder **2LITERAL** ausgeführt.

,A (cfa --)

Beim Kompilieren werden meistens die Codefeldadressen im Dictionary abgelegt. Da es (wie beim RTX-2000) auch Ausnahmen gibt, wurde ein eigener Befehl dafür definiert. Zusammen mit dem in einigen KKF-Versionen verfügbaren **,C** und der Variablen **CODE?** kann ein optimierender Kompiler eingebunden werden.

,C (code --) (nicht in alle Versionen verfügbar)

Dieser Befehl legt nach der Speicherung der Dictionaryadresse in **CODE?** den Wert im Dictionary ab. Zusammen mit **,A** dient **,C** zur Einbindung eines optimierenden Kompilers.

FIND (csa -- cfa f | csa 0)

Falls einmal die Befehlsheader für Befehle aus dem FORTH-System ausgelagert werden, kann durch Umdefinition des DEFER-Befehls **FIND** die Suche im externen Speicher

erfolgen. Als Ergebnis muß die Codefeldadresse des Befehls oder die Stringadresse und ein Flag zurückgeliefert werden.

CREATE (--)

Falls Aufbau des Befehlsheaders einmal verändert werden soll oder der Header extern verwaltet wird, so kann dies durch Veränderung von **CREATE** erreicht werden. Im FORTH83-Standard ist definiert, daß bei Ausführung des mit **CREATE** angelegten Befehls die Parameterfeld-Adresse der Definition im Dictionary zurückgeliefert wird.

ERRORTEXT@ (error -- addr len)

Neben der Umsetzung der Fehlermeldungstabelle auf eine andere Speicheradresse gibt es auch die Möglichkeit der Auslagerung auf ein externes File. Durch Veränderung von **ERRORTEXT@** kann dann die Fehlermeldung auch aus dem File gelesen werden, ohne daß dazu eine neue Fehlerbehandlung notwendig währe.

.STATE (--)

Vom Interpreter/Kompiler wird immer am Zeilenende "ok" oder am Zeilenanfang "]" ausgegeben. Falls es beliebt, kann diese Meldung auch verändert werden. Die Standardroutine zu **.STATE** ermittelt den richtigen Text aus dem Zustand der USER-Variable **STATE** (0=interpretieren).

.BLK (--)

Beim Laden eines Blockes wird durch den Vektor **.BLK** die Meldung ">" ausgegeben. Falls ein direkter Zugriff auf den Bildschirm möglich ist, kann diesr DEFER-Befehl auch zur Erzeugung einer Statuszeile mißbraucht werden. Es sollte aber nur dann etwas ausgegeben werden, wenn die Blocknummer in **BLK** einen Wert ungleich 0 hat.

Vom System genutzte DEFER-Wörter

Die letzten beiden Definitionen werden vom System selbst ständig verändert. Man sollte sie deshalb nur dann in eigenen Programmen manipulieren, wenn auch die entsprechenden Flags geändert werden.

PARSER (csa -- ...)

Je nach Zeiger in **PARSER** wird ein Befehlswort interpretiert oder kompiliert. Im Kern wird bei Veränderung dieses Zeigers durch **]** und **[** auch die USER-Variable **STATE** auf 0 oder -1 geändert.

PAUSE (--)

Der Multitasker ist im KKF-Kern nicht notwendig, da nur ein Task verwendet wird. Deshalb wird **PAUSE** auch beim Systemstart durch **SINGLETASK** auf **NOOP** gesetzt und dadurch deaktiviert. Nach Aktivierung durch **MULTITASK** zeigt dieses DEFER-Wort dann auf eine Routine, die den aktuellen Task beendet und den nächsten Task aktiviert.

3.5 -/- und Diskumleitung

Eines der wichtigsten Eigenschaften des KK-FORTH ist die Möglichkeit, alle Ein-/Ausgaben und das Fileinterface umzuleiten. Dazu werden neue Ausgabebefehle geschrieben und in entsprechenden Tabellen zusammengefaßt. Durch Umsetzen der USER-Variablen **INPUT** , **OUTPUT** oder **DISC** werden dann statt der Standard-Befehle die neuen Wörter verwendet.

Darüber hinaus ist auch eine Erweiterung der Ein-/Ausgabebefehle und die Verwendung der Grundroutinen für eigene (vektorisierte) Befehle möglich.

Verwaltung der Schnittstellen

Derartige Eigenschaften lassen sich nur erreichen, wenn die Schnittstellenbefehle nicht direkt ausgeführt werden, sondern die Adresse ihres Programmes ermitteln. In den drei USER-Variablen **INPUT** (für Eingabebefehle), **OUTPUT** (Ausgabe-Vektor) und **DISC** (Fileverwaltung) steht die Adresse eines Feldes, in dem die Codefeldadressen der entsprechenden Routinen abgelegt sind. Die Standardbefehle wie **KEY?** , **KEY** ... errechnen dann nur ihren Offset und springen zu der entsprechenden Routine. Nach Start des KK-FORTH zeigen die Vektoren auf **(INPUT** , **(OUTPUT** und **(DISC** .

Da die USER-Variablen gespeichert werden, bleiben einmal getroffene Veränderungen an der Schnittstelle bestehen. Bei Fehler aber werden die Schnittstellen neu initialisiert und dabei nicht auf die Vektoren im USER-Bereich, sondern auf die im SYSVAR-Bereich gespeicherten Werte zurückgegriffen. Dadurch kann während der Entwicklung gefahrlos der Ein-/Ausgabevektor verändert werden. Erst bei **SAVE** oder **SAVESYSTEM** werden dann die Umleitungen dauerhaft gespeichert und dann auch bei Fehler beibehalten.

Befehle UVECTOR und

Zur Definition der vektorisierten Befehle dienen im KK-FORTH zwei Wörter:

UVECTOR	(name ; offset user --) Definition eines Vektors
UTABLE:	(name ; Anzahl user --) Definition einer Tabelle

Mit dem ersten Befehl wird der Name des Ein-/Ausgabebefehls festgelegt. Als Parameter erwartet er nur die Nummer des Befehls in der Tabelle und die Speicheradresse der USER-Variable, über die der Befehl vektorisiert wird.

Mit **UTABLE:** wird die Tabelle erzeugt. Der Befehl verhält sich ähnlich wie **:** , erwartet aber auf dem Datenstack noch die Anzahl der Tabelleneinträge und wiederum die Speicheradresse der USER-Variable. Danach werden dann die Initialisierungsroutine und die auszuführenden Befehle für jeden Vektor angegeben. Beendet wird wie bei **:** mit dem **;** . Dabei muß man beachten, daß in diese Tabelle keine Kontrollstrukturen oder Literals kompiliert werden dürfen.

Bei der Anzahl der Einträge in der UTABLE:-Befehlsliste wird die Initialisierungsroutine nicht gezählt. Falls aber auch diese verwendet werden soll, so bekommt sie bei **UVECTOR** die Nummer 0.

Ausgabevektor OUTPUT

Für die Ausgabe wird der Zeiger **OUTPUT** verwendet. Die folgende Definition ist unvollständig, da die Grundroutinen **O-INIT** bis **((AT?** versionsabhängig sind. Wenn man später **(OUTPUT** aufruft, so wird zuerst die USER-Variable **OUTPUT** auf die Tabelle bei **(OUTPUT** gesetzt und dann die Initialisierungsroutine **O-INIT** aufgerufen. Der Aufruf von **EMIT?** ... **AT?** bewirkt die Ausführung von **((EMIT?** ... **((AT?** . Die senkrechten Striche vor **O-INIT** bis **((AT?** zeigen an, daß nach **SAVE** nur die Vektoren und die Tabelle als aufrufbare Befehle zur Verfügung stehen.

```
User output

  1 output UVector emit?
  2 output UVector emit
  3 output UVector type
  4 output UVector cr
  5 output UVector del
  6 output UVector bell
  7 output UVector cls
```

```
  8 output UVector maxat
  9 output UVector at
&10 output UVector at?

| : o-init ... ;                ( hier erfolgt die Definition )
...                             ( der Befehle o-init bis ((at? )
| : ((at? ... ;

&10 output UTable: (output
   o-init  ((emit?   ((emit    ((type    ((cr      ((del
           ((bell    ((cls     ((maxat   ((at      ((at?  ;
```

Eingabevektor INPUT

Für die Eingabe ist eine kürzere Tabelle vorgesehen. Man muß bei Änderung des INPUT-Vektors aber beachten, daß für das Editieren einer Eingabe mit **EDITSTRING** oder **QUERY** der Ausgabebefehl **EMIT** verwendet wird.

```
User input
  1 input UVector emit?
  2 input UVector emit
  3 input UVector type
  4 input UVector cr
  5 input UVector del

| : i-init ... ;                ( hier erfolgt die Definition )
...                             ( der Befehle i-init bis ((query )
| : ((query ... ;

&05 input UTable: (input
   i-init  ((key?    ((key     ((string  ((editstring  ((query ;
```

Fileinterface-Vektor DISC

Über den Vektor **DISC** werden alle Befehle für die Verwaltung des FORTH-Screenfiles abgewickelt. Da aber das verwendete Filesystem (auch über Terminal) meistens mehr als ein File verwalten kann, können diese Befehle auch zur direkten Filemanipulation herangezogen werden. Alle Befehle haben hier eine Klammer, da die Fehlerbehandlung nicht von diesen Routinen durchgeführt wird.

```
User disc
  1 disc UVector (file?
  2 disc UVector (file-create
  3 disc UVector (file-delete
  4 disc UVector (file-open
  5 disc UVector (file-close
  6 disc UVector (file-size
  7 disc UVector (file-pos!
  8 disc UVector (file-pos@
  9 disc UVector (file-read
&10 disc UVector (file-write
&11 disc UVector (file-free
&12 disc UVector (file-first
&13 disc UVector (file-next

| : d-init ... ;                ( hier erfolgt die Definition )
...                             ( der Befehle d-init bis ((file-next )
| : ((file-next ... ;

&13 disc UTable: (disc
```

```
d-init  ((file?
        ((file-create   ((file-delete
        ((file-open     ((file-close
        ((file-size     ((file-pos!      ((file-pos@
        ((file-read     ((file-write     ((file-free
        ((file-first    ((file-next  ;
```

3.6 Fehlerbehandlung

Die Kernroutine aller Fehlerbehandlungsroutinen ist **ERROR** . Sobald der Wert auf dem Datenstack ungleich 0 ist, wird die Routine aufgerufen, auf die USER-Variable **ERRORHANDLER** zeigt. Falls das KKF nicht verändert wurde, folgt nun durch die Routine **(ERRORHANDLER** die Ausgabe der zugehörigen Fehlermeldung, Test des Datenstacks und dann durch Aufruf von **QUIT** ein Rücksprung zum FORTH-Interpreter.

3.6.1 Fehlernummern

Da man auch wissen will, durch was der Fehler ausgelöst wurde und welcher Art der Fehler ist, wird ein System von Fehlernummern verwendet. Zusätzlich kann durch das höchste Bit (Bit 15) in der Fehlernummer **(ERRORHANDLER** mitgeteilt werden, daß vor Aufruf von **QUIT** der Kompilermodus verlassen und der Datenstack gelöscht werden soll.

$0000 **Kein Fehler (ERROR entfernt nur das Flag)**

Bit15=1 **Datenstack löschen und Interpretermodus aktivieren**

$0001 ... **Fehlermeldungen des Betriebssystems**

$0001 "unknown funktioncode"
Unbekannte Funktionsnummer
Wird vom Betriebssystem zurückgeliefert, wenn bei Aufruf eine falsche Befehlsnummer übergeben wird.

$0002 " file not found"
File nicht gefunden
Bei **OPEN** oder **MAKE** wurde das File mit angegebenen Namen nicht gefunden.

$0003 " path not found"
Pfad nicht gefunden
Der Pfad mit dem angegebenen Name wurde nicht gefunden.

$0004 " too many open files"
Zu viele Files offen
Dieser Fehler tritt auf, falls bei direktem Zugriff zu viele Files geöffnet oder beim wiederholten Öffnen eines Files die vorher verwendeten Handlenummern nicht mehr freigegeben wurden.

$0005 " access denied"
Zugriff verweigert
Beim Öffnen eines Files wird auch die Art des Zugriffes angegeben. Der Fehler $0005 tritt dann auf, wenn bei #RO-Attribut oder bei einer ROM-Floppy geschrieben wird.

$0006 " unknown handle"
Unbekannte Handlenummer
Falls man bei direktem Filezugriff eine falsche Handlenummer verwendet wird, wird Error $0006 zurückgeliefert. Dieser Fehler kann auch dann auftreten, wenn das Terminalprogramm bei einem geöffneten File im EMUF verlassen und dann neu gestartet wird.

$0007 " MCB destroid"
MCB zerstört
Falls nach Reservierung mehrerer externer Speicherblöcke über die angegebenene Grenzen hinaus geschrieben wird, können die Informationen der Speicherverwaltung zerstört werden. Wenn diese Fehlermeldung geliefert wird, ist oft das System nur noch durch einen Reset zurückzusetzen.

$0008 " no memory"
Kein Speicher verfügbar
Falls ein zu großer Speicherbereich angefordert wird oder ein schon reservierter Speicher nicht vergrößert werden kann, wird Error $0008 zurückgeliefert.

$0009 " unknown MCB"
Unbekannte MCB
Die bei **(MFREE** angegebene Speicheradresse ist nicht identisch mit einer angeforderten Adresse. Der dadurch referenzierte Speicher kann deshalb nicht freigegeben werden.

$7E01 ... Warnungen und Meldungen des KK-FORTH

$7e01 " and Datastack underflow "
Nach einem Fehler wird in **(ERRORHANDLER** noch überprüft, ob ein Datenstack-Unterlauf aufgetreten ist. Es wird dann diese Meldung ausgegeben und der Datenstack zurückgesetzt.

$7e02 " exist "
Falls bei **CREATE** der angegebene Befehl schon existiert, so wird diese Warnung ausgegeben. Sie hat aber keinen Einfluß auf die Ausführung von **CREATE** .

$7e03 " Include : "
Beim Laden eines Files mit **INCLUDE** oder **LOADFROM** wird am Anfang diese Meldung zusammen mit dem Filename ausgegeben.

$7e04 " End-Include : "
Nach dem Laden eines Files folgt noch die Ausgabe dieser Meldung.

$7F01 ... KKF-Fehlermeldungen

$7f01 " Stack underflow"
Datenstack-Unterlauf
Diese Fehlermeldung wird von **?STACK** erzeugt, wenn zu viele Werte vom Datenstack entfernt wurden.

$7f02 " Stack overflow"
Datenstack-Überlauf
Error $7f02 wird von **?STACK** erzeugt, wenn zu viele Werte auf dem Datenstack liegen.

$7f03 " R-Stack underflow"
Returnstack-Unterlauf
Von **QUIT** werden zusätzliche Rücksprungadressen auf den Returnstack gelegt. Falls in einem Befehl zu viele Returnstackwerte entfernt werden, kann sich das KKF-System dadurch in einigen Fällen mit dieser Fehlermeldung zurückmelden.

$7f04 " R-Stack overflow"
Returnstack-Überlauf
Dieser in den momentanen KKF-Versionen nicht verwendete Fehler zeigt an, daß zu viele Werte auf dem Returnstack liegen.

$7f05 " Too less parameters"
Zu wenige Parameter
Mit **?DEPTH** kann im KK-FORTH die Anzahl der Datenstackwerte getestet werden. Wird dabei die geforderte Anzahl unterschritten, so erfolgt die Ausgabe dieser Fehlermeldung.

$7f06 " Illegal value"
Unerlaubter Wert
Bei einigen Befehlen (wie z.B. **P@** beim RTX-2000) sind nur bestimmte Parameter erlaubt. Error $7f06 wird dann ausgegeben, wenn der Parameter fehlt oder einen falschen Wert hat.

$7f07 " Arithmetic overflow"
Überlauf bei Arithmetik
Einige Prozessoren lösen einen Interrupt aus, fall bei einer Division eine Überschreitung des gültigen Bereiches erkannt wird. Der Befehl wird dann abgebrochen und die Fehlernummer $7f07 ausgegeben.

$7f08 " Unexpected interrupt"
Nicht initialisierter Interrupt
In den EMUF-Versionen des KK-FORTH werden alle nicht vom FORTH verwendeten Interrupts auf eine Routine umgeleitet, die alle Interrupts abschaltet und diese Fehlermeldung ausgibt. Der Fehler tritt meistens dann auf, wenn der Interrupt ohne Initialisierung des Bausteins und Umleitung des Interruptvektors erlaubt wird.

$7f09 " Adress error"
Fehlerhafte Adresse
Einige Prozessoren erzeugen einen Interrupt, falls ein Wortzugriff auf eine ungerade Adresse erfolgt. Wenn es möglich ist, wird dieser Interrupt auf eine Routine für Error $7f09 umgeleitet.

$7f0a " DEFER not defined"
DEFER wurde nicht definiert
Beim Anlegen eines DEFER-Befehls wird **NODEFER** als auszuführende Routine angegeben. Falls der DEFER-Befehl vor seiner Verwendung nicht auf ein anderes Wort umgeleitet wird, erfolgt diese Fehlerausgabe.

$7f0b " Dictionary full"
Dictionary ist voll
Bei jedem Reservieren des Dictionary-, Variablen- und Heapspeichers wird durch **?ALLOT** getestet, ob noch genügend Speicher zwischen Dictionaryende in **DP** und Variablenanfang in **VDP** bleibt. Ist dies nicht mehr der Fall, so wird Error $7f0b ausgegeben.

$7f0c " USER-Area full"
USER-Bereich ist voll
Für den USER-Bereich sind nur **TMAXLEN@** Adressen vorgesehen. Sind alle Adressen belegt, so wird Error $7f0c ausgegeben.

$7f0d " CONTEXT full"
CONTEXT-Bereich ist voll
Im CONTEXT-Bereich sind nur insgesamt 6 Einträge vorgesehen. Error $7f0d wird dann ausgegeben, wenn alle Plätze besetzt sind und trotzdem noch **ALSO** aufgerufen wird.

$7f0e " DP in HEAP"
Dictionarypointer im Heap
REMOVE geht davon aus, daß alle Befehle zwischen erster und zweiter Speicheradresse gelöscht werden können. Falls aber der erste Werte im Heap und damit über dem zweiten Wert liegt ist das Löschen nicht mehr möglich und **REMOVE** bricht mit dieser Fehlermeldung ab.

$7f0f " is protected"
Befehl ist geschützt
Falls bei **FORGET** der Befehlsname unter der im SYSVAR-Bereich gespeichert Dictionaryadresse liegt, wird Error $7f0f ausgegeben.

$7f10 " name expected"
Name erwartet
Wird durch ein Definitionswort ein neuer Befehl angelegt, so muß nach dem Befehl noch der neue Name eingegeben werden. Ist dies nicht der Fall, so wird durch **?NAME** der Error $7f10 ausgegeben.

$7f11 " not found"
Name nicht gefunden
Falls durch den Befehl ' ein angegebener Befehlsname nicht gefunden wurde oder eine Eingabe nicht interpretiert oder kompiliert werden kann, so wird Error $7f11 ausgegeben.

$7f12 " no definition"
Kein Befehl definiert
Bei Verwendung von **IMMEDIATE** , **RESTRICT** und **INDIRECT** wird die Namensfeldadresse in **LAST** benötigt. Ist dieser Wert auf 0, so wurde kein Befehl definiert und es erfolgt diese Fehlerausgabe.

$7f13 " Is restrict"
Befehl nur in :-Definitionen zulässig
Dieser Fehler wird durch den Interpreter ausgelöst, wenn ein mit **RESTRICT** markierter Befehl eingegeben wird.

$7f14 " unstructured"
Fehlerhafte Kontrollstruktur
Wurden Kontrollstrukturen falsch verschachtelt oder bis zum Ende einer :-Definition nicht geschlossen, so erfolgt durch **?PAIRS** die Ausgabe des Error $7f14.

$7f15 " isn't DEFER"
Keine DEFER-Definition
Falls nach dem Befehl **IS** keine DEFER-Definition angegeben wird, so erfolgt diese Fehlerausgabe.

$7f16 " File not open"
Kein File geöffnet
Bei jedem Zugriff auf Screens erfolgt mit **?OPEN** ein Test der Variable **FILE-ID** . Hat sie den Wert 0, so ist kein File verfügbar und eine Fehlermeldung wird ausgegeben.

$7f17 " Number exceed Filesize"
Blocknummer zu groß
Überschreitet die beim Filezugriff angegebene Blocknummer die verfügbare Screenanzahl, so wird durch **?BLK** Error $7f17 ausgegeben.

$7f18 " Blocknumber not allowed"
Blocknummer nicht erlaubt
Der Wert 0 in **BLK** dient als Flag, daß der TIB interpretiert werden soll. Deshalb ist die Blocknummer 0 bei **LOAD** nicht erlaubt und verursacht Error $7f18.

$7f19 " COMMAND!-Error"
Terminal-Übertragungsfehler
Bei der Befehlsübertragung zwischen KK-FORTH und Terminalprogramm verursachen eingegebene Zeichen von Tastatur einen fehlerhaften Datenaustausch. Deshalb wird vor und während der Befehlsnummer-Übergabe (mit **COMMAND!**) getestet, ob Zeichen ankommen. Ist dies der Fall, so wird Error $7f19 zurückgeliefert.

$7f1a " UVECTOR-Error"
Fehler bei UVECTOR-Befehl
Falls die aktuell verwendete **UTABLE:**-Befehlsliste kleiner als die verwendete Offsetnummer des **UVECTOR**-Befehls ist, wird Error $7f1a ausgegeben.

$7f1b " not supported"
Befehl wird nicht unterstützt
In manchen KKF-Versionen fehlen einige Befehle oder Optionen. Falls dies nicht schon durch den fehlenden Befehlsnamen erkannt wird, so erfolgt die Ausgabe des Error $7f1b.

$7FFF Fehlermeldung liegt als CSA auf dem Datenstack

3.6.2 Anlegen eigener Texte zu Fehlermeldungen

Menüprogrammen können der Vektor **ERRORHANDLER** auf eigene Befehle umgeleitet und dadurch nach Ausgabe der Fehlermeldung wieder in den Hauptbefehl zurückspringen. Die Liste der englischen Fehlermeldung ist am Ende des KK-FORTH abgelegt. Die Routine **ERRORTEXT@** durchsucht zuerst die Fehlerliste (Zeiger in der Variable **>ERRORTEXT**) nach der entsprechenden Meldung und liefert dann Speicheradresse und Länge des Strings. Wurde kein Text gefunden, so wird ein eigener String mit Format "ERROR $xxxx" erzeugt. Durch Veränderung von **>ERRORTEXT** , oder **ERRORTEXT@** kann man auch eine deutschsprachige Fehlermeldung erzeugen.

Falls nur **>ERRORTEXT** verändert wird, so muß der Aufbau der neuen Tabelle wie im nachfolgenden Beispiel sein. Das Ende der Liste wird durch die Fehlernummer 0 markiert.

```
Create meinefehler
   $1001 ,  Ascii "  $, Port nicht aktiviert"  align
   $1002 ,  Ascii "  $, Eingabe falsch"        align
   $1003 ,  ...
   $0000 ,
meinefehler  >errortext  !
```

3.6.3 Austausch der Fehlermeldungen im KKF-Kern

Der Austausch der Fehlermeldungen ist auch ohne Metakompiler in dem ausgelieferten System möglich, da die Tabelle immer am Ende des FORTH steht.

In der RAM-Version kann nach Freigabe des Speichers mit

```
>errortext @  >errortext off  heap remove  save
```

die neue Tabelle gemäß dem obigen Beispiel kompiliert werden.

In den EPROM-Versionen muß eine neue Tabelle an Stelle der alten Fehlerliste eingefügt werden. Die Anfangsadresse kann man nach dem Start des KKF aus **>ERRORTEXT** entnehmen. Auf das Ende der Liste zeigt der in dem SYSVAR-Bereich des EPROM's gespeicherte Dictionarypointer. Soll die Größe des Bereiches verändert werden, so muß diese Adresse geändert und der nach der Fehlerliste folgende Speicherbereich mit Variablen und USER-Werte zur neuen Adresse verschoben werden. Diese Tätigkeit ist aber mit jedem guten EPROM-Programmierer durchzuführen.

Kapitel 4

Zusätze

In diesem Kapitel werden Tools und Beispielprogramme beschrieben, die bei den meisten KKF-Versionen mitgeliefert werden. Jedoch dienen die hier angegebenen Informationen nur zur Übersicht. Für die genaue Bedienung der entsprechenden Programme ist immer die Beschreibung des Zusatzhandbuches heranzuziehen.

4.1 Das KKF-Terminalprogramm

Bei einem EMUF übernimmt ein Termialprogramm für den PC/AT neben der Übermittlung von Eingaben und der Anzeige der Zeichen noch die Aufgabe des Fileservers. Über ESC-Sequenzen kann das KK-FORTH auf beliebige Files des aktuellen Verzeichnis zugreifen. Darüber hinaus können noch einige zusätzliche Befehle wie Setzen und Abfragen der Cursorposition oder Löschen des Bildschirms aufgerufen werden.

4.1.1 Bedienung des Terminalprogrammes

Neben dem vom EMUF-FORTH ausgelösten Aktionen hat aber auch der Bediener noch zusätzliche Möglichkeiten:

F1
ALT+H — Anzeige der Hilfsinformation zur Tastenbelegung

ALT+P — Ein-/Ausschalten des Druckers. In der unteren Statuszeile wird bei aktivem Drucker "P" ausgegeben. Da die Ausgabe parallel zum Bildschirm erfolgt, bleibt das Terminalprogramm stehen, bis der Drucker bereit ist. Es können aber trotzdem noch Zeichen empfangen werden.

ALT+L — Beim Arbeiten mit dem Terminal können alle empfangenen Zeichen auch in ein Logfile geschrieben werden. Dadurch kann, wie beim Erstellen dieser Beschreibungen, das Arbeiten mitprotokolliert werden. Nach Drücken von ALT+L muß der Name des Files (Vorgabe: KKF.LOG) eingegeben werden. Dieses File wird dann mit Länge 0 geöffnet und alle danach empfangenen Zeichen darin gespeichert. Falls beim Speichern ein Fehler auftritt (Diskette voll oder nicht beschreibbar), so wird das Logfile geschlossen. Es kann aber auch durch erneute Betätigung von ALT+L geschlossen werden. In der Statuszeile wird dann das "L" wieder gelöscht.

ALT+D — Das Directory des aktuellen Verzeichnis wird bei ALT+D angezeigt. Weder auf dem Drucker noch im Logfile sind diese Ausgaben sichtbar.

ALT+S — Aufruf der COMMAND-Oberfläche des Betriebssystems. Diese Funktion erlaubt danach die Eingabe von DOS-Befehlen. Da aber das Terminalprogramm weiterhin im Speicher steht, dürfen weder die verwendeten Files noch die aktive Schnittstelle durch diese Befehle verändert werden. Nach der Eingabe von EXIT kehrt man wieder in das Terminalprogramm zurück.

ALT+X — Terminalprogramm beenden. Dabei sollten alle vom KK-FORTH verwendeten Files geschlossen sein. Zur Sicherheit wird vorher abgefragt, ob das Programm wirklich verlassen werden soll.

ALT+Q Tasteneingaben können die Befehlsübertragung zwischen KK-FORTH und Terminalprogramm stören. Deshalb kann dies durch ein Kommando ($0002) oder über Tastatur verhindert werden. Bei blockierter Tastatur wird ein "X" in der Statuszeile angezeigt und die gedrückte Taste gespeichert. Bei Umstellung von Port oder Baudrate wird auch der Tastaturpuffer gelöscht.

ALT+C Die Nummer des COM-Ports kann nach ALT+C verändert werden. Dabei sind folgende Portadressen möglich:

COM1: $03F8	COM2: $02F8
COM3: $03E8	COM4: $02E8

ALT+B Die Baudrate kann beim PC-Terminalprogramm von 300 bis zu 115200 Baud verändert werden. Die Tasten 0 bis 9 sind dabei mit folgenden Baudraten belegt:

0:	115200	1:	57600	2:	38400
3:	19200	4:	12800	5:	9600
6:	2400	7:	1200	8:	600
9:	300				

ALT+T Um möglichst ohne Veränderung der unbenutzten Schnittstellen und ohne dauernde Umstellung der Vorgaben auszukommen, kann das aktuelle System unter einem beliebigen Manen abgespeichert werden.

ALT+E Durch Drücken von ALT+E kann der Empfangsspooler gelöscht werden. Dadurch werden die schon empfangenen Zeichen ignoriert und nicht mehr ausgegeben oder gespeichert.

4.1.2 Der Editor des Terminalprogrammes

Zusätzlich ist der nachfolgend beschriebene Bildschirmeditor integriert. Er wird aber nicht per Tastatur aufgerufen, sondern vom KK-FORTH bei den Befehlen **L** oder **V** gestartet. Dadurch wird sichergestellt, daß im gesamten System einheitliche Programmfiles verwendet werden.

4.1.3 Terminal-Befehlsschnittstelle

Erreicht wird diese für den Anwender transparente Befehlsübertragung durch ESC-Sequenzen. Sobald das Terminalprogramm das Zeichen mit Wert $1B empfangt, erwartet es weitere zwei Zeichen (höherwertiges Byte zuerst) und verbindet es zur Befehlsnummer. Entsprechend der Befehlsnummer werden dann die Kommandos ausgeführt und dazu weitere Daten ausgetauscht.

Allgemeines Protokoll

>1	ESC-Zeichen $1B an Terminalprogramm schicken
>2	Befehlsnummer schicken
<2	Fehlerflag
><	weiterer Datenaustausch

Bei allen übergebenen Werten werden immer die höchstwertigen Bytes zuerst geschickt. Sobald eine Fehlernummer ungleich 0 empfangen wird, ist die Befehlsübertragung beendet. Filenamen werden nicht mit einer Längenangabe geschickt, sondern durch ein 0-Ende abgeschlossen.

Vom Terminal unterstützte Befehle

$0000 ESC-Zeichen ausgeben

$0001 Terminal-Editor aufrufen
- >2 Screen-Nummer
- >2 Offset im Screen
- >$/0 Filename (ohne Längenangabe mit 0-Ende)
- <2 Fehlerflag nach Ende des Editors zurückliefern

$0002 Ein-/Ausschalten der Tastenblockierung für das Terminal
- >1 Flag (0=fre; sonst blockiert)

$0003 Abfrge, ob ein Zeichen von Tastatur bereitsteht
- <1 Flag ($FF=Zeichen bereit, sonst 0)

$0004 Tastenabfrage
- <1 Zeichen (0, wenn kein Zeichen im Tastaturpuffer)

$0101 Bildschirm löschen

$0102 Cursorposition abfragen
- <2 X-Position (0=links)
- <2 Y-Position (0=oben)

$0103 Cursorposition setzen
- >2 X-Position
- >2 Y-Position

$0104 Bildschirmgröße abfragen
- <2 Anzahl der Spalten (X)
- <2 Anzahl der Zeilen (Y)

$0201 Abfrage, ob File existiert
- >$/0 Filename
- <2 Fehlerflag (0: File ist vorhanden)

$0202 File neu anlegen
- >$/0 Filename
- <2 Fehlerflag
- <2 Handlenummer

$0203 File löschen
- >$/0 Filename
- <2 Fehlerflag

$0204 File öffnen
- >$/0 Filename
- <2 Fehlerflag
- <2 Handlenummer

$0205 File schließen
- >2 Handlenummer
- <2 Fehlerflag

$0206 Filegröße abfragen
- >2 Handlenummer
- <2 Fehlerflag
- <4 Filegröße in Bytes

$0207 Fileposition setzen
>2 Handlenummer
>2 Richtung (dir=0:ab Anfang ...)
>4 Position
<2 Fehlerflag

$0208 Fileposition abfragen
>2 Handlenummer
<2 Fehlerflag
<4 aktuelle Fileposition

$0209 Lesen aus dem File
>2 Handlenummer
>2 Länge n der zu übertragenden Daten in Bytes
<2 Fehlerflag
<n Daten empfangen

$020A Schreiben in das File
>2 Handlenummer
>2 Länge n der zu übertragenden Daten in Bytes
>n Daten
<2 Fehlerflag

$020B Erster Fileeintrag suchen
>2 Attribut
>$/0 Filename
<2 Fehlerflag
<1 Anzahl der Daten n
<n Angaben zum File

$020C Nächsten Fileeintrag suchen
<2 Fehlerflag
<1 Anzahl der Daten n
<n Angaben zum File

$020D Diskettendaten ermitteln
>2 Nummer des Laufwerks (0=Aktuell; 1=A ...)
<2 Fehlerflag
<4 Freier Speicherplatz in Bytes
<4 Gesamtgröße

4.2 Der bildschirmorientierte Editor

Entweder schon fest im Terminalprogramm eingebunden oder für andere Versionen nachladbar gibt es für das KK-FORTH einen Screeneditor. Er wird durch die Befehle **L** oder **V** aufgerufen und verwendet dann das aktuelle File. Es wird nur der momentan sichtbare Screen im Speicher gehalten. Der Rest des Files bleibt auf Diskette ausgelagert. Veränderungen in einem Screen werden erst dann zurückgeschrieben, wenn der Editor verlassen oder der Screen gewechselt wird. Deshalb können gerade durchgeführte Änderungen des aktuellen Screens durch Drücken der Taste F10 rückgängig gemacht werden.

Mit dem Befehl **FROM** kan im nachladbaren Editor ein zweites File angemeldet werden. Danach wird durch die Taste F4 die Files vertauscht und es können einfach mit den Zeilen-/Zeichenpuffer Programmteile übernommen werden.

Zusätzlich zu den für einen FORTH-Editor üblichen Befehlen können auch Leerscreens in das File eingefügt werden. Ist dabei der letzte Screen schon belegt, so wird das File vergrößert. Da leider das File nicht verkürzt werden kann, bleibt beim Löschen eines Screens der letzte Block leer.

Zusätzlich zum ladbaren Editor wird ein eigenständiger Editor mitgeliefert. Er ladet das ganze File in den Speicher des PC's und erlaubt deshalb eine von der Geschwindigkeit des Speichermediums unabhängige Veränderung des Files. Da beim Zurückschreiben auf Diskette das vorher existierende File gelöscht wird, kann das neue File auch leiner sein. Man sollte aber immer eine Sicherungskopie des Files besitzen, da bei einem Fehler während des Zurückschreibens der Editor sofort verlassen und deshalb das geänderte File nur unvollständig gespeichert wird.

Die nachfolgende Beschreibung der Tastenkombinationen gilt für das PC-Terminalprogramm und dem nachladbaren Editor des KKF_PC.COM. Er stimmt bis auf wenige Ausnahmen auch bei anderen KKF-Versionen. Abweichungen werden noch in der Zusatzbeschreibung erläutert. Das Zeichen ^ bedeutet, daß gleichzeitig mit der angegebenen Taste auch noch die CTRL- (oder Strg-)Taste gedrückt werden muß. Da CTRL alleine keine Aktion auslöst, sollte man sie immer zuerst Drücken.

Ungewohnt für Anfänger ist die Verwendung eines Zeilen- und Zeichenpuffers zum Verschieben von Programmteilen. Wie bei einem Stack wird dabei immer das zuletzt gespeicherte Zeichen (oder eine ganze Zeile) zuerst wieder ausgeben. In der Statuszeile kann dann aus der Angabe (L:000/C:000) entnommen werden, wieviele Zeilen (L=Lines) oder Zeichen (C=Character) noch gespeichert sind. Beim Verlassen des Editors wird der dazu verwendete Speicher zwischen Dictionary und Variablenbereich wieder freigegeben.

Cursorsteuerung:

^E oder Cursor_hoch	Eine Zeile höher
^X oder Cursor_tief	Eine Zeile tiefer
^S oder Cursor_links	Ein Zeichen links
^D oder Cursor_rechts	Ein Zeichen rechts
TAB	Zur nächsten 4er-Teilung
Shift+TAB	Zur vorherigen 4er-Teilung
^F	Zum nächsten Wortanfang
^A	Zum vorherigen Wortanfang
^Q B	Zum Zeilenanfang
^Q K	Zum Zeilenende-1
^Q E	Zum Screenanfang
^Q X	Zum Screenende-1
POS1	Zum ersten Zeichen der Zeile
ENDE	Hinter das letzte Zeichen der Zeile
^POS1	Zum ersten Zeichen des Screens
^ENDE	Hinter das letzte Zeichen des Screens
Return	Zum nächsten Zeilenanfang
^R oder Bild_hoch	Zum vorhergehenden Screen
^C oder Bild_tief	Zum nächsten Screen
^K R oder ^Bild_hoch	Zum ersten Screen
^K C oder ^Bild_tief	Zum letzten Screen
^Q G	Eingabe der gewünschten Screennummer
F4	Zum zweiten File / Cursorposition umschalten

Zwischenspeicherung von Zeichen und Zeilen:

F1	Zeichen speichern und löschen
F2	Zeichen speichern, Cursor rechts
F3	Zeichen einfügen

F5	Zeile speichern und löschen
F6	Zeile speichern, Cursor in die nächste Zeile
F7	Zeile einfügen

Steuerung der Zeicheneingabe und des Löschens:

^V	Insert-Modus umschalten (Anzeige: O oder I)
Einfg	Ein Leerzeichen einfügen
^G oder Entf	Zeichen unter dem Cursor löschen
^H oder Backspace	Zeichen vor dem Cursor löschen
F8	Rest der Zeile löschen
^Y	Aktuelle Zeile entfernen
^N	Leerzeile einfügen
^K N	Leerscreen einfügen
^K Y	Aktuellen Screen entfernen
^Backspace	Nächste Zeile ab Cursorposition übernehmen
^Return	Rest dieser Zeile in die nächste Zeile

Suchen und Ersetzen:

^Q F	String suchen (u=Option für Rückwärts-Suche)
^Q A	String suchen und ersetzen (mehrere Optionen)
^T	Nächstes Wort in Kleinschrift wandeln
^U	Nächstes Wort in Großschrift wandeln

Speicherung:

F10	Änderungen in diesem Screen rückgängig machen
F9	Eingabe der ID-Kennung (nicht bei EDITOR.COM)
ESC	File speichern, Editor verlassen

Zusätze bei EDITOR.COM:

^K S	File speichern, danach weiter editieren
^K D	Editor ohne Speicherung des Files verlassen

4.3 Assembler

FORTH ist sehr schnell. Trotzdem gibt es Fälle, bei dem diese Programmiersprache zu langsam ist oder nicht die gewünschten Möglichkeiten bietet. In solchen Fällen greift man, wie auch bei anderen Programmiersprachen, zum Assembler. Jedoch bietet hier FORTH die Möglichkeit, daß die Arbeitsumgebung nicht verlassen werden muß und die mit einem Assembler erzeugten Befehle wie normale FORTH-Wörter behandelt und getestet werden können.

Um dies zu erreichen, wird für die meisten KKF-Versionen ein ladbares Assemblerfile mitgeliefert. Obwohl jeder Prozessor seine eigene Mnemonics und Registerbezeichnungen hat, sind viele Eigenschaften von KKF-Assembler vereinheitlicht worden:

* Einleitung der Assemblerdefinition mit **CODE** , **PROC** oder **;CODE**
* Ende der Definition mit **END-CODE** oder **END-PROC**

* Einfache Definition von Makrobefehlen möglich
* Man befindet sich auch während der Befehlsdefinition im Interpretermodus
* Es können alle FORTH-Befehle zur Parameterberechnung verwendet werden
* Die Befehlsdefinitionen erfolgen in UPN-Logik
* Rücksprung zum FORTH-Interpreter mit **NEXT,**
* Lokale Labels werden mit **1$:** ... **9$:** markiert und mit **1$** bis **9$** referenziert
* Kurzsprünge werden durch Assembler-Kontrollstrukturen erzeugt

Das KK-FORTH ist natürlich selbst in FORTH geschieben. Dabei wurden viele Definitionen wegen der höheren Ausführungsgeschwindigkeit in Assemblercode realisiert. Bei den nachfolgenden Beispielen handelt es sich um einfache Befehle für verschiedene Prozessoren.

* 8086-Assemblerdefinitionen (KKF_PC.COM)
  (Achtung: oberster Datenstackwert liegt in Register BX = TOS)

```
Code count   ( csa -- addr n )
   tos w mov,   w ) tosl mov,   tosh tosh or,
   w inc,   w push,   next,
 End-Code

Code ?dup   ( n -- n n | 0 )
   tos tos or,   0<> IF,  tos push,  THEN,  next,
 End-Code
```

* Z80-Assemblerdefinitionen (KKF_8415.COM)

```
Code count   ( csa -- addr n )
   hl pop,   e ,( hl ld,   d , 0 ld,   hl inc,
   hl push,   de push,   next,
 End-Code

Code ?dup   ( n -- n n | 0 )
   hl pop,   hl push,   a , l ld,   h or,
   nz IF,  hl push,  THEN,  next,
 End-Code
```

Wie man aus diesen Beispielen sieht, ist die Erzeugung eines Assemblerbefehls genauso einfach wie die Definition eines FORTH-Befehls. Da man dabei aber auf alles zugreifen kann, muß man einige von FORTH belegte Register retten. Für viele von FORTH verwendeten Register wurde im Assembler ein ALIAS-Name definiert (z.B. TOS, FRP, FSP).

Da Assemblerbefehle immer interpretiert werden, kann man sehr einfach Makrodefinitionen erzeugen. Dadurch ist es sehr einfach möglich, eine aus mehreren Befehlen zusammengesetzte Pseudo-Adressierungsart zu erzeugen oder häufig verwendete Befehlsfolgen zusammenzufassen:

```
: ?hlpush,   ( HL zum Stack, wenn ungleich 0 )
   a , l ld,   h or,   nz IF,  hl push,  THEN,  ;

Code ?dup   ( n -- n n | 0 ) ( Mit Makro-Verwendung )
   hl pop,   hl push,   ?hlpush,   next,
 End-Code
```

Da der Assembler nur zur Erzeugung des Befehls verwendet wird und danach keine aktive Tätigkeit im System mehr besitzt, ist es aus Platzgründen oft sinnvoll, den Assembler nur für diese Definitionen auf den Heap zu laden und danach mit HCLEAR wieder zu entfernen. Dadurch bleiben dem Anwender die 2-8KByte für weitere Befehle oder Daten frei.

4.4 Disassembler

Falls man sich einmal die Code-Definitionen des FORTH oder sogar die Betriebssystemroutinen ansehen will, so steht dafür ein zum KKF ladbarer Disassembler zur Verfügung. Dabei gibt z.B. der 8086-Disassembler das Ergebnis entsprechend der FORTH-Mnemonics an. Der Z80-Disassembler dagegen erzeugt reine Zilog-Mnemonics.

Als Grundbefehle stehen folgende Befehle zur Verfügung:

addr dis	Ab Adresse addr disassemblieren
addr count ndis	count Bytes ab addr disassemblieren

Der Disassembler generiert immer 10 Zeilen und wartet dann auf eine Bestätigung. Mit CTRL+X kann die Ausgabe abgebrochen werden. Beim Laden des Disassemblers wird oft ein Vokabular namens DISASS erzeugt.

4.5 Debugger

Manchmal hat man einen Befehl, bei dem im Listing kein Fehler auffällt aber trotzdem ein falsches Ergebnis zurückgeliefert wird. In solchen hartnäckigen Fällen sollte man zu einem Debugger greifen, der Programme im Einzelschrittmodus bearbeiten kann. Für Z80- und 8086-kompatible Prozessoren wird im KK-FORTH ein Debugger mitgeliefert, der dies ermöglicht. Jedoch muß man bei der EPROM-Version des 8086-KKF darauf achten, daß die spezielle Debuggerversion verwendet wird.

Nach dem Laden des Debuggers stehen dem Anwender fünf zusätzliche Befehle zur Verfügung:

debug <name>	Befehl <name> debuggen
min max debug	Bereich von min bis max debuggen
unbug	Debugger wieder abschalten
decom <name>	Befehl <name> dekompilieren
cfa (decom	Befehl mit angegebener CFA dekompilieren

Startet man den Debugger z.B. mit DEBUG D. , so passiert momentan noch überhaupt nichts sichtbares. Der Debugger wird nur aktiviert und wartet dann, bis das FORTH-Programm den Befehl **D.** ausführt. Erst dann reagiert der Debugger und listet den nächsten Befehl mit Stackangaben auf und erwartet eine Eingabe.

```
$1234. d.                        ( Befehl aufrufen )

18A8 02E6 FALSE                    4 /   2: 1234 0000
^    ^    ^                        ^     ^  > Stackwerte (TOS zuletzt)
^    ^    ^                        ^     > Datenstacktiefe
^    ^    ^                        > Returnstacktiefe
^    ^    > Name
^    > Codefeldadresse des Befehls
> Aktueller Programmzeiger
```

Der Cursor bleibt hinter den Datenstackwerten stehen und erwartet eine Eingabe. Wie auch im Screen 0 des Debuggers beschrieben sind folgende Tasten erlaubt:

N	(für NEST)	Diesen Befehl selbst debuggen
U	(für UNNEST)	Debugger erst wieder im aufrufenden Wort aktivieren
L	(für UNLOOP)	Debugger erst hinter der Schleife wieder aktivieren
F	(für FORTH)	Eine Befehlszeile eingeben
jede andere Taste		Wort ausführen

Wenn man jetzt die Leertaste drückt, so kann der Befehl schrittweise bearbeitet werden. Wird mit N ein Unterbefehl bearbeitet, so werden diese Zeilen eingerückt. Am Ende des Befehls wird automatisch im aufrufenden Wort weitergemacht.

```
18AA 1881 D.R                  4 /   3: 1234 0000 0000  4660
18AC 16C8 SPACE                4 /   0:
18AE 018D EXIT                 4 /   0:  ok
```

Der Wert 4660 wird erst bei Ausführung des Befehls **D.R** ausgegeben. Bei dem unbekannten Befehl wird ??? statt dem Namen ausgegeben. Sobald das Tracen beendet ist, sollte unbedingt mit **UNBUG** der Debugger deaktiviert werden. Dies ist deshalb sinnvoll, da bei jedem FORTH-Befehl der Debugger untersucht, ob eine Ausgabe notwendig ist und dadurch das Programm verlangsamt.

Mit der Taste F können auch zusätzliche Befehle (wie z.B. **DUMP**) eingegeben werden. Solange das zuletzt übergebene Zeichen ein Leerzeichen ist, werden weitere Zeilen angefordert. Ansonsten wird der angezeigte Befehl ausgeführt und die nächste Taste (N, U, L, F oder Space) erwartet.

Dank eines eifrigen KKF-Programmierer konnte das Debugger-File um einen Dekompiler erweitert werden. Dabei sucht das Wort **DECOM** automatisch das Ende des Befehls und ist in der Lage, auch DEFER-, DOES>- und UVECTOR-Befehle zu dekompilieren. Auch hier wird wie im Disassembler die Ausgabe alle 10 Zeilen angehalten.

4.6 Beispielprogramme

4.6.1 Errortrapping für Menüprogramme

FORTH besitzt normalerweise eine so armselige Fehlerbehandlung, daß oft nur ein Rücksprung in den Interpreter oder ein Neustart des Anwenderprogrammes möglich ist. Die hier vorgestellten Befehle erlauben aber einen gezielten Wiedereinsprung bei einer bestimmten Routine. Darüber hinaus ist die Fehlerbehandlung kaskadierbar: In Unterprogramme können Einsprünge definiert sein, die beim Verlassen des Wortes wieder vergessen werden. Notwendig dazu ist aber eine Umleitungsmöglichkeit der Fehlerbehandlung, wie sie im KK-FORTH mit **ERRORHANDLER** vorgesehen ist. Darüber hinaus laufen in dem KK-FORTH alle Fehler über Fehlernummern, die erst vom ERRORHANDLER-Programm ausgewertet werden. Natürlich ist dieses ganze Konzept der Fehlerbehandlung nicht neu und wurde in ähnlicher Ausführung auch schon von Klaus Schleisiek in der vierten Dimension (gleichzeitig mit den lokalen Variablen) erwähnt.

Initialisiert wird das Errortrapping mit **ERRORTRAP** . Dieser Befehl setzt die Zeiger **ERP1** und **ERP2** zurück und verändert die Fehlerbehandlungroutine. Wenn jetzt eine Fehlerbehandlung eingeleitet wird, führt sie über den DEFER-Vektor '**ENDTRAP** zu **QUIT** .

Ansonsten kann mit **>*<** ein Einsprungpunkt definiert werden, bei dem nach der Fehlerausgabe das Programm fortgesetzt wird. Da nach einem Fehler oft noch Werte auf dem Stack übrig bleiben, wird dieser immer beim Wiedereinsprung gelöscht. Es können aber trotzdem Werte über diesen Einsprungpunkt auf dem

Datenstack übergeben werden. Der oberste Wert gibt dann an, ob der Befehl zum ersten mal aufgerufen (Flag=0) oder nach einem Fehler (Flag=1) wiederholt wurde.

Oft kommt es vor, daß am Anfang eines Menüpunktes eine Bildschirmmaske aufgebaut wird. Ein Breakpoint muß meist vorher gesetzt werden, da diese Maske natürlich nach Fehler wieder ausgegeben werden soll. Kommt es aber hier zu einem Fehler, so läuft das FORTH in einer Endlos-Schleife. Abhilfe dazu schaffen die Befehle { und } , die den letzten Breakpunkt innerhalb ihrer Struktur unwirksam machen. Es kommt dann bei größeren Systemfehlern zum netten Effekt des "kaskadierten Rücksturzes" manchmal bis zu **QUIT** .

```
Include ERRTRAP.SCR                ( Programm laden )
...                                ( Ausgaben beim Laden )

demo                               ( Ohne ERRORTRAP )
Wort : DEMO  ausgeführt
                  0 ( $0000 , &00000 )
               4660 ( $1234 , &04660 )
Wort : DEMO2 ausgeführt
                  0 ( $0000 , &00000 )
               4660 ( $1234 , &04660 )
 "DEMO" ? Error $4002  ok

dclear  ok                         ( Datenstack löschen )
errortrap  ok                      ( ERRORTRAP aktivieren )
demo
Wort : DEMO  ausgeführt
                  0 ( $0000 , &00000 )
               4660 ( $1234 , &04660 )
Wort : DEMO2 ausgeführt
                  0 ( $0000 , &00000 )
               4660 ( $1234 , &04660 )

Error $4002
Wort : DEMO  ausgeführt
                  1 ( $0001 , &00001 )
Wort : DEMO1 ausgeführt
                  0 ( $0000 , &00000 )

Error $4001
Wort : DEMO1 ausgeführt
                  1 ( $0001 , &00001 )

Verlasse DEMO1

Verlasse DEMO
 ok
```

Bei dem Beispiel wird durch **DEMO** zuerst **DEMO2** aufgerufen. Da dort ein Fehler verursacht wird, der wegen { } erst vom darüberliegenden Befehl bearbeitet wird, erfolgt ein Wiedereinsprung in **DEMO** . Da dies am Flag<>0 erkannt wurde, kann jetzt noch der Befehl **DEMO1** aufgerufen und dann das Wort verlassen werden. In **DEMO1** erfolgt die Fehlerbehandlung durch den eigenen Fehlereinsprung.

Die Verwaltung mit **ERP1** und **ERP2** wurde deswegen so kompiziert gestaltet, da es im KK-FORTH keinen einheitlichen Befehl zum Zugriff auf Daten des Returnstacks gibt. Die einzige

direkte Manipulationsmöglichkeit für den Returnstack sind die Befehle **RP@** und **RP!** , bei dem die zu übergebenden Parameter aber Versionsabhängig sind.

4.6.2 FFT-Analyse vom Signalen

Viele Leute haben mich schon gefragt, ob ich nicht Routinen für eine FFT-Analyse habe. Dieses Beispielprogramm (Listing im Anhang G) soll deshalb mit dem Listing und einem Beispiel diesen Nachfragen abhelfen. Die Kontrollstruktur **FOR** ... **NEXT** ist deshalb in diesem Beispiel, weil das Programm für den FORTH-Prozessor NC4000P entwickelt wurde. Die **FOR** ... **NEXT** -Struktur ist aber in allen KKF-Versionen verfügbar. Ein Nachteil des Prozessors ist, daß er Schwierigkeiten bei vorzeichenbehafteter 16Bit-Arithmetik hat. Deshalb wurden alle Skalierungen so gewählt, das bei Multiplikation und Division der Absolutbetrag des Wertes bei 0..16383 liegt.

Da das Programm bis auf die Datenfelder der Signale keine Werte speichert, mußten Routinen für Sinusberechnung, Quadratwurzelermittlung und Bitumkehr implementiert werden. Da sie unabhängig von der FFT-Analyse verwendbar sind, wurde ihnen auch etwas mehr Aufmerksamkeit gewidmet.

Die Größe des verwendeten Datenfeldes ist von den Konstanten **LN(P** und **POINTS** abhängig. Die Werte müssen so gewählt werden, daß mindestens 2*POINTS 32Bit-Werte im freien Speicher zwischen `HERE` und `VDP @` verfügbar sind. **LN(P** ist dabei eine gerade Integerzahl zwischen 4 und 12 (für 4096 Punkte) und **POINTS** gleich 2^ln(p . Von der Rechenzeit und dem Speicherbedarf her gut verwendbar sind Werte von 8 bis 10 (256 bis 1024).

Sinus und Cosinusberechnung in FORTH

Natürlich wird der Sinus über eine Reihenentwicklung ermittelt. In einer älteren ELEKTRONIK-Zeitschrift (Ausgabe 7/84) sah ich vor vielen Jahren einmal die Formel:

$$\sin x = 0.9997\,x - 0.165629\,x^3 + 0.0074886\,x^5$$

Diese Formel soll einen relativen Fehler kleiner als 2^-15 im ersten Quadranten besitzen und der Winkel wird in rad angegeben. Durch geeignete Umformung entstand die im Screen 7 angegebene Formel, in der ein 16Bit-Wert (0=0 Grad; 16384=90 Grad ...) in seinen Sinuswert umgewandelt und mit 16384 multipliziert wird. Um Schwierigkeiten in der FFT-Analyse aus dem Weg zu gehen, wird der größte Wert auf 16383 (statt 16384) heruntergesetzt.

Ich habe noch keine ausführlichen Test's mit der Formel und den Ergebnissen durchgeführt, jedoch glaube ich (schon wegen der geringen Fehler in der FFT), daß er bei ±2 Bit liegt.

Der Cosinus ist nur ein verschobener Sinus und deshalb muß nur 16384 zum Winkel-Wert addiert werden.

Quadratwurzel-Berechnung

Die Quadratwurzel eines 32Bit-Wertes wird genau so berechnet, wie man es vor vielen Jahren einmal in der Schule erlernt hat. Dadurch, daß im Binärsystem die Multiplikation zum Shiften um ein oder zwei Stellen wird, vereinfacht sich die Angelegenheit. Die gleiche Berechnungsart wird auch oft direkt in Assembler und meist nur mit den internen Register (falls groß genug) durchgeführt.

Im einzelnen läuft die Berechnung wie folgt ab:

* Summe und Wurzelwert auf 0 setzen
* Wiederholungsschleife (16 mal)
  - 2 Bits nach rechts (* 4) von Quadrat in Summe shiften
  - Wert = Wert * 2 + 1
  - Test, ob Summe kleiner als die Wurzel
    ja: Wert = Wert - 1

nein: Summe = Summe - Wert + 1
* Summe und Quadrat vom Stack entfernen

Bei voller Ausnützung des 32Bit-Wertebereiches kommt es zum Überlauf in der Summe. Dies ist nur durch Verwendung von 32Bit-Werten auch bei Summe und Wurzel zu verhindern, geht aber in die Rechenzeit ein.

Bitumkehr

Bei der FFT-Analyse werden immer zwei komplexe Werte miteinander multipliziert. Um die Position der Werte zu bestimmen, wird auch eine sogenannte Bitumkehr benötigt.

Je nach gewählter Punkteanzahl (im Programm 256) werden die zur Bezifferung der Position notwendigen Bits (0..255 benötigen 8 Bits) in ihrer Reihenfolge umgedreht. Aus dem Binärwert %1110010 (=226) wird dann %0100111 (=71).

Durch Umsetzung dieser Routine in Maschinensprache (ist es ja schon für den NC4000) läßt sich viel Zeit sparen, weil es schon im Beispiel über tausend mal verwendet wird.

FFT-Analyse

Der hier beschriebene Algorithmus ist eine echte FFT-Analyse nach COOLEY-TUKEY. Die Skalierung wurde so gewählt, daß immer der Scheitelwert der Sinus-/Cosinusschwingung und nicht der Effektivwert zurückgeliefert wird.

Der Kern des FFT ist eine komplexe Multiplikation zweier Werte, dessen Position durch Wert des Schleifenzählers und der Bitumkehr abhängiger Werte bestimmt wird. Bei der komplexen Multiplikation werden immer nur die obersten 16 Bit (gerundet) verwendet. Das entstehende 32Bit-Ergebnis wird dann mit den gespeicherten 32Bit-Wert verrechnet. Durch Umstellung dieser Routine auf volle 64 Bit-Zwischenwerte kann noch etwas Genauigkeit gewonnen werden.

Das Signal, daß als 16Bit-Wert (vorzeichenlos auf 32 Bit erweitert) im Real-Datenfeld vorliegt, wird der FFT-Analyse unterzogen und danach als Liste der Scheitelwerte zurückgeliefert. Die Adressierung von Signalposition und Amplitudenwert erfolgt mit dem Befehl **RE(N** . Dieser erwartet auf dem Stack die Nummer der Position (0..255) und liefert dessen Adresse zurück. Es muß immer ein 32Bit-Wert gelesen und geschrieben werden. Nach der FFT-Analyse ist in `0 RE(N` der Gleichspannungsanteil abgelegt.

Ablauf der FFT-Analyse im einzelnen:

* Vorbereitung des Datenspeichers
  - Realteil mit 32768 multiplizieren
  - Imaginärteil löschen
* Eigentliche FFT-Analyse
  - Dreifach ineinandergefügte Schleifen mit Kern **KMU**
  - Korrektur der Reihenfolge des Spektrums (**REVERSEL**)
* Berechnung der Amplitudenwerte aus komplexen Spektrum
  - Realteil = SQRT (Realteil^2 * Imaginärteil^2)
  - Imaginärteil löschen

Der Algorithmus selbst ist schon in vielen Büchern erwähnt und als Beispiel in vielen Programmiersprachen angegeben.

Beispiel: FFTTEST

Mit dem Befehl **FFTTEST** soll eine Schwingung der FFT-Analyse unterzogen und die ersten 10 Amplituden ausgegeben werden. Im Beispiel handelt es sich um ein Signal, daß aus DC-Anteil und der 2ten, 5ten und 6ten Oberschwingung mit unterschiedlichen Amplituden

zusammengesetzt ist. Es muß nur darauf geachtet werden, daß es kein Werteüberlauf in der 16Bit-Darstellung gibt.

Übrigens können auch 32Bit-Werte direkt an die FFT-Analyse weitergereicht werden, wenn die Vorbereitung in FFT entsprechend abgeändert wird.

Die auch im Beispiel schon ersichtlichen Fehler kommen hauptsächlich durch Rundungsfehler vor allem in den Sinus/Cosinus-Werten. Man kann aber damit leben, wenn das Ausgangssignal möglichst groß sind. Da bei mir die FFT-Analyse meist mit linksbündigen 12Bit-Werten rechnet, stören die 2-3 Digit Ungenauigkeit kaum.

Anhang A

Kurzglossar

A.1 Beschreibung der Abkürzungen

Bezeichnung der Stackparameter:

(C: ...)	**Veränderungen auf dem Datenstack während des Kompilierens**
(R: ...)	**Veränderungen auf dem Returnstack**
(name ; ...)	**Es wird noch ein Befehlsname erwartet**
flag	**Flag (0=ff=Falsch; sonst tf=Wahr)**
b	**Byte**
n	**Einfachgenauer, vorzeichenbehafteter Wert**
+n	**Einfachgenauer, positiver Wert oder 0**
u	**Einfachgenauer, vorzeichenloser Wert**
w	**Nicht definierter, einfachgenauer Wert**
addr	**Adresse**
sys	**Systemabhängie Speicheradresse**
lfa	**Linkfeld-Adresse**
nfa	**Namensfeld-Adresse**
cfa	**Codefeld-Adresse**
pfa	**Parameterfeld-Adresse**
csa	**Counted-String-Adresse**
d	**Doppeltgenauer, vorzeichenbehafteter Wert**
+d	**Doppeltgenauer, positiver Wert oder 0**
ud	**Doppeltgenauer, vorzeichenloser Wert**
wd	**Nicht definierter, doppeltgenauer Wert**
ptr	**32Bit-Zeiger für externen Speicherzugriff** **PC: ptr = seg:addr** **RTX: ptr = bank:addr** **68000 ptr = addrh:addrl**

Bezeichnung der Befehlsart:

I	**Dieser Befehl ist IMMEDIATE**
R	**Dieser Befehl ist RESTRICT**
S	**System-Variable**
U	**USER-Variable**
V	**Variable**
UT	**UTABLE:-Definition**
UV	**UVECTOR-Befehl**
D	**Mit NOOP vorbelegter DEFER-Befehl**
DX	**DEFER-Befehl mit nachfolgender (versteckter) Runtime-Routine**
X	**Dieser Befehl ist nicht in allen KKF-Versionen verfügbar**

A.2 Nach Gruppen geordnete Befehlsliste

Konstanten, Adressen und Variablen

TRUE	**(-- -1)**		**Wahr-Wert für alle Vergleiche im KKF**
FALSE	**(-- 0)**		**Falsch-Wert für alle Vergleiche im KKF**
#BL	**(-- $20)**		**Liefert ASCII-Wert des Leerzeichens**
#DELIN	**(-- sys)**		**Liefert ASCII-Wert der DEL-Taste**
#DELOUT	**(-- sys)**		**ASCII-Wert für Cursor-Links bei Ausgabe**
#CR	**(-- $0d)**		**Liefert ASCII-Wert der RETURN-Taste**
#ESC	**(-- $1b)**		**Liefert ASCII-Wert der ESC-Taste**
#BRK	**(-- $18)**		**Liefert ASCII-Wert der Abbruchtaste (meist CTRL+X)**
#B/BLK	**(-- $400)**		**Anzahl der Bytes pro Block (Screen)**
#L/BLK	**(-- $10)**		**Anzahl der Zeilen pro Block**
#C/L	**(-- $40)**		**Anzahl der Zeichen pro Zeile**
#TIB-MAX	**(-- sys)**		**Größe des Eingabepuffers (meist 79 Zeichen)**
#RO	**(-- sys)**		**Flag für (FILE-OPEN : File nur lesen**
#WO	**(-- sys)**		**Flag für (FILE-OPEN : File nur schreiben**
#RW	**(-- sys)**		**Flag für (FILE-OPEN : File lesen und schreiben**
SYSCON	**(-- sys)**		**Liefert Anfangsadresse der Systemkonstanten**
SYSVAR	**(-- sys)**		**Liefert Anfangsadresse der Systemvariablen**
(SYSVAR (SAVE)	**(-- sys)**		**Liefert Zieladresse für die Kopie der Systemvariablen**
SYSVARLEN@	**(-- sys)**		**Liefert Länge des Systemvariablenbereiches**
HERE	**(-- addr)**		**Liefert Adresse des aktuellen Dictionaryende**
VHERE	**(-- addr)**		**Liefert Adresse des aktuellen Variablenende**
HEAP	**(-- addr)**		**Liefert Adresse des HEAP-Anfangs**
HEAP?	**(addr -- flag)**		**Flag ist wahr, wenn addr im Heap liegt**
TASKADDR@	**(-- addr)**		**Liefert Adresse des aktuellen USER-Bereiches**
TASK0	**(-- addr)**		**Liefert USER-Adresse des Standardtask**
FIRST	**(-- sys)**		**Anfangsadresse des Diskettenpuffers**
LIMIT	**(-- sys)**		**Erste freie Speicheradresse über dem KKF**
VOC-LINK	**(-- sys)**	**S**	**Enthält Zeiger auf zuletzt definiertes Vokabular**
DP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Dictionaryende**
VDP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Variablenanfangs**
VLEN	**(-- sys)**	**S**	**Enthält aktuelle Länge des Variablenbereiches**
HDP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Heap-Anfangs**
HLEN	**(-- sys)**	**S**	**Enthält aktuelle Länge des Heap-Bereiches**
TDP	**(-- sys)**	**S**	**Enthält Anfangsadresse des Taskbereiches**
TASKADDR	**(-- sys)**	**S**	**Enthält USER-Adresse des aktuellen Tasks**
TLEN	**(-- sys)**	**S**	**Enthält Anzahl der belegten Bytes in den Tasks**
TASK-LINK	**(-- sys)**	**S**	**Enthält USER-Adresse des zuletzt definierten Tasks**
TASKS	**(-- sys)**	**S**	**Enthält Anzahl der definierten Tasks**
MAXTLEN@	**(-- len)**		**Liefert verwendete Gesamtlänge der Tasks**
SFLAG	**(-- sys)**	**S**	**Enthält Systemflag (einzelne Bits verwendet)**
UFLAG	**(-- sys)**	**S**	**Enthält Anwenderflag (frei verwendbar)**
WDP@	**(-- addr)**		**Liefert Anfangsadresse des Arbeitsbereiches für WORD**
PAD	**(-- addr)**		**Liefert Adresse des Stringbereiches für Zahlenausgabe**
S0	**(-- addr)**	**U**	**Enthält Anfangsadresse des Datenstack-Speichers**
R0	**(-- addr)**	**U**	**Enthält Anfangsadresse des Returnstack-Speichers**
STATE	**(-- addr)**	**U**	**Flag ob Text interpretiert (STATE=0) oder kompiliert wird**
CURRENT	**(-- addr)**	**U**	**Enthält Adresse+2 des aktuellen Kompiler-Vokabular**
CONTEXT	**(-- addr)**		**Liefert Adresse des obersten Suchvokabulars**
BLK	**(-- addr)**	**U**	**Enthält die Nummer des aktuellen Blocks (0=TIB)**
>IN	**(-- addr)**	**U**	**Enthält den Offset in den interpretierenden Puffers**
SCR	**(-- addr)**	**U**	**Enthält Nummer des zuletzt gelisteten oder fehlerhaften Screens**
R#	**(-- addr)**	**U**	**Enthält Position des letzten Fehlers im Text/Screen**
SPAN	**(-- addr)**	**U**	**Enthält Anzahl von Zeichen beim letzten EXPECT**
>TIB	**(-- addr)**	**U**	**Enthält Zeiger auf aktuellen Eingabepuffer**
TIB	**(-- addr)**		**Liefert Adresse des aktuellen Eingabepuffers**
#TIB	**(-- addr)**	**U**	**Enthält Anzahl der Zeichen im Eingabepuffer**
BASE	**(-- addr)**	**U**	**Enthält aktuelle Zahlenbasis**
HLD	**(-- addr)**	**U**	**Enthält Anfangsadresse des <# # #s #> -Zahlenstrings**

DPL	**(-- addr)**	**U**	**Enthält Anzahl der Ziffern nach einem Punkt nach >NUMBER**
INPUT	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Eingabebefehle**
OUTPUT	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Ausgabebefehle**
DISC	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Filebefehle**
ERRORHANDLER	**(-- addr)**	**U**	**Enthält CFA der aktuellen Fehlerroutine**
FILE-ID	**(-- addr)**	**V**	**Enthält Handlenummer des aktuellen Screenfiles**
FILE-FCB	**(-- addr)**	**V**	**Enthält Zeiger auf aktuellen Filename (CSA mit 0-Ende)**
FILE-LINK	**(-- addr)**	**V**	**Enthält Zeiger auf zuletzt definierten Fileeintrag**
INDENT?	**(-- addr)**	**V**	**Enthält Anzahl der auszugebenden Leerzeichen für INDENT**
NEXT-LINK	**(-- addr)**	**V**	**Enthält Zeiger auf zuletzt definierten (NEXT-Verkettung**
CODE?	**(-- addr)**	**V**	**Enthält Adresse des letzten Dictionaryeintrages mit ,C oder ,A**
LAST	**(-- addr)**	**V**	**Enthält NFA des zuletzt definierten Befehls**
>HEAP?	**(-- addr)**	**V**	**Enthält Anzahl headerlos zu definierenden Befehle**
CAPS	**(-- addr)**	**V**	**Enthält Flag, ob Befehle nur in Großschrift sind**

Datenstack- und Returnstackbefehle

SP@	**(-- addr)**		**Liefert aktuellen Datenstackadresse oder -tiefe**
SP!	**(addr --)**		**Setzt neue Datenstackadresse oder -tiefe**
DEPTH	**(-- n)**		**Anzahl der Stackeinträge vor Ausführung des Befehls**
'DCLEAR	**(??? -- ???)**	**D**	**Befehl zum Entfernen eigener Werte vor DCLEAR**
DCLEAR	**(??? --)**		**Löschen des Datenstacks**
ROLL	**(wu wt ... wa u -- wt ... wa wu)**		**Rotieren der obersten u Stackeinträge (ohne u)**
SWAP	**(w1 w2 -- w2 w1)**	**U**	**Vertauschen der obersten Datenstackeinträge**
ROT	**(w1 w2 w3 -- w2 w3 w1)**		**Rotieren der obersten drei Stackeinträge**
-ROT	**(w1 w2 w3 -- w3 w1 w2)**		**Umkehrung zum Befehl ROT**
2SWAP	**(dw1 dw2 -- dw2 dw1)**		**Oberste Wertepaar vertauschen**
2ROT	**(dw1 dw2 dw3 -- dw2 dw3 dw1)**		**Rotieren der obersten drei Wertepaare**
-2ROT	**(dw1 dw2 dw3 -- dw3 dw1 dw2)**		**Umkehrung zum Befehl 2ROT**
PICK	**(wu ... w1 w0 u -- wu ... w1 w0 wu)**		**Kopiert u-ten Stackeintrag zum TOS**
DUP	**(w -- w w)**		**Dupliziere w**
OVER	**(w1 w2 -- w1 w2 w1)**		**Kopiert den zweiten Stackeintrag zum TOS**
TUCK	**(w1 w2 -- w2 w1 w2)**		**Kopiert TOS unter den nächsten Stackeintrag**
2DUP	**(dw1 -- dw1 dw1)**		**Duplizieren des obersten Wertepaares**
2OVER	**(dw1 dw2 -- dw1 dw2 dw1)**		**Duplizieren des zweiten Stackpaares**
2TUCK	**(dw1 dw2 -- dw2 dw1 dw2)**		**Kopiert obere Wertepaar unter das zweite Wertepaar**
?DUP	**(n -- n n) oder (0 -- 0)**		**Dupliziert n nur, wenn der Wert ungleich 0 ist**
DROP	**(w --)**		**Entfernt w vom Datenstack**
NIP	**(w1 w2 -- w2)**		**Entferne zweiten Stackeintrag**
2DROP	**(dw --)**		**Oberstes Wertepaar entfernen**
2NIP	**(dw1 dw2 -- dw2)**		**Zweites Wertepaar entfernen**
S>D	**(n -- d)**		**Wandlung eines 16Bit-Wertes in ein 32Bit-Wert**
D>S	**(d -- n)**		**Wandlung eines 32Bit-Wertes in einen 16Bit-Wert**
RP@	**(-- addr)**		**Liefert aktuelle Returnstackadresse oder -tiefe**
RP!	**(addr --)**	**R**	**Setzt neue Returnstackadresse oder -tiefe**
RDEPTH	**(-- n)**		**Anzahl der Werte auf dem Returnstack**
'RCLEAR	**(--)**	**D**	**Befehl zum Entfernen eigenener Werte vom Returnstack**
RCLEAR	**(-- ; R: ??? --)**	**R**	**Löschen des Returnstacks**
>R	**(w -- ; R: -- w)**	**R**	**Wert zum Returnstack bringen**
R>	**(-- w ; R: w --)**	**R**	**Übertragt obersten Returnstackwert zum Datenstack**
R@	**(-- w ; R: w -- w)**	**R**	**Kopiert obersten Returnstackwert zum Datenstack**
RDROP	**(-- ; R: w --)**	**R**	**Entfernt obersten Returnstackwert**
2>R	**(w1 w2 -- ; R: -- w1 w2)**	**R**	**Oberstes Wertepaar zum Returnstack bringen**
2R>	**(-- w1 w2 ; R: w1 w2 --)**	**R**	**Oberstes Returnstackpaar zum Datenstack bringen**
2R@	**(-- w1 w2 ; R: w1 w2 -- w1 w2)**	**R**	**Oberstes Returnstackpaar zum Datenstack kopieren**
2RDROP	**(-- ; R: w1 w2 --)**	**R**	**Oberstes Returnstackpaar entfernen**
PUSH	**(addr -- ; R: -- w addr pop)**	**R**	**Variable bis zum nächsten EXIT merken**

Arithmetik, Logik und Vergleiche

+	**(n1 n2 -- n3)**	**Addiert n2 zum Wert n1**
-	**(n1 n2 -- n3)**	**Subtrahiert n2 von n1**
NEGATE	**(n1 -- n2)**	**Subtrahiert n1 von Null**
?NEGATE	**(n1 n2 -- n1 | -n1)**	**Negiert n1, wenn n2 negativ ist**
ABS	**(n1 -- +n2)**	**Subtrahiert n1 von 0, wenn n1 negativ ist (Absolutwert)**
2-	**(n -- n-2)**	**n um 2 erniedrigen**
1-	**(n -- n-1)**	**n um 1 erniedrigen**
1+	**(n -- n+1)**	**n um 1 erhöhen**
2+	**(n -- n+2)**	**n um 2 erhöhen**
AND	**(mask1 mask2 -- mask3)**	**Logische AND-Verknüpfung**
OR	**(mask1 maks2 -- mask3)**	**Logische OR-Verknüpfung**
XOR	**(mask1 mask2 -- mask3)**	**Logische XOR-Verknüpfung**
NOT	**(w1 -- w2)**	**Invertiert aller Bits in w1**
ASHIFT	**(n1 n2 -- n3)**	**n1 um n2 Bits verschieben (n2<0: nach rechts)**
2*	**(w1 -- w2)**	**w1 um ein Bit nach links schieben (Bit 0=0)**
2/	**(n1 -- n2)**	**n1 um ein Bit nach rechts schieben (Bit 15 bleibt)**
SHIFT	**(n1 n2 -- n3)**	**n1 logisch um n2 Bits verschieben (n2<0:nach rechts)**
U2/	**(u1 -- u2)**	**u1 logisch um ein Bit nach links schieben (Bit 15=0)**
FLIP	**($xxyy -- $yyxx)**	**High- und Low-Byte vertauschen**
0<	**(n -- flag)**	**Flag ist wahr, wenn n kleiner 0 ist**
0=	**(n -- flag)**	**Flag ist wahr, wenn n gleich 0 ist**
0<>	**(n -- flag)**	**Flag ist wahr, wenn n ungleich 0 ist**
0>	**(n -- flag)**	**Flag ist wahr, wenn n größer 0 ist**
<	**(n1 n2 -- flag)**	**Flag ist wahr, wenn n1 kleiner n2 ist**
=	**(n1 n2 --)**	**Flag ist wahr, wenn n1 gleich n2 ist**
<>	**(n1 n2 -- flag)**	**Flag ist wahr, wenn n1 ungleich n2 ist**
>	**(n1 n2 -- flag)**	**Flag ist wahr, wenn n1 größer n2 ist**
U<	**(u1 u2 -- flag)**	**Flag ist wahr, wenn u1 kleiner u2 ist**
U>	**(u1 u2 -- flag)**	**Flag ist wahr, wenn u1 kleiner u2 ist**
WITHIN	**(w1 w2 w3 -- flag)**	**Flag ist wahr, wenn w2<w1<w3 ist**
CASE?	**(n1 n2 -- n1 0 | -1)**	**Vergleich zweier Werte (liefert -1, wenn n1=n2)**
MIN	**(n1 n2 -- n3)**	**n3 ist der kleinere Wert**
MAX	**(n1 n2 -- n3)**	**n3 ist der größere Wert**
UMIN	**(u1 u2 -- u3)**	**u3 ist der kleinere Wert**
UMAX	**(u1 u2 -- u3)**	**u3 ist der größere Wert**
D+	**(d1 d2 -- d3)**	**Addiert d2 zu d1**
D-	**(d1 d2 -- d3)**	**Subtrahiert d2 von d1**
DNEGATE	**(d1 -- d2)**	**Subtrahiert d1 von 0**
?DNEGATE	**(d1 n -- d2)**	**d1 negieren, wenn n negativ**
DABS	**(d1 -- +d2)**	**Subtrahiert d1 von 0, wenn d1 kleiner 0 ist (Absolutwert)**
D2*	**(d1 -- d2)**	**Schieben von d1 um ein Bit nach links (Bit 0=0)**
D2/	**(d1 -- d2)**	**Schieben von d1 um ein Bit nach rechts (Bit15 bleibt)**
DU2/	**(du1 -- du2)**	**Schieben von du1 um ein Bit nach rechts (Bit15=0)**
D0<	**(d -- flag)**	**Flag ist wahr, wenn d kleiner 0 ist**
D0=	**(d -- flag)**	**Flag ist wahr, wenn d gleich 0 ist**
D0<>	**(d -- flag)**	**Flag ist wahr, wenn d ungleich 0 ist**
D0>	**(d -- flag)**	**Flag ist wahr, wenn d größer 0 ist**
D<	**(d1 d2 -- flag)**	**Flag ist wahr, wenn d1 kleiner d2 ist**
D=	**(d1 d2 -- flag)**	**Flag ist wahr, wenn d1 gleich d2 ist**
D<>	**(d1 d2 -- flag)**	**Flag ist wahr, wenn d1 ungleich d2 ist**
D>	**(d1 d2 -- flag)**	**Flag ist wahr, wenn d1 größer d2 ist**
DU<	**(du1 du2 -- flag)**	**Flag ist wahr, wenn du1 kleiner du2 ist**
DU>	**(du1 du2 -- flag)**	**Flag ist wahr, wenn du1 größer du2 ist**
DWITHIN	**(d1 d2 d3 -- flag)**	**Flag ist wahr, wenn d2<d1<d3 ist**
DMIN	**(d1 d2 -- d3)**	**d3 ist der kleinere Wert**
DMAX	**(d1 d2 -- d3)**	**d3 ist der größere Wert**

DUMIN	**(du1 du2 -- du3)**		**du3 ist der kleinere Wert**
DUMAX	**(du1 du2 -- du3)**		**du3 ist der größere Wert**
M+	**(d1 n -- d2)**		**Addiert n zu d1**
M-	**(d1 n -- d2)**		**Subtrahiert n von d1**
UM*	**(u1 u2 -- ud)**		**Multipliziert u1 mit u2 (32Bit-Ergebnis)**
M*	**(n1 n2 -- d)**		**Multipliziert n1 mit n2 (32Bit-Ergebnis)**
*****	**(n1 n2 -- n3)**		**Multipliziert n1 mit n2 (16Bit-Ergebnis)**
UM/MOD	**(ud u1 -- u2 u3)**		**Dividiert ud durch u1; liefert Quotient u3 und Rest u2**
M/MOD	**(d n1 -- n2 n3)**		**Dividiert d durch n1; liefert Rest n2 und Quotient n3**
M/	**(d1 n1 -- n2)**		**Dividiert d durch n1; liefert Quotient n2**
/MOD	**(n1 n2 -- n3 n4)**		**Dividiert n1 durch n2; liefert Rest n3 und Quotient n4**
/	**(n1 n2 -- n3)**		**Dividiert n1 durch n2; liefert Quotient n3**
MOD	**(n1 n2 -- n3)**		**Dividiert n1 durch n2; liefert Rest n3**
***/MOD**	**(n1 n2 n3 -- n4 n5)**		**n1*n2/n3 ergibt Rest n4 und Quotient n5**
***/**	**(n1 n2 n3 -- n4)**		**n1*n2/n3 ergibt Quotient n4 (32Bit-Zwischenergebnis)**

Speicherzugriffe

CHAR+	**(addr1 -- addr2)**		**Liefert nächste Zeichenadresse**
CHARS	**(n -- addr)**		**Liefert Anzahl der Adressen für n Zeichen**
CELL+	**(addr1 -- addr2)**		**Liefert nächste Zellenadresse**
CELLS	**(n -- addr)**		**Liefert Anzahl der Adressen für n Zellen**
>NEXTTASK	**(addr1 -- addr2)**		**Ermittelt nächste Taskadresse**
>VADDR	**(addr1 -- addr2)**		**Ermittelt Variablenadresse addr2 aus Offset addr1**
C@	**(addr -- byte)**		**Liefert Byte von angegebener Adresse**
C!	**(w addr --)**		**Speichert das niederwertige Byte von w ab addr**
@	**(addr -- w)**		**Ein bei addr gespeicherter Wert auslesen**
!	**(w addr --)**		**w ab addr speichern**
2@	**(addr -- w1 w2)**		**Wertepaar w2 aus addr und w1 aus nächste Zelle lesen**
2!	**(w1 w2 addr --)**		**Speichert w2 bei addr und w1 in nächste Zelle**
CMOVE	**(addr1 addr2 u --)**		**Kopiert u Bytes von addr1 nach addr2 (aufsteigend)**
CMOVE>	**(addr1 addr2 u --)**		**Wie CMOVE, jedoch in absteigender Reihenfolge**
MOVE	**(addr1 addr2 u --)**		**Wahlweise CMOVE oder CMOVE> (nicht überlappend)**
FILL	**(addr u byte --)**		**Füllt u Bytes ab addr mit Wert byte**
ERASE	**(addr u --)**		**Füllt u aufeinanderfolgende Bytes mit Null**
BLANK	**(addr u --)**		**Füllen von u Bytes ab addr**
+!	**(n addr --)**		**Addiert den Wert n zur Variable ab addr**
-!	**(n addr --)**		**Subtrahiert den Wert n von der Variable ab addr**
ON	**(addr --)**		**Variable ab Adresse addr wird mit -1 beschrieben**
OFF	**(addr --)**		**Variable ab Adresse addr wird mit 0 beschrieben**
SKIP	**(addr1 len1 char -- addr2 len2)**		**Im String wird das angegebene Zeichen char überlesen**
SCAN	**(addr1 len1 char -- addr2 len2)**		**Im String wird nach dem Zeichen char gesucht**
SKIP>	**(addr len1 char -- addr len2)**		**Die Zeichen char am Stringende werden entfernt**
SCAN>	**(addr len1 char -- addr len2)**		**Vom Stringende ab wird nach dem Zeichen char gesucht**
-TRAILING	**(addr len1 -- addr len2)**		**Leerzeichen am Stringende abschneiden**
/STRING	**(addr1 len1 len2 -- addr2 len2)**		**Die ersten len2 Zeichen am Stringanfang überspringen**
/$	**(csa/0 -- addr)**		**Liefert Adresse nach einem Inline-String (durch $, erzeugt)**
COUNT	**(csa -- addr len)**		**Anfangsadresse und Länge eines Counted-String bei csa**
COUNT>0	**(addr -- addr len)**		**Zählt die Zeichen im String bis zum 0-Ende**
>$	**(addr len -- csa/0)**		**String mit Längenangabe und 0-Ende zum PAD bringen**
UPC	**(char1 -- char2)**		**Wandelt ein Zeichen in Großschrift (auch Umlaute)**
UPPER	**(addr len --)**		**Wandelt alle Zeichen des Strings in Großbuchstaben um**
PC@	**(addr -- b)**	**X**	**Byte aus Port lesen**
PC!	**(b addr --)**	**X**	**Bytes in Port einschreiben**
P@	**(addr -- w)**		**Wort aus (benachbarte) Portadresse(n) lesen**
P!	**(w addr --)**		**Wort in (benachbarte) Portadresse(n) schreiben**

Zusatzbefehle bei externen Speicher

(MALLOC	**(len. -- ptr 0 | rlen error)**	**X**	**Reserviert len. Adressen und liefert Pointer auf Anfang**
(MRELOC	**(ptr len. -- error)**	**X**	**Verändert Länge des reservierten Speichers**
(MFREE	**(ptr -- error)**	**X**	**Gibt reservierten Speicher wieder frei**
PTR>D	**(ptr -- d)**		**Wandelt Pointers in eine 32Bit-Adresse**
D>PTR	**(d -- ptr)**		**Wandelt eine 32Bit-Adresse in einen Pointer**
CS@	**(-- ptr)**		**Liefert Anfang des Programmspeichers**
DS@	**(-- ptr)**		**Liefert Anfang des Datenspeichers**
SS@	**(-- ptr)**	**X**	**Liefert Anfang des Stackspeichers**
LC@	**(ptr -- b)**	**X**	**Byte aus externen Speicher holen**
LC!	**(b ptr --)**	**X**	**Byte in externen Speicher schreiben**
L@	**(ptr -- n)**	**X**	**Wort aus externen Speicher holen**
L!	**(n ptr --)**	**X**	**Wort in exterenen Speicher schreiben**
L2@	**(ptr -- d)**	**X**	**Doppelwort holen (höherwertiges Wort zuerst)**
L2!	**(d ptr --)**	**X**	**Doppelwort schreiben (höherwertiges Wort zuerst)**
LCMOVE	**(ptr1 ptr2 u --)**	**X**	**Kopiert u Bytes bytes von ptr1 nach ptr2 (aufsteigend)**
LCMOVE>	**(ptr1 ptr2 u --)**	**X**	**Wie LCMOVE, jedoch in absteigender Reihenfolge**
LMOVE	**(ptr1 ptr2 u --)**	**X**	**Wahlweise LCMOVE oder LCMOVE>**
LFILL	**(ptr u byte --)**	**X**	**Externer Speicher füllen**
LWFILL	**(ptr u w --)**	**X**	**Externer Speicher mit u Worte füllen (low zuerst)**

Ein-/Ausgabebefehle

STANDARD-IO	**(--)**		**Nutze gespeichertes INPUT-, OUTPUT- und DISC-Interface**
(INPUT	**(--)**	**UT**	**Das Standard-Eingabedevice verwenden**
(OUTPUT	**(--)**	**UT**	**Das Standard-Ausgabedevice verwenden**
(DISC	**(--)**	**UT**	**Das Standard-Fileinterface verwenden**
COMMAND!	**(com -- error)**	**X**	**Befehlsübertragung zum Terminal**
EMIT?	**(-- f)**	**UV**	**Liefert Flag, ob Zeichen ausgegeben werden kann**
EMIT	**(char --)**	**UV**	**Ausgabe des Zeichen char**
TYPE	**(addr u --)**	**UV**	**u Zeichen ab addr ausgeben**
CR	**(--)**	**UV**	**Zum Anfang der nächsten Zeile gehen**
DEL	**(--)**	**UV**	**Zeichen vor dem Cursor löschen**
BELL	**(--)**	**UV**	**Ton ausgeben**
CLS	**(--)**	**UV**	**Bildschirm löschen**
MAXAT	**(-- x y)**	**UV**	**Maxmimale Größe des Bildschirm liefern**
AT	**(x y --)**	**UV**	**Neue Cursorposition setzen**
AT?	**(-- x y)**	**UV**	**Aktuelle Cursorposition abfragen**
SPACE	**(--)**		**Ein Leerzeichen ausgeben**
SPACES	**(n --)**		**Falls n positiv ist: n Leerzeichen ausgeben**
INDENT	**(--)**		**Anzahl der Leerzeichen stehen in INDENT?**
-TYPE	**(addr u --)**		**ASCII-Werte unter $20 als Punkt anzeigen**
.ID	**(csa --)**		**Befehlsname ausgeben (maximal 31 Zeichen)**
KEY?	**(-- f)**	**UV**	**Test, ob Zeichen bereitsteht**
KEY	**(-- char)**	**UV**	**Empfängt ein Zeichen (wartet)**
STRING	**(-- addr u)**	**UV**	**u Zeichen ohne Test empfangen und ab addr ablegen**
EDITSTRING	**(addr maxlen pos len -- pos2 len2)**	**UV**	**String ausgeben und editieren (ändert SPAN)**
QUERY	**(--)**	**UV**	**Nächsten Befehlsstring holen**
EXPECT	**(addr +n --)**		**Erwartet Empfang von maximal n Zeichen (ändert SPAN)**
STOP?	**(-- f)**		**Wartet auf Taste; liefert bei #BRK (meist CTRL+X) -1**
BINARY	**(--)**		**Zahlenbasis 2 einstellen**
DECIMAL	**(--)**		**Zahlenbasis 10 einstellen**
HEX	**(--)**		**Zahlenbasis 16 einstellen**
>NUMBER	**(du1 addr1 len1 -- du2 addr2 len2)**		**String zu du1 akkumulieren (DPL+1 wenn ungleich -1)**
CONVERT	**(du1 addr1 -- ud2 addr2)**		**String ab addr1+1 zu ud1 akkumulieren**
NUMBER?	**(csa -- addr 0 | n -1 | d 0>)**	**DX**	**Test auf Zahlenstring (Prefix $,& oder % und Vorzeichen)**
<#	**(--)**		**Initialisiert Zahlenausgabe**
HOLD	**(char --)**		**Übernimmt ein Zeichen in den Zahlenstring**

#	(ud1 -- ud2)		Nächste Ziffer in den Zahlenstring übernehmen
#S	(ud1 -- ud2)		Bis ud2=0 mindestens noch eine Ziffer umwandeln
SIGN	(n --)		Fügt bei n<0 "-" in den Zahlenstring ein
#>	(wd -- addr u)		Abschluß einer Zahlenstring-Erzeugung
D.R	(d n --)		Rechtsbündige Ausgabe von d im Feld mit n Zeichen
D.	(d --)		Ausgabe von d im freien Format mit Leerzeichen
U.R	(u n --)		u rechtsbündig in einem Feld mit n Zeichen ausgeben
0U.R	(u n --)		Wie U.R, aber Rest mit 0 füllen
U.	(u --)		Ausgabe von u mit nachfolgenden Leerzeichen
.R	(n1 n2 --)		Rechtsbündige Ausgabe von n1 im Feld mit n2 Zeichen
.	(n --)		Ausgabe von n mit nachfolgenden Leerzeichen
?	(addr --)		Ausgabe des Inhalts einer Variablen
U?	(addr --)		Vorzeichenlose Ausgabe des Inhalts einer Variablen
D?	(addr --)		Ausgabe des Inhalts einer 32Bit-Variablen
.STATE	(--)	DX	" OK" oder "]" ausgeben (Statusmeldung von QUIT)
.BLK	(--)	DX	">" ausgeben (beim Laden von Screens)
.S	(--)		Kopiert die obersten Stackwerte und gibt sie aus
DUMP	(addr u --)		Anzeige von u Adressen ab addr
LDUMP	(ptr u --)	X	Anzeige beliebiger Speicheradressen
ORDER	(--)		Zeigt die aktuelle Suchreihenfolge an
VOCS	(--)		Anzeige aller Vokabulare
WORDS	(--)		Befehlsliste ausgeben
FILES	(--)		Alle gespeicherten Filenamen ausgeben
FILE.	(--)		Aktueller Filename ausgeben
LIST	(+n --)		Screen n anzeigen
IDENT	(--)		Einschaltmeldung ausgeben
-IDENT	(--)		Endemeldung ausgeben

Kontrollstrukturen

EXECUTE	(cfa --)		Führt Befehl mit angegebener CFA aus
PERFORM	(addr --)		Führt Befehl mit der in addr kompilierten CFA aus
NUMBER	(n -1 | d 0> --)	DX	Interpretiert NUMBER?-Flags (läßt Werte auf dem Stack)
EXIT	(-- ; R: sys --)	R	Aussprung aus einem FORTH-Befehl
?EXIT	(flag --)	R	Aussprung, wenn Flag ungleich Null
0=EXIT	(flag --)	R	Aussprung, wenn Flag gleich Null
BRANCH	(--)	R	Bedingungsloser Sprung (Offset als Inline-Wert)
?BRANCH	(flag --)	R	Sprung wenn flag=0 (Offset als Inline-Wert)
NEXTBRANCH	(-- ; R: n -- n-1 |)	R	Sprung und n-1, wenn Wert <> 0 (Offset als Inline-Wert)
>MARK	(-- addr)		Vorwärtssprung vorbereiten
>RESOLVE	(addr --)		Vorwartssprung korrigieren
<MARK	(-- addr)		Rückwärtssprung vorbereiten
<RESOLVE	(addr --)		Rückwärtssprung korrigieren
RECURSE	(--)	R	Aktuellen Befehl rekursiv aufrufen
IF	(flag --) (C: -- addr 1)	I,R	Anwendung: IF ... [ELSE ...] THEN
ELSE	(--) (C: addr1 1 -- addr2 -1)	I,R	Anwendung: IF...ELSE...THEN
THEN	(--) (C: addr |1| --)	I,R	Ende einer IF...THEN oder IF...ELSE...THEN-Struktur
BEGIN	(--) (C: -- 2 addr 2)	I,R	Anfang einer BEGIN...UNTIL oder BEGIN...WHILE...REPEAT-Schleife
WHILE	(flag --) (C: addr1 2 -- addr2 -2 addr1 2)	I,R	Bei flag=0 die BEGIN...UNTIL / BEGIN...REPEAT-Schleife verlassen
UNTIL	(flag --) (C: 2 ... addr 2 --)	I,R	Ende einer BEGIN...UNTIL-Struktur
REPEAT	(--) (C: 2 ... addr 2 --)	I,R	Ende einer BEGIN...REPEAT-Definition
?DO	(n1 n2 --) (C: -- addr 3)	I,R	Anwendung: ?DO...LOOP oder ?DO...+LOOP
DO	(n1 n2 --) (C: -- addr 3)	I,R	Anwendung: DO...LOOP oder DO...+LOOP
LOOP	(--) (C: addr 3 --)	I,R	Ende ...LOOP-Schleife; addiert 1 zum Schleifenindex
+LOOP	(n --) (C: addr 3 --)	I,R	Ende ...+LOOP-Schleife; addiert n zum Schleifenindex
BOUNDS	(n1 len -- n2 n3)		Schleifenparameter aus Anfangswert und Länge
I	(-- n)		Liefert aktuellen Wert der innersten Schleife
I'	(-- n)		Liefert Endwert der innersten Schleife
J	(-- n)		Liefert aktuellen Wert der nächst äußeren Schleife
J'	(-- n)		Liefert Endwert der nächst äußeren Schleife

LEAVE	**(--) (C: --)**	**R**	**Verläßt sofort die innerste Schleife**
?LEAVE	**(f --)**	**R**	**Verläßt Schleife, wenn Flag ungleich Null**
UNLOOP	**(--)**	**R**	**Entfernt LOOP-Parameter**
FOR	**(u -- ; R: -- u) (C: -- addr 4)**	**I,R**	**Anwendung: FOR...NEXT (wiederholt Befehle n+1 mal)**
NEXT	**(-- ; R: u -- u-1 |) (C: addr 4 --)**	**I,R**	**Ende einer FOR...NEXT-Schleife (Ende, wenn u=0)**
CASE	**(--) (C: -- 5 5)**	**I,R**	**Anfang einer CASE-Struktur**
OF	**(n1 n2 -- n1 |) (C: 5 -- addr -5)**	**I,R**	**CASE-Struktur: Vergleich und Sprung wenn ungleich**
ENDOF	**(--) (C: -5 -- addr -2 5)**	**I,R**	**CASE-Struktur: Sprung zu ENDCASE**
ENDCASE	**(n --) (C: 5 ... 5 --)**	**I,R**	**Ende einer CASE-Struktur**

Definition und Dictionary

ALIGNED	**(addr1 -- addr2)**		**Aligned Adresse, wenn notwendig**
ALLOT	**(n --)**		**Reserviert n Adressen im Dictionary**
ALIGN	**(--)**		**Aligned Dictionaryadresse, wenn notwendig**
C,	**(char --)**		**Speichert Byte im Dictionary**
,	**(w --)**		**Speichert Wort im Dictionary**
,C	**(cfa --)**		**Speichert HERE in CODE? und kompiliert CFA**
,A	**(w --)**		**Speichert HERE in CODE? und speichert w**
LITERAL	**(n --)**	**I,R**	**Kompiliert Wert als Inline-Literal**
2LITERAL	**(d --)**	**I,R**	**Kompiliert 32Bit-Wert als Inline-Literal**
NUMBER,	**(n -1 | d 0> --)**	**DX**	**Kompiliert 16Bit- oder 32Bit-Wert als Inline-Literal**
COMPILE	**(--)**		**Kompiliert nachfolgenden Inline-Befehl**
[COMPILE]	**(--)**	**I,R**	**Kompiliert Codefeldadresse des nächsten Befehls**
POSTPONE	**(--)**	**I,R**	**Verwendet COMPILE oder [COMPILE] (wenn Immediate)**
-$,	**(char --)**		**Speichert nachfolgenden String ohne SKIP im Dictionary**
$,	**(char --)**		**Speichert nachfolgenden String mit SKIP im Dictionary**
>$,	**(addr len --)**		**Speichert angegebenen String im Dictionary**
"	**(-- csa) (C: ccc" ; --)**	**I**	**Liefert Stringadresse**
."	**(--) (C: ccc" ; --)**	**I,R**	**String kompilieren und bei Aufruf ausgeben**
VALLOT	**(n --)**		**Reserviert n Adressen im Variablenbereich**
VALIGN	**(--)**		**Aligned Variablenadresse**
VC,	**(char --)**		**Speichert Byte im Variablenbereich**
V,	**(w --)**		**Speichert Wort im Variablenbereich**
HALLOT	**(n --)**		**Reserviert n Adressen im Heap (verschiebt Variablen)**
HALIGN	**(--)**		**Aligned Heapadresse**
HC,	**(char --)**		**Speichert Byte im Heap**
H,	**(w --)**		**Speichert Wort im Heap**
EVALUATE	**(addr len --)**		**Angegebenen String interpretieren/kompilieren**
INTERPRET	**(--)**		**Aktuelle Befehlszeile interpretieren/kompilieren**
PARSER	**(csa -- ???)**	**D**	**Befehlswort interpretieren/kompilieren**
[	**(--)**	**I**	**Setzt Interpreter-Modus (STATE und PARSER)**
]	**(--)**		**Setzt Kompiler-Modus (STATE und PARSER)**
DEFINITIONS	**(--)**		**Das erste Suchvokabular wird zum Kompiler-Vokabular**
ALSO	**(--)**		**Übernimmt aktuelles Vokabular in den festen Teil**
SEAL	**(--)**		**Entfernt oberstes Suchvokabular**
ONLYFORTH	**(--)**		**FORTH wird zum Such- und Kompilier-Vokabular**
FORTH	**(--)**		**Ersetzte das erste Suchvokabular durch das FORTH**
-WORD	**(char -- csa)**		**Nachfolgender String wird zu WDP@ kopiert (mit 0-Ende)**
WORD	**(char-- csa)**		**Nachfolgender String nach SKIP mit 0-Ende zu WDP@**
FIND	**(csa -- csa 0 | cfa n)**	**DX**	**Sucht Befehl, liefert CFA (|2|=Restrict;0< bei Immediate)**
(FIND	**(-1 lfan ... lfa1 cfa -- cfa 0 | lfa -1)**		**Sucht den Befehl in den angegebenen Befehlslisten**
'	**(name ; -- cfa)**		**Liefert Codefeldadresse des Befehls (Interpretermodus)**
[']	**(-- w) (c: name ; --)**	**I,R**	**Liefert Codefeldadresse des Befehls (Kompilermodus)**
L>NAME	**(lfa -- nfa)**		**Ermittelt Namensfeldadresse aus Linkfeldadresse**
N>LINK	**(nfa -- lfa)**		**Ermittelt Linkfeldadresse aus der Namensfeldadresse**
LINK>	**(lfa -- cfa)**		**Ermittelt Codefeldadresse aus Linkfeldadresse**
BODY>	**(pfa -- cfa)**		**Ermittelt Codefeldadresse aus Parameterfeldadresse**
NAME>	**(nfa -- cfa)**		**Ermittelt Codefeldadresse aus Namensfeldadresse**
>LINK	**(cfa -- lfa | 0)**		**Ermittelt Linkfeldadresse aus Codefeldadresse**
>BODY	**(cfa -- pfa)**		**Ermittelt Parameterfeldadresse aus Codefeldadresse**
>NAME	**(cfa -- nfa | 0)**		**Ermittelt Namensfeldadresse aus Codefeldadresse**

FORGET	**(name ; --)**		**Löschen eines Befehls und alle nachfolgenden Wörter**
(FORGET	**(addr --)**		**Alle Befehle ab addr entfernen (auch geschützter Bereich)**
REMOVE	**(dp heap --)**		**Alle Befehle zwischen dp und heap entfernen**
'REMOVE	**(dp heap -- dp heap)**	**D**	**Entfernen eigener Befehlsstrukturen**
HCLEAR	**(--)**		**Heap löschen**
SAVE	**(--)**		**Heap löschen, Systemvariablen in (SYSVAR sichern (außer SFLAG und UFLAG)**
EMPTY	**(--)**		**Zurückholen des SAVE-Zustandes (löscht neue Befehle)**
(	**(ccc) ; --)**	**I**	**Kommentar überlesen**
.(	**(ccc) ; --)**	**I**	**Ein mit) abgeschlossenen String unmittelbar ausgeben**
****	**(--)**	**I**	**Gesamte Zeile ist Kommentar**
****	**(--)**	**I**	**Rest des Screens ist Kommentar**
\NEEDS	**(name ; --)**	**I**	**Zeile ist Kommentar, wenn Befehl vorhanden ist**
ASCII	**(ccc ; -- char)**		**Liefert Wert des ersten Zeichens im nachfolgenden String**
IMMEDIATE	**(--)**		**Markiert letzten Befehl als Immediate**
RESTRICT	**(--)**		**Markiert letztes Befehl als Restrict**
INDIRECT	**(--)**		**Markiert letzten Befehl als Indirect**
HIDE	**(--)**		**Neu definierten Befehl verstecken**
REVEAL	**(--)**		**Zuletzt definierten Befehl wieder sichtbar machen**
-HEADERS	**(--)**		**Alle nachfolgenden Befehlsheader zum Heap bringen**
|	**(--)**		**Nur nächsten Befehlsheader zum Heap bringen**
HEADERS	**(--)**		**Alle nachfolgenden Befehlsheader zum Dictionary**
:	**(-- ; R: -- sys) (C: name ; -- 0)**		**Definition eines FORTH-Befehls (0 für Kontrollzwecke)**
;	**(-- ; R: sys --) (C: 0 --)**	**I**	**Abschluß einer FORTH-Definition**
CREATE	**(name ; --)**	**DX**	**Erzeugt ein Befehlsname ohne reservierten Speicher**
DOES>	**(-- addr)**	**I,R**	**Leitet den Ausführungsteil einer CREATE-Definition ein**
VCREATE	**(name ; --)**		**Erzeugt eine Variable ohne reservierten Speicher**
VDOES>	**(-- vaddr)**	**I,R**	**Leitet den Ausführungsteil einer VCREATE-Definition ein**
VARIABLE	**(name ; --)**		**Definition einer Variable**
2VARIABLE	**(name ; --)**		**Definition einer 32Bit-Variablen**
CONSTANT	**(name ; w --)**		**Definition einer Konstante**
2CONSTANT	**(name ; d --)**		**Definition einer 32Bit-Konstante**
VOCABULARY	**(name ; --)**		**Definition eines Vokabulars**
USER	**(name ; --)**		**Definition einer USER-Variable**
ALIAS	**(name ; cfa --)**		**Definition eines ALIAS-Befehls**
>LABEL	**(name ; addr --)**	**I**	**Definition einer Konstante (vollständig auf dem Heap)**
LABEL	**(name ; --)**	**I**	**Aktuelle Dictionary-Adresse mit >LABEL merken**
DEFER	**(name ; --)**		**Definition eines DEFER-Befehls**
IS	**(name ; cfa --)**		**Umleitung eines DEFER-Befehls auf die angegebene CFA**
UVECTOR	**(name ; # user --)**		**Definition eines UVECTOR-Befehls**
UTABLE:	**(name ; len user --)**		**Definition einer neuen Tabelle für UVECTOR-Befehle**
(;CODE	**(-- ; R: sys --)**	**R**	**Letzter Befehl hat folgenden (Assembler-)Ausführungsteil**

Fileverwaltung

MAKE	**(name ; --)**		**Erzeugt neues File mit Länge 0 und öffnet es mit OPEN**
DELETE	**(name ; --)**		**Löscht angegebenes File**
OPEN	**(name ; --)**		**Öffnet File (legt Filename auch als Befehl an**
(OPEN	**(csa/0 --)**		**Öffnet angegebenes File ohne neuen Befehlseintrag**
CLOSE	**(--)**		**Schließt aktuelles File**
MORE	**(u --)**		**File um u Blöcke (je 1024 Byte) vergrößern**
CAPACITY	**(-- u)**		**Filegröße abfragen**
BLOCK	**(u -- addr)**		**Liefert die Adresse des ersten Bytes im Block u**
BUFFER	**(u -- addr)**		**Wie BLOCK, jedoch ist der Inhalt undefiniert**
UPDATE	**(--)**		**Markiert zuletzt angeforderten Block als verändert**
SAVE-BUFFERS	**(--)**		**Alle mit UPDATE markierten Blöcke sichern**
EMPTY-BUFFERS		**(--)**	**Gibt alle Diskblöcke ohne Sicherung frei**
FLUSH	**(--)**		**Führt SAVE-BUFFERS aus und gibt alle Diskpuffer frei**
LOAD	**(u --)**		**Laden Block u (0 nicht erlaubt)**
THRU	**(u1 u2)**		**Ladet Screens u1 bis einschließlich u2**
-->	**(--)**		**Ladet nächsten Screen (BLK erhöhen und >IN auf 0)**
INCLUDE	**(name ; --)**		**Screen 1 vom angegebenes File laden**
LOADFROM	**(name ; u --)**		**Screen u vom angegebenen File laden**

SAVESYSTEM	**(name; --)**		**Speicherung des aktuellen FORTH-Systems auf Diskette**
(FILE?	**(string/0 -- error)**	**UV**	**Test, ob File vorhanden ist**
(FILE-CREATE	**(string/0 -- id 0 | error)**	**UV**	**File neu anlegen**
(FILE-DELETE	**(string/0 -- error)**	**UV**	**File löschen**
(FILE-OPEN	**(string/0 attr -- id 0 | error)**	**UV**	**File öffnen (Attribute #RO, #WO oder #RW)**
(FILE-CLOSE	**(id -- error)**	**UV**	**File löschen**
(FILE-SIZE	**(id -- d 0 | error)**	**UV**	**Filegröße ermitteln**
(FILE-POS!	**(d dir id -- error)**	**UV**	**Schreib-/Lesezeiger auf d setzen (dir=0:ab Anfang)**
(FILE-POS@	**(id -- d 0 | error)**	**UV**	**Position des Schreib-/Lesezeigers abfragen**
(FILE-READ	**(ptr len id -- error)**	**UV**	**len Bytes lesen und ab ptr ablegen**
(FILE-WRITE	**(ptr len id -- error)**	**UV**	**len Bytes ab ptr zum File schreiben**
(FILE-FREE	**(dev -- free. max. 0 | error)**	**UV**	**Freie und maximale Diskettengröße abfragen**
(FILE-FIRST	**(string/0 attr -- addr 0 | error)**	**UV**	**Sucht ersten File und liefert Zeiger auf Information**
(FILE-NEXT	**(-- addr 0 | error)**	**UV**	**Sucht nächstes File und liefer Zeiger auf Informationen**

Multitasker

PAUSE	**(--)**	**D**	**Aufruf des Multitaskers**
SINGLETASK	**(--)**		**PAUSE hat keine Wirkung**
MULTITASK	**(--)**		**Bei PAUSE wird der Multitasker aufgerufen**

Fehlerbehandlung

ERROR	**(error | cfa $7fff | cfa $ffff --)**		**Fehlerbehandlung bei error ungleich 0**
?ERROR	**(flag error | flag cfa $7fff | flag cfa $ffff --)**		**Fehlerbehandlung bei Flag ungleich 0**
?STACK	**(--)**		**Fehlerbehandlung wenn Datenstack außerhalb Grenzen**
?DEPTH	**(n --)**		**Fehlerbehandlung bei weniger als n Werten**
?ALLOT	**(n --)**		**Fehlerbehandlung wenn weniger als n Bytes frei**
?PAIRS	**(n1 n2 --)**		**Fehlerbehandlung wenn n1 ungleich n2 ist**
?OPEN	**(--)**		**Fehlerbehandlung wenn kein File geöffnet wurde**
?BLK	**(n --)**		**Fehlerbehandlung bei nicht verfügbarer Blocknummer**
?NAME	**(-- cfa)**		**Fehlerbehandlung wenn kein Befehlsname mehr folgt**
NOTFOUND	**(--)**		**Fehlerbehandlung bei nicht gefundenen Befehlen**
NODEFER	**(--)**		**Fehlerbehandlung bei nicht initialisierten DEFER**
ABORT"	**(flag --) (C: string" ; --)**	**I,R**	**Bei flag<>0: Stack löschen und ERROR ausgeführen**
ERROR"	**(flag --) (C: string" ; --)**	**I,R**	**Bei flag<>0: ERROR mit angegebenen String aufgerufen**
>ERRORTEXT	**(-- addr)**	**V**	**Enthält Zeiger auf aktuelle Fehlertabelle**
ERRORTEXT@	**(error | cfa $7fff -- addr len)**	**DX**	**Holt Text zur Fehlernummer**
(ERRORHANDLER	**(error --)**		**Standard-Fehlerbehandlung: 'ERROR und dann QUIT**
'ERROR	**(--)**	**D**	**Wird von (ERRORHANDLER vor QUIT aufgerufen**

Sonstiges

BOOT	**(??? --)**		**Befehl enthält BOOT-Routine, führt aber COLD aus**
'BOOT	**(--)**	**D**	**DEFER-Wort zum Einbau eigener BOOT-Routinen**
COLD	**(??? --)**		**Restart ab EMPTY; löscht beide Stacks, schießt File**
'COLD	**(--)**	**D**	**DEFER-Wort zum Einbau eigener Warmstart-Routinen**
ABORT	**(??? --)**		**Stack löschen und nach 'ABORT QUIT aufrufen**
'ABORT	**(--)**	**D**	**Defer für eigenes Autostart-Programm (Default: IDENT)**
QUIT	**(--)**		**Interpreterschleife (endlos)**
NOOP	**(--)**		**Leerbefehl für nicht aktive DEFER's**
BYE	**(--)**		**Verlassen des KK-FORTH nach 'BYE (liefert UFLAG als Fehlerflag ans Betriebssystem)**
'BYE	**(--)**	**D**	**DEFER-Wort für eigene Abschlußbefehle (Default: -IDENT)**

A.3 Alphabetisch sortierte Befehlsliste

-	(n1 n2 -- n3)		Subtrahiert n2 von n1
'	(name ; -- cfa)		Liefert Codefeldadresse des Befehls (Interpretermodus)
-!	(n addr --)		Subtrahiert den Wert n von der Variable ab addr
!	(w addr --)		w ab addr speichern
"	(-- csa) (C: ccc" ; --)	I	Liefert Stringadresse
#	(ud1 -- ud2)		Nächste Ziffer in den Zahlenstring übernehmen
#>	(wd -- addr u)		Abschluß einer Zahlenstring-Erzeugung
#B/BLK	(-- $400)		Anzahl der Bytes pro Block (Screen)
#BL	(-- $20)		Liefert ASCII-Wert des Leerzeichens
#BRK	(-- $18)		Liefert ASCII-Wert der Abbruchtaste (meist CTRL+X)
#C/L	(-- $40)		Anzahl der Zeichen pro Zeile
#CR	(-- $0d)		Liefert ASCII-Wert der RETURN-Taste
#DELIN	(-- sys)		Liefert ASCII-Wert der DEL-Taste
#DELOUT	(-- sys)		ASCII-Wert für Cursor-Links bei Ausgabe
#ESC	(-- $1b)		Liefert ASCII-Wert der ESC-Taste
#L/BLK	(-- $10)		Anzahl der Zeilen pro Block
#RO	(-- sys)		Flag für (FILE-OPEN : File nur lesen
#RW	(-- sys)		Flag für (FILE-OPEN : File lesen und schreiben
#S	(ud1 -- ud2)		Bis ud2=0 mindestens noch eine Ziffer umwandeln
#TIB	(-- addr)	U	Enthält Anzahl der Zeichen im Eingabepuffer
#TIB-MAX	(-- sys)		Größe des Eingabepuffers (meist 79 Zeichen)
#WO	(-- sys)		Flag für (FILE-OPEN : File nur schreiben
$,	(char --)		Speichert nachfolgenden String mit SKIP im Dictionary
-$,	(char --)		Speichert nachfolgenden String ohne SKIP im Dictionary
(	(ccc) ; --)	I	Kommentar überlesen
(;CODE	(-- ; R: sys --)	R	Letzter Befehl hat folgenden (Assembler-)Ausführungsteil
(DISC	(--)	UT	Das Standard-Fileinterface verwenden
(ERRORHANDLER	(error --)		Standard-Fehlerbehandlung: 'ERROR und dann QUIT
(FILE?	(string/0 -- error)	UV	Test, ob File vorhanden ist
(FILE-CLOSE	(id -- error)	UV	File löschen
(FILE-CREATE	(string/0 -- id 0 | error)	UV	File neu anlegen
(FILE-DELETE	(string/0 -- error)	UV	File löschen
(FILE-FIRST	(string/0 attr -- addr 0 | error)	UV	Sucht ersten File und liefert Zeiger auf Information
(FILE-FREE	(dev -- free. max. 0 | error)	UV	Freie und maximale Diskettengröße abfragen
(FILE-NEXT	(-- addr 0 | error)	UV	Sucht nächstes File und liefer Zeiger auf Informationen
(FILE-OPEN	(string/0 attr -- id 0 | error)	UV	File öffnen (Attribute #RO, #WO oder #RW)
(FILE-POS!	(d dir id -- error)	UV	Schreib-/Lesezeiger auf d setzen (dir=0:ab Anfang)
(FILE-POS@	(id -- d 0 | error)	UV	Position des Schreib-/Lesezeigers abfragen
(FILE-READ	(ptr len id -- error)	UV	len Bytes lesen und ab ptr ablegen
(FILE-SIZE	(id -- d 0 | error)	UV	Filegröße ermitteln
(FILE-WRITE	(ptr len id -- error)	UV	len Bytes ab ptr zum File schreiben
(FIND	(-1 lfan ... lfa1 cfa -- cfa 0 | lfa -1)		Sucht den Befehl in den angegebenen Befehlslisten
(FORGET	(addr --)		Alle Befehle ab addr entfernen (auch geschützter Bereich)
(INPUT	(--)	UT	Das Standard-Eingabedevice verwenden
(MALLOC	(len. -- ptr 0 | rlen error)	X	Reserviert len. Adressen und liefert Pointer auf Anfang
(MFREE	(ptr -- error)	X	Gibt reservierten Speicher wieder frei
(MRELOC	(ptr len. -- error)	X	Verändert Länge des reservierten Speichers
(OPEN	(csa/0 --)		Öffnet angegebenes File ohne neuen Befehlseintrag
(OUTPUT	(--)	UT	Das Standard-Ausgabedevice verwenden
(SYSVAR	(-- sys)		Zieladresse für die Kopie der Systemvariablen (SAVE)
*	(n1 n2 -- n3)		Multipliziert n1 mit n2 (16Bit-Ergebnis)
*/	(n1 n2 n3 -- n4)		n1*n2/n3 ergibt Quotient n4 (32Bit-Zwischenergebnis)
*/MOD	(n1 n2 n3 -- n4 n5)		n1*n2/n3 ergibt Rest n4 und Quotient n5
,	(w --)		Speichert Wort im Dictionary
,A	(w --)		Speichert HERE in CODE? und speichert w
,C	(cfa --)		Speichert HERE in CODE? und kompiliert CFA
.	(n --)		Ausgabe von n mit nachfolgenden Leerzeichen
."	(--) (C: ccc" ; --)	I,R	String kompilieren und bei Aufruf ausgeben
.(	(ccc) ; --)	I	Ein mit) abgeschlossenen String unmittelbar ausgeben
.BLK	(--)	DX	">" ausgeben (beim Laden von Screens)
.ID	(csa --)		Befehlsname ausgeben (maximal 31 Zeichen)
.R	(n1 n2 --)		Rechtsbündige Ausgabe von n1 im Feld mit n2 Zeichen
.S	(--)		Kopiert die obersten Stackwerte und gibt sie aus
.STATE	(--)	DX	" OK" oder "]" ausgeben (Statusmeldung von QUIT)
/	(n1 n2 -- n3)		Dividiert n1 durch n2; liefert Quotient n3
/$	(csa/0 -- addr)		Adresse nach einem Inline-String (durch $, erzeugt)
/MOD	(n1 n2 -- n3 n4)		Dividiert n1 durch n2; liefert Rest n3 und Quotient n4
/STRING	(addr1 len1 len2 -- addr2 len2)		Die ersten len2 Zeichen am Stringanfang überspringen
:	(-- ; R: -- sys) (C: name ; -- 0)		Definition eines FORTH-Befehls (0 für Kontrollzwecke)
;	(-- ; R: sys --) (C: 0 --)	I	Abschluß einer FORTH-Definition

?	**(addr --)**		**Ausgabe des Inhalts einer Variablen**
?ALLOT	**(n --)**		**Fehlerbehandlung wenn weniger als n Bytes frei**
?BLK	**(n --)**		**Fehlerbehandlung bei nicht verfügbarer Blocknummer**
?BRANCH	**(flag --)**	**R**	**Sprung wenn flag=0 (Offset als Inline-Wert)**
?DEPTH	**(n --)**		**Fehlerbehandlung bei weniger als n Werten**
?DNEGATE	**(d1 n -- d2)**		**d1 negieren, wenn n negativ**
?DO	**(n1 n2 --) (C: -- addr 3)**	**I,R**	**Anwendung: ?DO...LOOP oder ?DO...+LOOP**
?DUP	**(n -- n n) oder (0 -- 0)**		**Dupliziert n nur, wenn der Wert ungleich 0 ist**
?ERROR	**(flag error | flag cfa $7fff | flag cfa $ffff --)**		**Fehlerbehandlung bei Flag ungleich 0**
?EXIT	**(flag --)**	**R**	**Aussprung, wenn Flag ungleich Null**
?LEAVE	**(f --)**	**R**	**Verläßt Schleife, wenn Flag ungleich Null**
?NAME	**(-- cfa)**		**Fehlerbehandlung wenn kein Befehlsname mehr folgt**
?NEGATE	**(n1 n2 -- n1 | -n1)**		**Negiert n1, wenn n2 negativ ist**
?OPEN	**(--)**		**Fehlerbehandlung wenn kein File geöffnet wurde**
?PAIRS	**(n1 n2 --)**		**Fehlerbehandlung wenn n1 ungleich n2 ist**
?STACK	**(--)**		**Fehlerbehandlung wenn Datenstack außerhalb Grenzen**
@	**(addr -- w)**		**Ein bei addr gespeicherter Wert auslesen**
[	**(--)**	**I**	**Setzt Interpreter-Modus (STATE und PARSER)**
[']	**(-- w) (c: name ; --)**	**I,R**	**Liefert Codefeldadresse des Befehls (Kompilermodus)**
[COMPILE]	**(--)**	**I,R**	**Kompiliert Codefeldadresse des nächsten Befehls**
****	**(--)**	**I**	**Gesamte Zeile ist Kommentar**
****	**(--)**	**I**	**Rest des Screens ist Kommentar**
\NEEDS	**(name ; --)**	**I**	**Zeile ist Kommentar, wenn Befehl vorhanden ist**
]	**(--)**		**Setzt Kompiler-Modus (STATE und PARSER)**
|	**(--)**		**Nur nächsten Befehlsheader zum Heap bringen**
+	**(n1 n2 -- n3)**		**Addiert n2 zum Wert n1**
+!	**(n addr --)**		**Addiert den Wert n zur Variable ab addr**
+LOOP	**(n --) (C: addr 3 --)**	**I,R**	**Ende ...+LOOP-Schleife; addiert n zum Schleifenindex**
<	**(n1 n2 -- flag)**		**Flag ist wahr, wenn n1 kleiner n2 ist**
<#	**(--)**		**Initialisiert Zahlenausgabe**
<>	**(n1 n2 -- flag)**		**Flag ist wahr, wenn n1 ungleich n2 ist**
<MARK	**(-- addr)**		**Rückwärtssprung vorbereiten**
<RESOLVE	**(addr --)**		**Rückwärtssprung korrigieren**
=	**(n1 n2 --)**		**Flag ist wahr, wenn n1 gleich n2 ist**
-->	**(--)**		**Ladet nächsten Screen (BLK erhöhen und >IN auf 0)**
>	**(n1 n2 -- flag)**		**Flag ist wahr, wenn n1 größer n2 ist**
>$	**(addr len -- csa/0)**		**String mit Längenangabe und 0-Ende zum PAD bringen**
>$,	**(addr len --)**		**Speichert angegebenen String im Dictionary**
>BODY	**(cfa -- pfa)**		**Ermittelt Parameterfeldadresse aus Codefeldadresse**
>ERRORTEXT	**(-- addr)**	**V**	**Enthält Zeiger auf aktuelle Fehlertabelle**
>HEAP?	**(-- addr)**	**V**	**Enthält Anzahl headerlos zu definierenden Befehle**
>IN	**(-- addr)**	**U**	**Enthält den Offset in den interpretierenden Puffers**
>LABEL	**(name ; addr --)**	**I**	**Definition einer Konstante (vollständig auf dem Heap)**
>LINK	**(cfa -- lfa | 0)**		**Ermittelt Linkfeldadresse aus Codefeldadresse**
>MARK	**(-- addr)**		**Vorwärtssprung vorbereiten**
>NAME	**(cfa -- nfa | 0)**		**Ermittelt Namensfeldadresse aus Codefeldadresse**
>NEXTTASK	**(addr1 -- addr2)**		**Ermittelt nächste Taskadresse**
>NUMBER	**(du1 addr1 len1 -- du2 addr2 len2)**		**String zu du1 akkumulieren (DPL+1 wenn ungleich -1)**
>R	**(w -- ; R: -- w)**	**R**	**Wert zum Returnstack bringen**
>RESOLVE	**(addr --)**		**Vorwartssprung korrigieren**
>TIB	**(-- addr)**	**U**	**Enthält Zeiger auf aktuellen Eingabepuffer**
>VADDR	**(addr1 -- addr2)**		**Ermittelt Variablenadresse addr2 aus Offset addr1**
0<	**(n -- flag)**		**Flag ist wahr, wenn n kleiner 0 ist**
0<>	**(n -- flag)**		**Flag ist wahr, wenn n ungleich 0 ist**
0=	**(n -- flag)**		**Flag ist wahr, wenn n gleich 0 ist**
0=EXIT	**(flag --)**	**R**	**Aussprung, wenn Flag gleich Null**
0>	**(n -- flag)**		**Flag ist wahr, wenn n größer 0 ist**
0U.R	**(u n --)**		**Wie U.R, aber Rest mit 0 füllen**
1-	**(n -- n-1)**		**n um 1 erniedrigen**
1+	**(n -- n+1)**		**n um 1 erhöhen**
2-	**(n -- n-2)**		**n um 2 erniedrigen**
2!	**(w1 w2 addr --)**		**Speichert w2 bei addr und w1 in nächste Zelle**
2*	**(w1 -- w2)**		**w1 um ein Bit nach links schieben (Bit 0=0)**
2/	**(n1 -- n2)**		**n1 um ein Bit nach rechts schieben (Bit 15 bleibt)**
2@	**(addr -- w1 w2)**		**Wertepaar w2 aus addr und w1 aus nächste Zelle lesen**
2+	**(n -- n+2)**		**n um 2 erhöhen**
2>R	**(w1 w2 -- ; R: -- w1 w2)**	**R**	**Oberstes Wertepaar zum Returnstack bringen**
2CONSTANT	**(name ; d --)**		**Definition einer 32Bit-Konstante**
2DROP	**(dw --)**		**Oberstes Wertepaar entfernen**
2DUP	**(dw1 -- dw1 dw1)**		**Duplizieren des obersten Wertepaares**
2LITERAL	**(d --)**	**I,R**	**Kompiliert 32Bit-Wert als Inline-Literal**
2NIP	**(dw1 dw2 -- dw2)**		**Zweites Wertepaar entfernen**
2OVER	**(dw1 dw2 -- dw1 dw2 dw1)**		**Duplizieren des zweiten Stackpaares**
2R@	**(-- w1 w2 ; R: w1 w2 -- w1 w2)**	**R**	**Oberstes Returnstackpaar zum Datenstack kopieren**

2R>	**(-- w1 w2 ; R: w1 w2 --)**	**R**	**Oberstes Returnstackpaar zum Datenstack bringen**
2RDROP	**(-- ; R: w1 w2 --)**	**R**	**Oberstes Returnstackpaar entfernen**
2ROT	**(dw1 dw2 dw3 -- dw2 dw3 dw1)**		**Rotieren der obersten drei Wertepaare**
-2ROT	**(dw1 dw2 dw3 -- dw3 dw1 dw2)**		**Umkehrung zum Befehl 2ROT**
2SWAP	**(dw1 dw2 -- dw2 dw1)**		**Oberste Wertepaar vertauschen**
2TUCK	**(dw1 dw2 -- dw2 dw1 dw2)**		**Kopiert obere Wertepaar unter das zweite Wertepaar**
2VARIABLE	**(name ; --)**		**Definition einer 32Bit-Variablen**
'ABORT	**(--)**	**D**	**Defer für eigenes Autostart-Programm (Default: IDENT)**
ABORT	**(??? --)**		**Stack löschen und nach 'ABORT QUIT aufrufen**
ABORT"	**(flag --) (C: string" ; --)**	**I,R**	**Bei flag<>0: Stack löschen und ERROR ausgeführen**
ABS	**(n1 -- +n2)**		**Subtrahiert n1 von 0, wenn n1 negativ ist (Absolutwert)**
ALIAS	**(name ; cfa --)**		**Definition eines ALIAS-Befehls**
ALIGN	**(--)**		**Aligned Dictionaryadresse, wenn notwendig**
ALIGNED	**(addr1 -- addr2)**		**Aligned Adresse, wenn notwendig**
ALLOT	**(n --)**		**Reserviert n Adressen im Dictionary**
ALSO	**(--)**		**Übernimmt aktuelles Vokabular in den festen Teil**
AND	**(mask1 mask2 -- mask3)**		**Logische AND-Verknüpfung**
ASCII	**(ccc ; -- char)**		**Liefert Wert des ersten Zeichens im nachfolgenden String**
ASHIFT	**(n1 n2 -- n3)**		**n1 um n2 Bits verschieben (n2<0: nach rechts)**
AT	**(x y --)**	**UV**	**Neue Cursorposition setzen**
AT?	**(-- x y)**	**UV**	**Aktuelle Cursorposition abfragen**
BASE	**(-- addr)**	**U**	**Enthält aktuelle Zahlenbasis**
BEGIN	**(--) (C: -- 2 addr 2)**	**I,R**	**Anfang einer ...UNTIL oder ...WHILE...REPEAT-Schleife**
BELL	**(--)**	**UV**	**Ton ausgeben**
BINARY	**(--)**		**Zahlenbasis 2 einstellen**
BLANK	**(addr u --)**		**Füllen von u Bytes ab addr**
BLK	**(-- addr)**	**U**	**Enthält die Nummer des aktuellen Blocks (0=TIB)**
BLOCK	**(u -- addr)**		**Liefert die Adresse des ersten Bytes im Block u**
BODY>	**(pfa -- cfa)**		**Ermittelt Codefeldadresse aus Parameterfeldadresse**
'BOOT	**(--)**	**D**	**DEFER-Wort zum Einbau eigener BOOT-Routinen**
BOOT	**(??? --)**		**Befehl enthält BOOT-Routine, führt aber COLD aus**
BOUNDS	**(n1 len -- n2 n3)**		**Schleifenparameter aus Anfangswert und Länge**
BRANCH	**(--)**	**R**	**Bedingungsloser Sprung (Offset als Inline-Wert)**
BUFFER	**(u -- addr)**		**Wie BLOCK, jedoch ist der Inhalt undefiniert**
BYE	**(--)**		**Verlassen des KK-FORTH nach 'BYE (liefert UFLAG als Fehlerflag ans Betriebssystem)**
'BYE	**(--)**	**D**	**DEFER-Wort für eigene Abschlußbefehle (Default: -IDENT)**
C!	**(w addr --)**		**Speichert das niederwertige Byte von w ab addr**
C,	**(char --)**		**Speichert Byte im Dictionary**
C@	**(addr -- byte)**		**Liefert Byte von angegebener Adresse**
CAPACITY	**(-- u)**		**Filegröße abfragen**
CAPS	**(-- addr)**	**V**	**Enthält Flag, ob Befehle nur in Großschrift sind**
CASE	**(--) (C: -- 5 5)**	**I,R**	**Anfang einer CASE-Struktur**
CASE?	**(n1 n2 -- n1 0 | -1)**		**Vergleich zweier Werte (liefert -1, wenn n1=n2)**
CELL+	**(addr1 -- addr2)**		**Liefert nächste Zellenadresse**
CELLS	**(n -- addr)**		**Liefert Anzahl der Adressen für n Zellen**
CHAR+	**(addr1 -- addr2)**		**Liefert nächste Zeichenadresse**
CHARS	**(n -- addr)**		**Liefert Anzahl der Adressen für n Zeichen**
CLOSE	**(--)**		**Schließt aktuelles File**
CLS	**(--)**	**UV**	**Bildschirm löschen**
CMOVE	**(addr1 addr2 u --)**		**Kopiert u Bytes von addr1 nach addr2 (aufsteigend)**
CMOVE>	**(addr1 addr2 u --)**		**Wie CMOVE, jedoch in absteigender Reihenfolge**
CODE?	**(-- addr)**	**V**	**Enthält Adresse des letzten Dictionaryeintrages mit ,C oder ,A**
'COLD	**(--)**	**D**	**DEFER-Wort zum Einbau eigener Warmstart-Routinen**
COLD	**(??? --)**		**Restart ab EMPTY; löscht beide Stacks, schießt File**
COMMAND!	**(com -- error)**	**X**	**Befehlsübertragung zum Terminal**
COMPILE	**(--)**		**Kompiliert nachfolgenden Inline-Befehl**
CONSTANT	**(name ; w --)**		**Definition einer Konstante**
CONTEXT	**(-- addr)**		**Liefert Adresse des obersten Suchvokabulars**
CONVERT	**(du1 addr1 -- ud2 addr2)**		**String ab addr1+1 zu ud1 akkumulieren**
COUNT	**(csa -- addr len)**		**Anfangsadresse und Länge eines Counted-String bei csa**
COUNT>0	**(addr -- addr len)**		**Zählt die Zeichen im String bis zum 0-Ende**
CR	**(--)**	**UV**	**Zum Anfang der nächsten Zeile gehen**
CREATE	**(name ; --)**	**DX**	**Erzeugt ein Befehlsname ohne reservierten Speicher**
CS@	**(-- ptr)**		**Liefert Anfang des Programmspeichers**
CURRENT	**(-- addr)**	**U**	**Enthält Adresse+2 des aktuellen Kompiler-Vokabular**
D-	**(d1 d2 -- d3)**		**Subtrahiert d2 von d1**
D.	**(d --)**		**Ausgabe von d im freien Format mit Leerzeichen**
D.R	**(d n --)**		**Rechtsbündige Ausgabe von d im Feld mit n Zeichen**
D?	**(addr --)**		**Ausgabe des Inhalts einer 32Bit-Variablen**
D+	**(d1 d2 -- d3)**		**Addiert d2 zu d1**
D<	**(d1 d2 -- flag)**		**Flag ist wahr, wenn d1 kleiner d2 ist**
D<>	**(d1 d2 -- flag)**		**Flag ist wahr, wenn d1 ungleich d2 ist**

D=	**(d1 d2 -- flag)**		**Flag ist wahr, wenn d1 gleich d2 ist**
D>	**(d1 d2 -- flag)**		**Flag ist wahr, wenn d1 größer d2 ist**
D>PTR	**(d -- ptr)**		**Wandelt eine 32Bit-Adresse in einen Pointer**
D>S	**(d -- n)**		**Wandlung eines 32Bit-Wertes in einen 16Bit-Wert**
D0<	**(d -- flag)**		**Flag ist wahr, wenn d kleiner 0 ist**
D0<>	**(d -- flag)**		**Flag ist wahr, wenn d ungleich 0 ist**
D0=	**(d -- flag)**		**Flag ist wahr, wenn d gleich 0 ist**
D0>	**(d -- flag)**		**Flag ist wahr, wenn d größer 0 ist**
D2*	**(d1 -- d2)**		**Schieben von d1 um ein Bit nach links (Bit 0=0)**
D2/	**(d1 -- d2)**		**Schieben von d1 um ein Bit nach rechts (Bit15 bleibt)**
DABS	**(d1 -- +d2)**		**Subtrahiert d1 von 0, wenn d1 kleiner 0 ist (Absolutwert)**
DCLEAR	**(??? --)**		**Löschen des Datenstacks**
'DCLEAR	**(??? -- ???)**	**D**	**Befehl zum Entfernen eigener Werte vor DCLEAR**
DECIMAL	**(--)**		**Zahlenbasis 10 einstellen**
DEFER	**(name ; --)**		**Definition eines DEFER-Befehls**
DEFINITIONS	**(--)**		**Das erste Suchvokabular wird zum Kompiler-Vokabular**
DEL	**(--)**	**UV**	**Zeichen vor dem Cursor löschen**
DELETE	**(name ; --)**		**Löscht angegebenes File**
DEPTH	**(-- n)**		**Anzahl der Stackeinträge vor Ausführung des Befehls**
DISC	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Filebefehle**
DMAX	**(d1 d2 -- d3)**		**d3 ist der größere Wert**
DMIN	**(d1 d2 -- d3)**		**d3 ist der kleinere Wert**
DNEGATE	**(d1 -- d2)**		**Subtrahiert d1 von 0**
DO	**(n1 n2 --) (C: -- addr 3)**	**I,R**	**Anwendung: DO...LOOP oder DO...+LOOP**
DOES>	**(-- addr)**	**I,R**	**Leitet den Ausführungsteil einer CREATE-Definition ein**
DP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Dictionaryende**
DPL	**(-- addr)**	**U**	**Enthält Anzahl der Ziffern nach einem Punkt nach >NUMBER**
DROP	**(w --)**		**Entfernt w vom Datenstack**
DS@	**(-- ptr)**		**Liefert Anfang des Datenspeichers**
DU<	**(du1 du2 -- flag)**		**Flag ist wahr, wenn du1 kleiner du2 ist**
DU>	**(du1 du2 -- flag)**		**Flag ist wahr, wenn du1 größer du2 ist**
DU2/	**(du1 -- du2)**		**Schieben von du1 um ein Bit nach rechts (Bit15=0)**
DUMAX	**(du1 du2 -- du3)**		**du3 ist der größere Wert**
DUMIN	**(du1 du2 -- du3)**		**du3 ist der kleinere Wert**
DUMP	**(addr u --)**		**Anzeige von u Adressen ab addr**
DUP	**(w -- w w)**		**Dupliziere w**
DWITHIN	**(d1 d2 d3 -- flag)**		**Flag ist wahr, wenn d2<d1<d3 ist**
EDITSTRING	**(addr maxlen pos len -- pos2 len2)**	**UV**	**String ausgeben und editieren (ändert SPAN)**
ELSE	**(--) (C: addr1 1 -- addr2 -1)**	**I,R**	**Anwendung: IF...ELSE...THEN**
EMIT	**(char --)**	**UV**	**Ausgabe des Zeichen char**
EMIT?	**(-- f)**	**UV**	**Liefert Flag, ob Zeichen ausgegeben werden kann**
EMPTY	**(--)**		**Zurückholen des SAVE-Zustandes (löscht neue Befehle)**
EMPTY-BUFFERS		**(--)**	**Gibt alle Diskblöcke ohne Sicherung frei**
ENDCASE	**(n --) (C: 5 ... 5 --)**	**I,R**	**Ende einer CASE-Struktur**
ENDOF	**(--) (C: -5 -- addr -2 5)**	**I,R**	**CASE-Struktur: Sprung zu ENDCASE**
ERASE	**(addr u --)**		**Füllt u aufeinanderfolgende Bytes mit Null**
'ERROR	**(--)**	**D**	**Wird von (ERRORHANDLER vor QUIT aufgerufen**
ERROR	**(error | cfa $7fff | cfa $ffff --)**		**Fehlerbehandlung bei error ungleich 0**
ERROR"	**(flag --) (C: string" ; --)**	**I,R**	**Bei flag<>0: ERROR mit angegebenen String aufgerufen**
ERRORHANDLER	**(-- addr)**	**U**	**Enthält CFA der aktuellen Fehlerroutine**
ERRORTEXT@	**(error | cfa $7fff -- addr len)**	**DX**	**Holt Text zur Fehlernummer**
EVALUATE	**(addr len --)**		**Angegebenen String interpretieren/kompilieren**
EXECUTE	**(cfa --)**		**Führt Befehl mit angegebener CFA aus**
EXIT	**(-- ; R: sys --)**	**R**	**Aussprung aus einem FORTH-Befehl**
EXPECT	**(addr +n --)**		**Erwartet Empfang von maximal n Zeichen (ändert SPAN)**
FALSE	**(-- 0)**		**Falsch-Wert für alle Vergleiche im KKF**
FILE.	**(--)**		**Aktueller Filename ausgeben**
FILE-FCB	**(-- addr)**	**V**	**Enthält Zeiger auf aktuellen Filename (CSA mit 0-Ende)**
FILE-ID	**(-- addr)**	**V**	**Enthält Handlenummer des aktuellen Screenfiles**
FILE-LINK	**(-- addr)**	**V**	**Enthält Zeiger auf zuletzt definierten Fileeintrag**
FILES	**(--)**		**Alle gespeicherten Filenamen ausgeben**
FILL	**(addr u byte --)**		**Füllt u Bytes ab addr mit Wert byte**
FIND	**(csa -- csa 0 | cfa n)**	**DX**	**Sucht Befehl, liefert CFA (|2|=Restrict;0< bei Immediate)**
FIRST	**(-- sys)**		**Anfangsadresse des Diskettenpuffers**
FLIP	**($xxyy -- $yyxx)**		**High- und Low-Byte vertauschen**
FLUSH	**(--)**		**Führt SAVE-BUFFERS aus und gibt alle Diskpuffer frei**
FOR	**(u -- ; R: -- u) (C: -- addr 4)**	**I,R**	**Anwendung: FOR...NEXT (wiederholt Befehle n+1 mal)**
FORGET	**(name ; --)**		**Löschen eines Befehls und alle nachfolgenden Wörter**
FORTH	**(--)**		**Ersetzte das erste Suchvokabular durch das FORTH**
H,	**(w --)**		**Speichert Wort im Heap**
HALIGN	**(--)**		**Aligned Heapadresse**
HALLOT	**(n --)**		**Reserviert n Adressen im Heap (verschiebt Variablen)**
HC,	**(char --)**		**Speichert Byte im Heap**

HCLEAR	**(--)**		**Heap löschen**
HDP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Heap-Anfangs**
HEADERS	**(--)**		**Alle nachfolgenden Befehlsheader zum Dictionary**
-HEADERS	**(--)**		**Alle nachfolgenden Befehlsheader zum Heap bringen**
HEAP	**(-- addr)**		**Liefert Adresse des HEAP-Anfangs**
HEAP?	**(addr -- flag)**		**Flag ist wahr, wenn addr im Heap liegt**
HERE	**(-- addr)**		**Liefert Adresse des aktuellen Dictionaryende**
HEX	**(--)**		**Zahlenbasis 16 einstellen**
HIDE	**(--)**		**Neu definierten Befehl verstecken**
HLD	**(-- addr)**	**U**	**Enthält Anfangsadresse des <# # #s #> -Zahlenstrings**
HLEN	**(-- sys)**	**S**	**Enthält aktuelle Länge des Heap-Bereiches**
HOLD	**(char --)**		**Übernimmt ein Zeichen in den Zahlenstring**
I	**(-- n)**		**Liefert aktuellen Wert der innersten Schleife**
I'	**(-- n)**		**Liefert Endwert der innersten Schleife**
IDENT	**(--)**		**Einschaltmeldung ausgeben**
-IDENT	**(--)**		**Endemeldung ausgeben**
IF	**(flag --) (C: -- addr 1)**	**I,R**	**Anwendung: IF ... [ELSE ...] THEN**
IMMEDIATE	**(--)**		**Markiert letzten Befehl als Immediate**
INCLUDE	**(name ; --)**		**Screen 1 vom angegebenes File laden**
INDENT	**(--)**		**Anzahl der Leerzeichen stehen in INDENT?**
INDENT?	**(-- addr)**	**V**	**Enthält Anzahl der auszugebenden Leerzeichen für INDENT**
INDIRECT	**(--)**		**Markiert letzten Befehl als Indirect**
INPUT	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Eingabebefehle**
INTERPRET	**(--)**		**Aktuelle Befehlszeile interpretieren/kompilieren**
IS	**(name ; cfa --)**		**Umleitung eines DEFER-Befehls auf die angegebene CFA**
J	**(-- n)**		**Liefert aktuellen Wert der nächst äußeren Schleife**
J'	**(-- n)**		**Liefert Endwert der nächst äußeren Schleife**
KEY	**(-- char)**	**UV**	**Empfängt ein Zeichen (wartet)**
KEY?	**(-- f)**	**UV**	**Test, ob Zeichen bereitsteht**
L!	**(n ptr --)**	**X**	**Wort in exterenen Speicher schreiben**
L@	**(ptr -- n)**	**X**	**Wort aus externen Speicher holen**
L>NAME	**(lfa -- nfa)**		**Ermittelt Namensfeldadresse aus Linkfeldadresse**
L2!	**(d ptr --)**	**X**	**Doppelwort schreiben (höherwertiges Wort zuerst)**
L2@	**(ptr -- d)**	**X**	**Doppelwort holen (höherwertiges Wort zuerst)**
LABEL	**(name ; --)**	**I**	**Aktuelle Dictionary-Adresse mit >LABEL merken**
LAST	**(-- addr)**	**V**	**Enthält NFA des zuletzt definierten Befehls**
LC!	**(b ptr --)**	**X**	**Byte in externen Speicher schreiben**
LC@	**(ptr -- b)**	**X**	**Byte aus externen Speicher holen**
LCMOVE	**(ptr1 ptr2 u --)**	**X**	**Kopiert u Bytes bytes von ptr1 nach ptr2 (aufsteigend)**
LCMOVE>	**(ptr1 ptr2 u --)**	**X**	**Wie LCMOVE, jedoch in absteigender Reihenfolge**
LDUMP	**(ptr u --)**	**X**	**Anzeige beliebiger Speicheradressen**
LEAVE	**(--) (C: --)**	**R**	**Verläßt sofort die innerste Schleife**
LFILL	**(ptr u byte --)**	**X**	**Externer Speicher füllen**
LIMIT	**(-- sys)**		**Erste freie Speicheradresse über dem KKF**
LINK>	**(lfa -- cfa)**		**Ermittelt Codefeldadresse aus Linkfeldadresse**
LIST	**(+n --)**		**Screen n anzeigen**
LITERAL	**(n --)**	**I,R**	**Kompiliert Wert als Inline-Literal**
LMOVE	**(ptr1 ptr2 u --)**	**X**	**Wahlweise LCMOVE oder LCMOVE>**
LOAD	**(u --)**		**Laden Block u (0 nicht erlaubt)**
LOADFROM	**(name ; u --)**		**Screen u vom angegebenen File laden**
LOOP	**(--) (C: addr 3 --)**	**I,R**	**Ende ...LOOP-Schleife; addiert 1 zum Schleifenindex**
LWFILL	**(ptr u w --)**	**X**	**Externer Speicher mit u Worte füllen (low zuerst)**
M-	**(d1 n -- d2)**		**Subtrahiert n von d1**
M*	**(n1 n2 -- d)**		**Multipliziert n1 mit n2 (32Bit-Ergebnis)**
M/	**(d1 n1 -- n2)**		**Dividiert d durch n1; liefert Quotient n2**
M/MOD	**(d n1 -- n2 n3)**		**Dividiert d durch n1; liefert Rest n2 und Quotient n3**
M+	**(d1 n -- d2)**		**Addiert n zu d1**
MAKE	**(name ; --)**		**Erzeugt neues File mit Länge 0 und öffnet es mit OPEN**
MAX	**(n1 n2 -- n3)**		**n3 ist der größere Wert**
MAXAT	**(-- x y)**	**UV**	**Maxmimale Größe des Bildschirm liefern**
MAXTLEN@	**(-- len)**		**Liefert verwendete Gesamtlänge der Tasks**
MIN	**(n1 n2 -- n3)**		**n3 ist der kleinere Wert**
MOD	**(n1 n2 -- n3)**		**Dividiert n1 durch n2; liefert Rest n3**
MORE	**(u --)**		**File um u Blöcke (je 1024 Byte) vergrößern**
MOVE	**(addr1 addr2 u --)**		**Wahlweise CMOVE oder CMOVE> (nicht überlappend)**
MULTITASK	**(--)**		**Bei PAUSE wird der Multitasker aufgerufen**
N>LINK	**(nfa -- lfa)**		**Ermittelt Linkfeldadresse aus der Namensfeldadresse**
NAME>	**(nfa -- cfa)**		**Ermittelt Codefeldadresse aus Namensfeldadresse**
NEGATE	**(n1 -- n2)**		**Subtrahiert n1 von Null**
NEXT	**(-- ; R: u -- u-1 |) (C: addr 4 --)**	**I,R**	**Ende einer FOR...NEXT-Schleife (Ende, wenn u=0)**
NEXTBRANCH	**(-- ; R: n -- n-1 |)**	**R**	**Sprung und n-1, wenn Wert <> 0 (Offset als Inline-Wert)**
NEXT-LINK	**(-- addr)**	**V**	**Enthält Zeiger auf zuletzt definierten (NEXT-Verkettung**
NIP	**(w1 w2 -- w2)**		**Entferne zweiten Stackeintrag**

NODEFER	**(--)**		**Fehlerbehandlung bei nicht initialisierten DEFER**
NOOP	**(--)**		**Leerbefehl für nicht aktive DEFER's**
NOT	**(w1 -- w2)**		**Invertiert aller Bits in w1**
NOTFOUND	**(--)**		**Fehlerbehandlung bei nicht gefundenen Befehlen**
NUMBER	**(n -1 | d 0> --)**	**DX**	**Interpretiert NUMBER?-Flags (läßt Werte auf dem Stack)**
NUMBER,	**(n -1 | d 0> --)**	**DX**	**Kompiliert 16Bit- oder 32Bit-Wert als Inline-Literal**
NUMBER?	**(csa -- addr 0 | n -1 | d 0>)**	**DX**	**Test auf Zahlenstring (Prefix $,& oder % und Vorzeichen)**
OF	**(n1 n2 -- n1 |) (C: 5 -- addr -5)**	**I,R**	**CASE-Struktur: Vergleich und Sprung wenn ungleich**
OFF	**(addr --)**		**Variable ab Adresse addr wird mit 0 beschrieben**
ON	**(addr --)**		**Variable ab Adresse addr wird mit -1 beschrieben**
ONLYFORTH	**(--)**		**FORTH wird zum Such- und Kompilier-Vokabular**
OPEN	**(name ; --)**		**Öffnet File (legt Filename auch als Befehl an**
OR	**(mask1 maks2 -- mask3)**		**Logische OR-Verknüpfung**
ORDER	**(--)**		**Zeigt die aktuelle Suchreihenfolge an**
OUTPUT	**(-- addr)**	**U**	**Enthält Zeiger auf Tabelle mit Ausgabebefehle**
OVER	**(w1 w2 -- w1 w2 w1)**		**Kopiert den zweiten Stackeintrag zum TOS**
P!	**(w addr --)**		**Wort in (benachbarte) Portadresse(n) schreiben**
P@	**(addr -- w)**		**Wort aus (benachbarte) Portadresse(n) lesen**
PAD	**(-- addr)**		**Liefert Adresse des Stringbereiches für Zahlenausgabe**
PARSER	**(csa -- ???)**	**D**	**Befehlswort interpretieren/kompilieren**
PAUSE	**(--)**	**D**	**Aufruf des Multitaskers**
PC!	**(b addr --)**	**X**	**Bytes in Port einschreiben**
PC@	**(addr -- b)**	**X**	**Byte aus Port lesen**
PERFORM	**(addr --)**		**Führt Befehl mit der in addr kompilierten CFA aus**
PICK	**(wu ... w1 w0 u -- wu ... w1 w0 wu)**		**Kopiert u-ten Stackeintrag zum TOS**
POSTPONE	**(--)**	**I,R**	**Verwendet COMPILE oder [COMPILE] (wenn Immediate)**
PTR>D	**(ptr -- d)**		**Wandelt Pointers in eine 32Bit-Adresse**
PUSH	**(addr -- ; R: -- w addr pop)**	**R**	**Variable bis zum nächsten EXIT merken**
QUERY	**(--)**	**UV**	**Nächsten Befehlsstring holen**
QUIT	**(--)**		**Interpreterschleife (endlos)**
R#	**(-- addr)**	**U**	**Enthält Position des letzten Fehlers im Text/Screen**
R@	**(-- w ; R: w -- w)**	**R**	**Kopiert obersten Returnstackwert zum Datenstack**
R>	**(-- w ; R: w --)**	**R**	**Übertragt obersten Returnstackwert zum Datenstack**
R0	**(-- addr)**	**U**	**Enthält Anfangsadresse des Returnstack-Speichers**
'RCLEAR	**(--)**	**D**	**Befehl zum Entfernen eigenener Werte vom Returnstack**
RCLEAR	**(-- ; R: ??? --)**	**R**	**Löschen des Returnstacks**
RDEPTH	**(-- n)**		**Anzahl der Werte auf dem Returnstack**
RDROP	**(-- ; R: w --)**	**R**	**Entfernt obersten Returnstackwert**
RECURSE	**(--)**	**R**	**Aktuellen Befehl rekursiv aufrufen**
REMOVE	**(dp heap --)**		**Alle Befehle zwischen dp und heap entfernen**
'REMOVE	**(dp heap -- dp heap)**	**D**	**Entfernen eigener Befehlsstrukturen**
REPEAT	**(--) (C: 2 ... addr 2 --)**	**I,R**	**Ende einer BEGIN...REPEAT-Definition**
RESTRICT	**(--)**		**Markiert letztes Befehl als Restrict**
REVEAL	**(--)**		**Zuletzt definierten Befehl wieder sichtbar machen**
ROLL	**(wu wt ... wa u -- wt ... wa wu)**		**Rotieren der obersten u Stackeinträge (ohne u)**
ROT	**(w1 w2 w3 -- w2 w3 w1)**		**Rotieren der obersten drei Stackeinträge**
-ROT	**(w1 w2 w3 -- w3 w1 w2)**		**Umkehrung zum Befehl ROT**
RP!	**(addr --)**	**R**	**Setzt neue Returnstackadresse oder -tiefe**
RP@	**(-- addr)**		**Liefert aktuelle Returnstackadresse oder -tiefe**
S>D	**(n -- d)**		**Wandlung eines 16Bit-Wertes in ein 32Bit-Wert**
S0	**(-- addr)**	**U**	**Enthält Anfangsadresse des Datenstack-Speichers**
SAVE	**(--)**		**Heap löschen, Systemvariablen in (SYSVAR sichern (außer SFLAG und UFLAG)**
SAVE-BUFFERS	**(--)**		**Alle mit UPDATE markierten Blöcke sichern**
SAVESYSTEM	**(name; --)**		**Speicherung des aktuellen FORTH-Systems auf Diskette**
SCAN	**(addr1 len1 char -- addr2 len2)**		**Im String wird nach dem Zeichen char gesucht**
SCAN>	**(addr len1 char -- addr len2)**		**Vom Stringende ab wird nach dem Zeichen char gesucht**
SCR	**(-- addr)**	**U**	**Enthält Nummer des zuletzt gelisteten oder fehlerhaften Screens**
SEAL	**(--)**		**Entfernt oberstes Suchvokabular**
SFLAG	**(-- sys)**	**S**	**Enthält Systemflag (einzelne Bits verwendet)**
SHIFT	**(n1 n2 -- n3)**		**n1 logisch um n2 Bits verschieben (n2<0:nach rechts)**
SIGN	**(n --)**		**Fügt bei n<0 "-" in den Zahlenstring ein**
SINGLETASK	**(--)**		**PAUSE hat keine Wirkung**
SKIP	**(addr1 len1 char -- addr2 len2)**		**Im String wird das angegebene Zeichen char überlesen**
SKIP>	**(addr len1 char -- addr len2)**		**Die Zeichen char am Stringende werden entfernt**
SP!	**(addr --)**		**Setzt neue Datenstackadresse oder -tiefe**
SP@	**(-- addr)**		**Liefert aktuellen Datenstackadresse oder -tiefe**
SPACE	**(--)**		**Ein Leerzeichen ausgeben**
SPACES	**(n --)**		**Falls n positiv ist: n Leerzeichen ausgeben**
SPAN	**(-- addr)**	**U**	**Enthält Anzahl von Zeichen beim letzten EXPECT**
SS@	**(-- ptr)**	**X**	**Liefert Anfang des Stackspeichers**
STANDARD-IO	**(--)**		**Nutze gespeichertes INPUT-, OUTPUT- und DISC-Interface**
STATE	**(-- addr)**	**U**	**Flag ob Text interpretiert (STATE=0) oder kompiliert wird**

STOP?	**(-- f)**		**Wartet auf Taste; liefert bei #BRK (meist CTRL+X) -1**
STRING	**(-- addr u)**	**UV**	**u Zeichen ohne Test empfangen und ab addr ablegen**
SWAP	**(w1 w2 -- w2 w1)**	**U**	**Vertauschen der obersten Datenstackeinträge**
SYSCON	**(-- sys)**		**Liefert Anfangsadresse der Systemkonstanten**
SYSVAR	**(-- sys)**		**Liefert Anfangsadresse der Systemvariablen**
SYSVARLEN@	**(-- sys)**		**Liefert Länge des Systemvariablenbereiches**
TASK0	**(-- addr)**		**Liefert USER-Adresse des Standardtask**
TASKADDR	**(-- sys)**	**S**	**Enthält USER-Adresse des aktuellen Tasks**
TASKADDR@	**(-- addr)**		**Liefert Adresse des aktuellen USER-Bereiches**
TASK-LINK	**(-- sys)**	**S**	**Enthält USER-Adresse des zuletzt definierten Tasks**
TASKS	**(-- sys)**	**S**	**Enthält Anzahl der definierten Tasks**
TDP	**(-- sys)**	**S**	**Enthält Anfangsadresse des Taskbereiches**
THEN	**(--) (C: addr |1| --)**	**I,R**	**Ende einer IF…THEN oder IF…ELSE…THEN-Struktur**
THRU	**(u1 u2)**		**Ladet Screens u1 bis einschließlich u2**
TIB	**(-- addr)**		**Liefert Adresse des aktuellen Eingabepuffers**
TLEN	**(-- sys)**	**S**	**Enthält Anzahl der belegten Bytes in den Tasks**
-TRAILING	**(addr len1 -- addr len2)**		**Leerzeichen am Stringende abschneiden**
TRUE	**(-- -1)**		**Wahr-Wert für alle Vergleiche im KKF**
TUCK	**(w1 w2 -- w2 w1 w2)**		**Kopiert TOS unter den nächsten Stackeintrag**
-TYPE	**(addr u --)**		**ASCII-Werte unter $20 als Punkt anzeigen**
TYPE	**(addr u --)**	**UV**	**u Zeichen ab addr ausgeben**
U.	**(u --)**		**Ausgabe von u mit nachfolgenden Leerzeichen**
U.R	**(u n --)**		**u rechtsbündig in einem Feld mit n Zeichen ausgeben**
U?	**(addr --)**		**Vorzeichenlose Ausgabe des Inhalts einer Variablen**
U<	**(u1 u2 -- flag)**		**Flag ist wahr, wenn u1 kleiner u2 ist**
U>	**(u1 u2 -- flag)**		**Flag ist wahr, wenn u1 kleiner u2 ist**
U2/	**(u1 -- u2)**		**u1 logisch um ein Bit nach links schieben (Bit 15=0)**
UFLAG	**(-- sys)**	**S**	**Enthält Anwenderflag (frei verwendbar)**
UM*	**(u1 u2 -- ud)**		**Multipliziert u1 mit u2 (32Bit-Ergebnis)**
UM/MOD	**(ud u1 -- u2 u3)**		**Dividiert ud durch u1; liefert Quotient u3 und Rest u2**
UMAX	**(u1 u2 -- u3)**		**u3 ist der größere Wert**
UMIN	**(u1 u2 -- u3)**		**u3 ist der kleinere Wert**
UNLOOP	**(--)**	**R**	**Entfernt LOOP-Parameter**
UNTIL	**(flag --) (C: 2 … addr 2 --)**	**I,R**	**Ende einer BEGIN…UNTIL-Struktur**
UPC	**(char1 -- char2)**		**Wandelt ein Zeichen in Großschrift (auch Umlaute)**
UPDATE	**(--)**		**Markiert zuletzt angeforderten Block als verändert**
UPPER	**(addr len --)**		**Wandelt alle Zeichen des Strings in Großbuchstaben um**
USER	**(name ; --)**		**Definition einer USER-Variable**
UTABLE:	**(name ; len user --)**		**Definition einer neuen Tabelle für UVECTOR-Befehle**
UVECTOR	**(name ; # user --)**		**Definition eines UVECTOR-Befehls**
V,	**(w --)**		**Speichert Wort im Variablenbereich**
VALIGN	**(--)**		**Aligned Variablenadresse**
VALLOT	**(n --)**		**Reserviert n Adressen im Variablenbereich**
VARIABLE	**(name ; --)**		**Definition einer Variable**
VC,	**(char --)**		**Speichert Byte im Variablenbereich**
VCREATE	**(name ; --)**		**Erzeugt eine Variable ohne reservierten Speicher**
VDOES>	**(-- vaddr)**	**I,R**	**Leitet den Ausführungsteil einer VCREATE-Definition ein**
VDP	**(-- sys)**	**S**	**Enthält Adresse des aktuellen Variablenanfangs**
VHERE	**(-- addr)**		**Liefert Adresse des aktuellen Variablenende**
VLEN	**(-- sys)**	**S**	**Enthält aktuelle Länge des Variablenbereiches**
VOCABULARY	**(name ; --)**		**Definition eines Vokabulars**
VOC-LINK	**(-- sys)**	**S**	**Enthält Zeiger auf zuletzt definiertes Vokabular**
VOCS	**(--)**		**Anzeige aller Vokabulare**
WDP@	**(-- addr)**		**Liefert Anfangsadresse des Arbeitsbereiches für WORD**
WHILE	**(flag --) (C: addr1 2 -- addr2 -2 addr1 2)**	**I,R**	**Bei flag=0 die BEGIN…UNTIL / BEGIN…REPEAT-Schleife verlassen**
WITHIN	**(w1 w2 w3 -- flag)**		**Flag ist wahr, wenn w2<w1<w3 ist**
-WORD	**(char -- csa)**		**Nachfolgender String wird zu WDP@ kopiert (mit 0-Ende)**
WORD	**(char-- csa)**		**Nachfolgender String nach SKIP mit 0-Ende zu WDP@**
WORDS	**(--)**		**Befehlsliste ausgeben**
XOR	**(mask1 mask2 -- mask3)**		**Logische XOR-Verknüpfung**

Anhang B

Glossar

Bezeichnung der Stackparameter:

(C: ...)	Veränderungen auf dem Datenstack während des Kompilierens
(R: ...)	Veränderungen auf dem Returnstack
(name ; ...)	Es wird noch ein Befehlsname erwartet
flag	Flag (0=ff=Falsch; sonst tf=Wahr)
b	Byte
n	Einfachgenauer, vorzeichenbehafteter Wert
+n	Einfachgenauer, positiver Wert oder 0
u	Einfachgenauer, vorzeichenloser Wert
w	Nicht definierter, einfachgenauer Wert
addr	Adresse
sys	Systemabhängie Speicheradresse
lfa	Linkfeld-Adresse
nfa	Namensfeld-Adresse
cfa	Codefeld-Adresse
pfa	Parameterfeld-Adresse
csa	Counted-String-Adresse
d	Doppeltgenauer, vorzeichenbehafteter Wert
+d	Doppeltgenauer, positiver Wert oder 0
ud	Doppeltgenauer, vorzeichenloser Wert
wd	Nicht definierter, doppeltgenauer Wert
ptr	32Bit-Zeiger für externen Speicherzugriff PC oder Z80: ptr = seg:addr RTX: ptr = bank:addr 68000 ptr = addrh:addrl

Bezeichnung der Befehlsart:

I	Dieser Befehl ist IMMEDIATE
R	Dieser Befehl ist RESTRICT
Con	Konstante
SV	System-Variable
U	USER-Variable
V	Variable
UT	UTABLE:-Definition
UV	UVECTOR-Befehl
D	Mit NOOP vorbelegter DEFER-Befehl
DX	DEFER-Befehl mit nachfolgender (versteckter) Runtime-Routine
X	Dieser Befehl ist nicht in allen KKF-Versionen verfügbar
83	Befehl des FORTH83-Standards
E83	Zusatzbefehle zum FORTH83-Standard
ANSI	Befehl des gepanten ANSI-Standards

! (w addr --) 83
Der 16Bit-Wert w wird ab der Speicheradresse addr abgelegt. Bei einigen 16Bit-Prozessoren (RTX, 68000) wird der Befehl fehlerhaft bearbeitet oder löst eine Fehlermeldung aus, wenn es sich bei addr um eine ungerade Adresse handelt.

" (String" -- csa) 83 I
Im Kompiler-Mode wird der nachfolgende String bis zum abschießenden " (oder bis zum Zeilenende) als Inline-String in das aktuelle Wort übernommen. Bei der Ausführung wird dann dessen "Counted-String-Adresse" auf dem Stack abgelegt.
Abweichend zu anderen FORTH-Versionen kann der Befehl auch im Interpreter-Mode eingegeben werden. Hier wird dann der String zum **PAD** gebracht und die PAD-Adresse sofort als csa zurückgeliefert. Ein zweiter mit " eingegebener String überschreibt dabei den ersten String.
In beiden Fällen wird an den String eine 0 angehängt. Deshalb ist es möglich, die csa nach Erhöhung um 1 als fcb-Adresse zu verwenden.

(ud1 -- ud2) 83
Der Befehl dient zur Umwandlung einer 32Bit-Zahl in einen Zahlenstring. Dazu wird der vorzeichenlose 32Bit-Wert durch die aktuelle Zahlenbasis geteilt, der Rest als Ziffer vor dem aktuellen Zahlenstring gesetzt und der 32Bit-Quotient wieder auf dem Stack gelegt. Siehe dazu auch die Befehle **<#** , **#S** , **#>** , **HLD** und **BASE** .
Anwendung: `: xxxxx <# # # # # # #> type ; ( immer 5 Stellen )`

#> (ud -- addr len) 83
Ende einer Zahlenstring-Generierung. Die noch auf dem Stack verbliebene 32Bit-Zahl wird entfernt und dafür die Anfangsadresse des Zahlenstrings und dessen Länge abgelegt. Es kann danach direkt der Befehl **TYPE** verwendet werden.

#B/BLK (-- $400) Con
Die Systemkonstante **#B/BLK** gibt an, wieviel Zeichen (Bytes) in einem Diskettenblock untergebracht sind. Da im KKF Screenfiles verwendet werden, beträgt die Länge 1024 Bytes.

#BL (-- $20) Con
Der ASCII-Wert des Leerzeichens (32 = $20) wird auf dem Stack abgelegt. im FORTH83-Standard wird dazu der Befehl **BL** verwendet.

#BRK (-- $18) Con
Der ASCII-Wert für die Abbruchtaste wird auf dem Stack abgelegt. Im KKF für den PC und in den Terminalversionen wird dazu CTRL+X verwendet.

#C/L (-- $40) Con
Die Konstante **#C/L** gibt an, wieviele Zeichen in einer Screenzeile verfügbar sind.

#CR (-- $0D) Con
Der ASCII-Wert der ENTER-Taste wird auf dem Stack abgelegt.

#DELIN (-- sys) Con
Der ASCII-Wert der DEL-Taste wird auf dem Stack abgelegt. Dieses Zeichen ist von der Tastatur des jeweiligen Computers abhängig. Bei PC und den Terminal-Versionen liefert **#DELIN** den Wert $08.

#DELOUT (-- sys) Con
Bei der Ausgabe dient der durch **#DELOUT** gelieferte ASCII-Wert zur Verschiebung des Cursors um eine Position nach Links. Es werden aber keine Zeichen gelöscht.

#ESC (-- $1B) Con
Der ASCII-Wert des Escape-Code &27 wird auf dem Stack abgelegt. Dieser Wert wird vom Eingabebefehl **EDITSTRING** zum Zurückholen der letzten Eingabe oder von **COMMAND!** zur Befehlsübertragung zum Terminal verwendet.

#L/BLK (-- $10) Con
Die Konstante **#L/BLK** gibt die Anzahl der Zeilen in einem Screen an. Der Wert ist traditionsgemäß auf 16 gesetzt und hat nur Bedeutung bei der Ausgabe von Screens (Editor oder Listing). Bei der Interpretation eines Screens werden immer alle 1024 Zeichen als linearer String verwendet.

#RO (-- sys) Con
Attribut für **(FILE-OPEN** , daß nur gelesen werden soll.

#RW (-- sys) Con
Attribut für **(FILE-OPEN** , daß sowohl gelesen als auch geschrieben werden soll. Dieser Wert wird als Attribut für Screenfiles verwendet.

#S (ud1 -- 0.) 83
Wie **#** dient dieser Befehl zur Erzeugung eines Zahlenstrings. Jedoch wird in **#S** der Befehl **#** mindestens einmal aufgerufen und danach solange wiederholt, bis der Rest 0. übrigbleibt.

#TIB (-- addr) U,83
#TIB ist eine USER-Variable, in der die Gesamtlänge der zu interpretierenden Zeile steht. Sie wird von **LOAD** oder **QUERY** gesetzt und dient dem Befehl **WORD** zur Erkennung des Zeilen-/Screenende.

#TIB-MAX (-- sys) Con
Konstante mit der maximalen Zeichenanzahl im Eingabepuffer **TIB**. Der Wert wird von **QUERY** verwendet und ist beim KKF_PC auf 79 Zeichen gesetzt.

#WO (-- sys) Con
Attribut für **(FILE-OPEN** , daß nur geschrieben werden muß.

$, (String; char --)
Legt den nachfolgende String bis zum abschließenden Zeichen char oder Zeilenende als "Counted-String" mit 0-Ende im Dictionary ab. Führende Zeichen mit ASCII-Wert char werden ignoriert.

' (Name; -- cfa) 83
Die Codefeldadresse des nachfolgenden Befehls wird ermittelt. Falls kein Befehlsname folgt oder der Befehl nicht gefunden wurde, wird eine Fehlerbehandlung eingeleitet.

'ABORT (--) D
Die DEFER-Routine **'ABORT** wird beim Start des FORTH und bei Aufruf von ABORT

aufgerufen. Sie zeigt im KKF auf **IDENT** und gibt deshalb die Einschaltmeldung aus. In eigenen Applikationen können durch Änderung dieses Vektors eigene Meldungen ausgegeben oder Autostartprogramme erzeugt werden.

'BOOT (--) D

Die mit **NOOP** besetzte DEFER-Routine '**BOOT** wird nach dem Start des FORTH-Systems nur einmal aufgerufen und dient zur Initialisierung eigener Hard- und Software.

'BYE (--) D

Die DEFER-Routine '**BYE** wird vor dem Verlassen des FORTH-Programmes aufgerufen. Sie zeigt im KKF auf **-IDENT** und gibt deshalb die Abschiedsmeldung aus. In eigenen Applikationen können auch eigene Meldungen oder Operation eingefügt werden.

'COLD (--) D

Die üblicherweise auf **NOOP** gesetzte DEFER-Routine dient zur Einfügung eigener Reinitialisierungsroutinen in die Warmstart-Sequenz. Der Befehl wird sowohl nach **BOOT** als auch nach **COLD** unmittelbar vor **ABORT** aufgerufen.

'DCLEAR (??? -- ???) D

Die mit **NOOP** besetzte DEFER-Routine '**DCLEAR** wird von **DCLEAR** aufgerufen und dient zur Sicherung von Anwenderdaten auf dem Datenstack. '**DCLEAR** darf den Datenstack beliebig verändern, da er danach sowieso gelöscht wird.

'ERROR (--) D

Nach der Ausgabe der Fehlermeldungen, aber unmittelbar vor Aufruf von **QUIT** wird die DEFER-Routine '**ERROR** aufgerufen. Sie ist mit **NOOP** besetzt und dient zur Einfügung eigener Routinen zur Fortsetzung eines Programmes nach Fehler. Man muß aber dann (wie auch **QUIT**) den Returnstack löschen und das Programm mit **BYE** verlassen.

'RCLEAR (--) (R: ??? -- ???)

Die mit **NOOP** besetzte DEFER-Routine '**RCLEAR** wird von **RCLEAR** vor dem Löschen des Returnstacks aufgerufen. Werden vom eingebundenen Befehl der Returnstack verändert, so müssen mindestens die obersten zwei Werte erhalten bleiben, um wieder zurück nach **RCLEAR** und dann zum aufrufenden Wort zu gelangen.

'REMOVE (here heap -- here heap) D

Die mit **NOOP** besetzte DEFER-Routine '**REMOVE** wird von **REMOVE** vor dem Entfernen der Befehle aufgerufen und dient zur Sicherung von Anwenderdaten. Der Datenstack darf von '**REMOVE** nicht verändert werden.

((String, --) 83 I

Der nachfolgende String bis zum abschließenden ")" oder bis zum Zeilen-/Screenende wird übersprungen. Der Befehl wird in FORTH für die Eingabe von Kommentaren verwendet.

(;CODE (-- ; R: addr --) R

Diese Runtimeroutine wird z.B. von **DOES>** oder **;CODE** verwendet. Das Wort, in dem **(;CODE** steht wird verlassen. Zuvor wird jedoch die Adresse nach **(;CODE** als neue Codefeldadresse in den zuletzt definierten Befehl übernommen. Falls ein so veränderter Befehl ausgeführt wird, erfolgt sofort die Ausführung des hinter **(;CODE** stehenden Assemblerprogrammes.

Beispiel: Definition einer Variable in KKF_PC

```
: VARIABLE  CREATE  0 ,              ( Befehl, 16Bit-Wert)
            ;CODE   TOS PUSH,        ( TOS sichern )
                    W TOS MOV,       ( CFA zum TOS )
                    2 TOS ADD,       ( zur PFA )
                    NEXT,            ( nächster Befehl )
            END-CODE
```

(DISC (--) UT

Die Filebefehle werden wieder durch die Routinen des KKF-Kerns durchgeführt.

(ERRORHANDLER (error --)

Die USER-Variable **ERRORHANDLER** zeigt normalerweise auf die Fehlerbehandlungsroutine **(ERRORHANDER** . Dieser Befehl gibt zusammen mit dem verursachenden Wort entweder den Text oder die Nummer des Fehlers aus. Bei Unterlauf des Datenstacks wird zusätzlich eine Meldung ausgegeben und der Stack wieder auf den Default-Wert gesetzt. Die Screennummer des Fehlers wird in **SCR** und die Position in **R#** gespeichert. Falls der Fehler bei der Interpretation eines Screens ausgelößt wurde oder das Bit 15 der Fehlernummer gesetzt ist, wird der Datenstack gelöscht und der Interpretermodus wieder aktiviert.
Nach Ausgabe der Statusmeldung ("]" oder "ok") durch **.STATE** und dem Aufruf von '**ERROR** wird dann wieder zur Interpreterschleife **QUIT** zurückgesprungen.

(FILE-CLOSE (handle -- error) UV

Schließen eines offenen Files mit angegebener Handlenummer. Ob dabei auch alle Daten sofort zum Speichermedium zurückgeschrieben werden, hängt vom Betriebssystem ab.

(FILE-CREATE (string/0 -- handle 0 | error) UV

Anlegen eines leeren Files mit angegebenem Filename. Falls ein gleichnamiges File schon existiert, so wird es vorher gelöscht.

(FILE-DELETE (string/0 -- error) UV

Löschen eines Files mit angegebenem Filename. Es sollten keine Wildcards ("*" oder "?") in diesem Filename angegeben werden. Falls dabei das aktuelle Screenfile entfernt werden soll, so muß es vorher durch **CLOSE** geschlossen werden.

(FILE-FIRST (string/0 attr -- dta 0 | error) UV

Suchen nach dem ersten File, dessen Name und Attribut mit den angegebenen Parameter übereinstimmt. Als Fileattribut können beim KKF_PC und damit bei Terminalversionen folgende Werte angegeben werden:

$00	normale Dateien
$01	Nur-Lese-Dateien
$02	Versteckte Dateien
$04	Systemdateien
$08	Volume-Name (Label)
$10	Unterverzeichnisse
$20	Archivierungs-Dateien

In dta wird dann der Name des gefundenen Files und einige Zusatzparameter zurückgeliefert. Beim KKF_PC hat das Datenfeld eine Länge von 43 Bytes und folgenden Aufbau:

dta	21 vom Betriebssystem reservierte Bytes
dta+21	Attribut der Datei
dta+22/23	Uhrzeit der letzten Modifikation (MSDOS-Format)
dta+24/25	Datum der letzten Modifikation (MSDOS-Format)
dta+26/27	Low-Wort der Dateigröße

dta+28/29 High-Wort der Dateigröße
dta+30-42 Dateiname mit Extension (mit 0-Ende)

(FILE-FREE (dev -- free. max. 0 | error) UV
Test des angegebenen Laufwerk dev (0=aktuell, 1=A: ...) auf die Anzahl der maximal verfügbaren und der noch freien Bytes. Da die Laufwerke in Blöcke verwaltet wird, kann es selbst bei genügend freien Bytes noch vorkommen, daß keine Daten mehr gespeichert werden können.

(FILE-NEXT (-- fcb 0 | error) UV
Nachdem mit **(FILE-FIRST** die Suche initialisiert wurde, kann mit **(FILE-NEXT** der nächste Eintrag ermittelt werden. Eine erneute Angabe von Filename oder Attribut ist nicht mehr erforderlich.

(FILE-OPEN (string/0 attr -- handle 0 | error) UV
Öffnen eines schon vorhandenen Files zur Bearbeitung. Es gibt für das Attributbyte mehrere Möglichkeiten, die vom verwendeten System abhängig sind. Als Konstanten sind #RO , #WO und #RW verfügbar. Falls dieses File nicht existiert, so wird eine Fehlermeldung ausgegeben.

(FILE-POS! (pos. dir handle -- error) UV
Der aktuelle Lese-/Schreibzeiger eines Files wird auf die Position pos (32Bit-Wert) gesetzt. Das Flag dir entscheidet darüber, ob diese Position ab dem Fileanfang (dir=0), der aktuellen Position (dir=1) oder ab dem Fileende (dir=2) gerechnet wird.

(FILE-POS@ (handle -- pos. 0 | error) UV
Die aktuelle Position des Lese-/Schreibzeiger eines Files wird ermittelt und als 32Bit-Wert auf dem Stack abgelegt.

(FILE-READ (ptr len handle -- error) UV
Ab der aktuellen Position des mit handle angegebenen Files werden len Bytes gelesen in im Speicher ab der (32Bit-)Adresse ptr abgelegt.

(FILE-SIZE (handle -- len. 0 | error) UV
Die Gesamtlänge des angegebenen Files wird ermittelt und als 32Bit-Wert auf dem Datenstack abgelegt. Dazu werden die beiden Befehle **(FILE-POS!** und **(FILE-POS@** herangezogen, aber der Lese-/Schreibzeiger wieder auf seinen ursprünglichen Wert zurückgesetzt.

(FILE-WRITE (ptr len handle -- error) UV
Ab der (32Bit-)Adresse ptr werden len Bytes gelesen und ab der aktuellen Position des mit handle angegebenen Files geschrieben. Wird dabei über das Fileende hinausgeschrieben, so vergrößert sich das File. Bei Fehler wird die Anzahl der schon übertragenen Bytes nicht zurückgeliefert.

(FILE? (fcb -- error) UV
Test ob ein File mit angegebenem Filename verfügbar ist. Dazu wird dieses File geöffnet und sofort wieder geschlossen.

(FIND (-1 lfa1 lfa2 ... csa -- csa 0 | lfa -1)

(FIND ist meistens als schnelle Assemblerroutine realisiert und sucht in den angegebenen Vokabularen nach einem Befehl, der mit dem String csa übereinstimmt. Ist der Befehl gefunden worden, so wird dessen Linkfeldadresse lfa und das Flag -1 auf dem Stack abgelegt. Ansonsten bleibt die Stringadresse csa und das "Fehlerflag" 0 auf dem Stack.
Es dürfen beliebig viele Vokabulare angegeben werden. Mit -1 ist das Ende der Liste gekennzeichnet. Bei aufeinanderfolgenden gleichen lfa's wird nur einmal gesucht.

(FORGET (addr --)

Entfernt alle Worte, deren Codefeldadresse oberhalb Adresse addr liegt, aus dem Dictionary und setzt **DP** auf addr. Ein Fehler liegt vor, falls addr im Heap liegt. Dieses Wort kann verwendet werden, um schon mit **SAVE** geschützte Befehlswörter freizugeben. Dabei sollte die Linkfeldadresse des Befehls (`' Name >LINK`) übergeben werden.

(INPUT (--) UT

Die Eingabe erfolgt wieder durch das im Kern vorgesehene Eingabegerät. Sind dazu Initialisierungen notwendig, so werden sie nur dann durchgeführt, wenn Bit 3 von **SFLAG** auf 0 steht.

(MALLOC (len1. -- ptr 0 | len2. error) X

Dieser spezielle Befehl zur Reservierung externer Speicher liefert einen Zeiger auf das erste freie Byte. Bei Fehler wird der Speicher nicht belegt und nur die verfügbare Restlänge zurückgeliefert. Der Pointer ptr muß bei **(MRELOC** und **(MFREE** (Veränderung der Speichergröße oder Freigabe des Speichers) wieder angegeben werden.

(MFREE (ptr -- error) X

Freigeben eines mit **(MALLOC** reservierten Speichers. Ein Fehler wird ausgegeben, falls ptr nicht auf den mit **(MALLOC** belegten Speicher zeigt.

(MRELOC (ptr len. -- error) X

Ein mit **(MALLOC** belegter Speicher kann in seiner Größe verändert werden. Eine Vergrößerung ist nur dann möglich, wenn kein weiterer Speicher angefordert wurde.

(OPEN (string/0 -- handle 0 | error)

Öffnen eines vorhandenen Diskettenfiles mit angegebenem Namen und **#RW** -Attribut. Es wird eine Handlenummer zurückgeliefert, die bei allen Fileoperationen und beim Schließen des Files angegeben werden muß.

(OUTPUT (--) UT

Die Ausgabe wird wieder auf das im Kern vorgesehene Ausgabegerät zurückgestellt. Sind dazu Initialisierungen notwendig, so werden sie nur dann durchgeführt, wenn Bit 3 von **SFLAG** auf 0 steht.

(SYSVAR (-- sys)

(SYSVAR liefert die Adresse des von **SAVE** für die Speicherung der Systemvariablen verwendeten Bereichs. Diese Adresse ist selbst in den Systemvariablen (ab Offset 12) gespeichert und zeigt bei ROM-Versionen des KK-FORTH auf den RAM-Anfang.

* (n1 n2 -- n3) 83

Multiplikation der beiden vorzeichenbehafteten 16Bit-Werte n1 und n2. Es wird kein Fehler ausgegeben, falls eine Bereichsüberschreitung im Produkt n3 auftritt.

*/ (n1 n2 n3 -- n4) 83
Multiplikation von n1 mit n2 zu einem 32Bit-Produkt, welches dann durch n3 geteilt wird. Nur der Quotient n4 wird zurückgeliefert. Alle Werte sind vorzeichenbehaftete 16Bit-Werte. Falls durch 0 dividiert wird oder ein Überlauf auftritt, so wird je nach KKF-Version n4=-1 zurückgeliefert oder eine Fehlerbehandlung eingeleitet.

*/MOD (n1 n2 n3 -- n4 n5) 83
Wie bei ***/** werden n1 mit n2 multipliziert und dann durch n3 dividiert. Zusätzlich zum Quotient n5 wird noch der Divisionsrest n4 zurückgeliefert. Wie bei allen vorzeichenbehafteten Divisionen ist zu beachten, daß immer das der Rest immer das Vorzeichen des Divisors hat und deshalb auch negativ sein kann.

+ (w1 w2 -- w3) 83
Addition der 16Bit-Werte w1 und w2 zur Summe w3. Da mit dem Zweierkomplement gearbeitet wird, kann der Befehl sowohl bei vorzeichenlosen als auch bei vorzeichenbehafteten Werten verwendet werden. Ein Überlauf wird dabei nicht erkannt, sondern nur der niederwertige 16Bit-Teil zurückgeliefert.

+! (w addr --) 83
Addition des Wertes w zum Inhalt der 16Bit-Variablen ab Adresse addr.

+LOOP (n --) 83 I,R
(C: addr 4 --)
Beispiel: `: UNGERADE 20 1 DO I . 2 +LOOP ;`
Abschluß einer **DO** oder **?DO**-Schleife. Der Wert n wird zum Schleifenindex addiert. Falls durch die Addition die Grenze zwischen Endwert-1 und Endwert überschritten wurde, wird die Schleife beendet und die Parameter vom Returnstack entfernt. Ansonsten wird die Schleife ab **DO** bzw. **?DO** wiederholt. Beim Kompilieren des Befehls wird auf dem Datenstack die Rücksprungadresse und der Kontrollwert 4 erwartet.

, (w --) 83
Ablage des 16Bit-Wertes w ab der aktuellen Dictionary-Adresse. Der Zeiger **DP** wird dabei um ein Wort (meist 2 Adressen) erhöht. Der Befehl wird sowohl zur Kompilierung von Programmen als auch zur Ablage von Werten in Datenstrukturen verwendet.

,A (cfa --) DX
Nach der Speicherung der aktuellen Dictionaryadresse in **CODE?** wird die angegebene Codefeldadresse kompiliert. Dabei wird zum Beispiel beim KKF_RTX auch eine Anpassung der Codefeldadresse durchgeführt (hier **U2/**).

,C (w --) DX
Nach der Speicherung der aktuellen Dictionaryadresse in **CODE?** wird der angegebene Wert im Dictionary abgelegt. Ob es sich bei w um ein Byte oder ein Wort handelt, ist Versionsabhängig.

- (w1 w2 -- w3) 83
Subtraktion des Wertes w2 von w1. Wie bei **+** können sowohl vorzeichenbehaftete als auch vorzeichenlose 16Bitwerte verwendet werden, wobei ein Überlauf nicht erkannt wird.

-! (w addr --)
Analog zu +! wird der Wert w von der ab Adresse addr abgelegten 16Bit-Variable subtrahiert und das Ergebnis wieder ab addr gespeichert.

-$, (char --)
Der nachfolgende String bis zum Zeilenende oder dem abschließenden Zeichen char wird als Counted-String mit führender Längenangabe und abschließenden 0-Ende ins Dictionary kompiliert. Anders als bei **$,** werden die Zeichen char am Stringanfang nicht übergangen. Es kann deshalb mit **-$** auch ein Leerstring eingegeben werden.

--> (--) I
Beim Laden eines Screens wird die Interpretation/Kompilierung des aktuellen Blocks abgebrochen und am Anfang des nächsten Screens fortgesetzt. Dieser Befehl kann auch innerhalb einer :-Definition verwendet werden.

-2ROT (dw1 dw2 dw3 -- dw3 dw1 dw2)
Analog zu **-ROT** werden durch **-2ROT** 32Bit-Werte rotiert.

-HEADERS (--)
Durch Setzen der Variable **>HEAD?** auf -1 wird angegeben, daß bei alle nachfolgenden Befehlen der Namensheader getrennt vom Programmcode auf dem Heap abgelegt wird.

-IDENT (--)
Standardroutine zum DEFER-Befehl **'BYE** . Dieser Befehl wird unmittelbar vor Verlassen des FORTH-Programmes aufgerufen und gibt noch eine Abschlußmeldung aus. **'BYE** kann aber auch auf eigene Abschaltmeldungen oder einfach zu **CLS** umgeleitet werden.

-ROT (w1 w2 w3 -- w3 w1 w2) 83
Die obersten drei Datenstackeinträge werden rotiert. Dieser Befehl ist die Umkehrung von **ROT** .

-TRAILING (addr u1 -- addr u2) 83
Der Befehl **-TRAILING** wird zum Abschneiden der Leerzeichen am Ende eines Strings verwendet. Der String hat eine Länge von u1 Zeichen und steht ab Adresse addr im FORTH-Speicher. Der String selbst wird durch **-TRAILING** nicht verändert, sondern nur die Länge korrigiert.

-TYPE (addr u --)
Analog zu **TYPE** wird ein String mit Länge u ab Speicheradresse addr ausgegeben. Dabei werden aber alle Zeichen mit ASCII-Wert unter 32 (also alle Steuerzeichen) als Punkt ausgegeben. Dadurch werden ungewünschte Effekte bei Ausgaben in **DUMP** und **LIST** verhindert.

-WORD (char -- addr)
Analog zu **WORD** wird aus der aktuellen Eingabe (Tastatur oder Disk) der nächste mit char begrenze String ermittelt und mit Längenabgabe und abschießender Null zu Adresse addr (in KKF immer **WDP@**) gebracht. Abweichend zu **WORD** werden die Zeichen char am Stringanfang nicht überlesen und es kann deshalb auch ein Leerstring eingegeben werden.

. (n --) 83
Ausgabe des vorzeichenbehafteten 16Bit-Wertes n in der aktuellen Zahlenbasis und mit nachfolgenden Leerzeichen. Nur bei negativen Werten wird auch ein Vorzeichen ausgegeben.

." (--) 83 I,R
Anwendung: `: -IDENT ." Leave KK-FORTH" ;`
Der nachfolgende String bis zum abschießenden Anführungszeichen oder zum Zeilenende wird kompiliert und bei Ausführung des Befehls ausgegeben. Der Befehl kann nur kompiliert werden, da es für den Interpretermodus gibt es den Befehl **.(** zur direkten Ausgabe von Kommentaren gibt.

.((--) 83 I
Sofortige Ausgabe des nachfolgenden Strings ohne abschließenden) . Der Befehl kann auch in :-Definitionen verwendet werden, wird aber immer sofort ausgeführt.

.BLK (--) DX
Der DEFER-Befehl **.BLK** dient zur Ausgabe von Informationen während des Laden von Screenfiles. Die versteckt unmittelbar hinter **.BLK** abelegte Standardroutine zeigt statt der Screennummer nur ein ">" beim Laden eines Blockes an.

.ID (csa --)
Der ab csa abgelegte Counted-String wird mit **-TYPE** ausgeben. Vorher wird aber vom Längenbyte die Bit 5-7 ausmaskiert und dadurch die Länge des Strings auf 31 Zeichen begrenzt. **.ID** wird von **WORDS** zur Ausgabe von Befehlsnamen herangezogen.

.R (n1 n2 --) 83
Ausgabe des vorzeichenbehafteten 16Bit-Wertes n1 rechtsbündig in einem Feld mit n2 Zeichen. Es wird dabei die aktuelle Zahlenbasis verwendet und nur bei negativen Werten ein Vorzeichen ausgegeben. Überschreitet der Ausgabestring die vorgegebene Feldlänge, so wird der Zahlenstring trotzdem vollständig ausgegeben.

.S (... -- ...)
Ausgabe von bis zu 16 Datenstackeinträge ohne Veränderung des Stacks. Dabei werden die Werte als vorzeichenbehaftete 16Bit-Zahlen in der aktuellen Zahlenbasis und als vorzeichenlose Zahlen in der dezimalen (Prefix &) und hexadezimalen (Prefix $) Zahlenbasis ausgegeben. Der oberste Stackeintrag wird auch als erster Wert ausgegeben. Die Ausgabe kann mit beliebiger Taste angehalten oder mit #BRK-Taste auch abgebrochen werden.

.STATE (--) DX
DEFER-Befehl **.STATE** dient zur Ausgabe von Statusinformationen nach dem Interpretieren/Kompilieren einer Eingabe. Die unmittelbar hinter **.STATE** versteckt abgelegte Standardroutine gibt je nach Status entweder in der gleichen Zeile noch " ok" aus oder kennzeichnet erst nach dem Zeilenumbruch mit "]" , daß noch kompiliert wird.

/ (n1 n2 -- n3) 83
Division des vorzeichenbehafteten Wertes n1 durch n2. Es wird nur der Quotient n3 zurückgeliefert. Bei Division durch 0 wird entweder eine -1 zurückgeliefert oder eine Fehlermeldung eingeleitet.

/$ (csa/0 -- addr)
Da bei einigen Prozessoren ein Align der Programme oder 16Bit-Daten durchgeführt werden

muß, wurde ein eigener Befehl zum Überspringen eines Counted-String mit 0-Ende eingeführt. Der z.B. von " oder **ERRORTEXT@** verwendete Befehl liefert die erste verwendbar Adresse hinter dem String.

/MOD (n1 n2 -- n3 n4) 83
Wie bei **/** wird n1 durch n2 geteilt, aber jetzt Quotient n4 und Rest n3 zurückgeliefert.

/STRING (addr1 len1 len2 -- addr2 len3)
Der String ab addr1 mit Länge len1 wird am Anfang um len2 Zeichen gekürzt. Der String wird dabei nicht verändert, sondern nur die neue Anfangsadresse addr2 (=addr1+len2) und die neue Länge len3 (=len1-len2) berechnet.

0< (n -- f) 83
Ein TRUE-Flag (f=-1) wird zurückgeliefert, wenn der vorzeichenbehaftete 16Bit-Wert n negativ ist.

0<> (w -- f)
Es wird ein TRUE-Flag zurückgeliefert, wenn der Wert w ungleich Null ist.

0= (w -- f) 83
Es wird ein TRUE-Flag zurückgeliefert, wenn der 16Bit-Wert w gleich Null ist.

0=EXIT (f --) (R: addr --) R
0=EXIT ist das Gegenstück zu **?EXIT** . Der aktuelle FORTH-Befehl wird verlassen und zum aufrufenden Befehl zurückgekehrt, wenn das Flag f den Wert 0 hat. Da die Rückkehradresse auf dem Returnstack aufbewahrt wird, dürfen sich dort keine mit **>R** abgelegten Werte befinden.

0> (w -- f)
Es wird ein TRUE-Flag zurückgeliefert, wenn der 16Bit-Wert w größer als Null ist.

0U.R (u n --)
Wie bei **U.R** wird der vorzeichenlose 16Bit-Wert u rechtsbündig in einem Feld von n Zeichen ausgegeben. Statt mit Leerzeichen wird das Feld aber mit "0" aufgefüllt. Anwendung findet der Befehl z.B. bei **DUMP** .

1+ (w -- w+1) 83
Erhöhung des 16Bit-Wertes w um 1. Der Befehl ist meist in Assembler definiert und deshalb schneller und kürzer als `1 +` .

1- (w -- w-1) 83
Erniedrigung des 16Bit-Wertes w um 1.

2! (dw addr --) 83
Der 32Bit-Wert dw wird ab Adresse addr im Speicher abgelegt. Der Aufbau und der benötigte Speicherplatz dazu ist vom Prozessor abhängig. Es wird aber immer das höherwertige Wort zuerst abgelegt. Aus den beiden niederwertigsten Bit des Attribut-Wortes kann die Reihenfolge ermittelt werden.
Z80: addr+0 Bit 16-23

	addr+1	Bit 24-31	(höchstwertigste Byte)
	addr+2	Bit 0-7	(niederwertigste Byte)
	addr+3	Bit 8-15	
PC:	wie Z80		
68000:	addr+0	Bit 24-31	(höchstwertigste Byte)
	addr+1	Bit 16-23	
	addr+2	Bit 8-15	
	addr+3	Bit 0-7	(niederwertigste Byte)
RTX:	wie 68000		

2* (w1 -- w2) 83
w1 wird durch Bitverschiebung nach links verdoppelt.

2+ (w -- w+2) 83
Erhöhung des 16Bit-Wertes w um 2.

2- (w -- w-2) 83
Erniedrigung des 16Bit-Wertes w um 2.

2/ (n1 -- n2) 83
Vorzeichenrichtiges Halbieren des 16Bit-Wertes n1 durch arithmetisches Schieben um ein Bit nach rechts. Bit 15 bleibt dabei erhalten.

2>R (w1 w2 --) (R: -- w1 w2) ANSI R
Dieser im ANSI-Vorschlag enthaltene Befehl befördert zwei 16Bit-Werte vom Datenstack zum Returnstack. Dieser Befehl ist nicht identisch mit `>R >R` , da die Reihenfolge unverändert bleibt.

2@ (addr -- dw) E83
Ein ab Adresse addr im Speicher abgelegter 32Bit-Wert dw wird zum Datenstack geholt. Zum Aufbau des Wertes im Speicher siehe **2!** .

2CONSTANT (name; dw --) E83
Anwendung: `$7FFF.FFFF 2CONSTANT MAXDW`
Anlegen einer 32Bit-Konstante unter nachfolgenden Befehlsname. Der auf dem Datenstack befindliche 32Bit-Wert wird im Befehl gespeichert und bei jedem Aufruf des neuen Wortes wieder zum Datenstack gebracht.

2DROP (dw --) E83
Der 32Bit-Wert dw wird vom Datenstack entfernt.

2DUP (dw -- dw dw) E83
Der 32Bit-Wert dw wird dupliziert.

2LITERAL (dw --) E83 I,R
Der auf dem Datenstack befindliche 32Bit-Wert wird als Literal in ein Programm übernommen. **2LITERAL** sollte nur innerhalb von :-Definitionen verwendet werden und dient zur Ablage von berechneten Werten im Programm.
Beispiel: `: 40*40 [ 40 40 M* ] 2LITERAL ;`

2NIP (dw1 dw2 -- dw2)
Entfernen des zweiten 32Bit-Eintrages vom Datenstack.

2OVER (dw1 dw2 -- dw1 dw2 dw1) E83
Der zweite 32Bit-Eintrag wird dupliziert.

2R> (-- w1 w2) (R: w1 w2 --) ANSI R
Die beiden mit **2>R** zum Returnstack gebrachten Werte werden wieder zum Datenstack zurückgeholt.

2R@ (-- w1 w2) (R: w1 w2 -- w1 w2) ANSI R
Die beiden mit 2>R zum Returnstack gebrachten Werte werden zum Datenstack kopiert.

2RDROP (--) (R: w1 w2 --) R
Vom Returnstack werden zwei 16Bit-Einträge entfernt.

2ROT (dw1 dw2 dw3 -- dw2 dw3 dw1) E83
Rotieren der obersten drei 32Bit-Einträge.

2SWAP (dw1 dw2 -- dw2 dw1) E83
Vertauschung der obersten beiden 32Bit-Einträge.

2TUCK (dw1 dw2 -- dw2 dw1 dw2)
Analog zu **TUCK** wird hier der oberste 32Bit-Eintrag unter den zweiten 32Bit-Eintrag kopiert.

2VARIABLE (--) E83
Anwendung: `2VARIABLE Name`
Anlegen einer 32Bit-Variable. Der Zugriff auf diese Variable kann mit **2@** oder **2!** erfolgen.

: (--) 83
(C: -- 0)
Anwendung: `: <name> ... ;`
Mit **:** wird eine Befehlsdefinition eingeleitet. Der nachfolgende Name wird ins Dictionary übernommen und durch Aufruf von **]** die Statusvariable **STATE** auf -1 gesetzt und der Kompilermodus eingeschaltet. Alle nachfolgenden Befehle und Literals bis zum abschießenden Strichpunkt werden kompiliert. Die bei ... aufgelisteten Befehle werden erst dann ausgeführt, wenn man den neu definierten Namen eingibt oder ein Befehl aufgeruft, daß den definierten Namen kompiliert hat. Nur Immediate-Befehle wie Kontrollstrukturen oder das abschießende ; werden direkt ausgeführt und verwenden bzw. kontrollieren und entfernen daß durch **:** auf dem Datenstack abgelegte 0-Flag. Da der gerade in der Definition befindliche Befehl noch nicht als gültiges Wort verfügbar ist, kann ein alter, gleichlautender Befehlsname aufgerufen werden. Dies ist für die Umdefinition alter Befehle interessant. Muß ein Befehl rekursiv aufgerufen werden, so ist **RECURSE** zu verwenden.

; (--) 83 I,R
(C: 0 --)
Abschluß einer :-Definition. Bevor ein abschießendes **EXIT** zum Befehl kompiliert wird und der neu definierte Befehl danach verfügbar gemacht wird, kontrolliert **;** noch das auf dem Datenstack abgelegte Flag. Ist es ungleich 0, so deutet dies auf eine unvollständige

oder fehlerhafte Kontrollstruktur oder ein nicht verwendetes Literal. Es wird dann mit einer entsprechenden Fehlermeldung abgebrochen.

< (n1 n2 -- f) 83
Die beiden vorzeichenbehafteten 16Bit-Werte werden verglichen. Falls n1 kleiner als n2 ist, wird das TRUE-Flag -1 zurückgeliefert.

<# (--) 83
Anwendung: `: u. 0 <# #s #> type ;`
<# initialisiert die USER-Variable **HLD** für die Erzeugung eines Zahlenstrings. **HLD** wird dazu auf den aktuellen Wert von **PAD** gesetzt und bei jeder Übernahme eines Zeichens in den String (siehe **HOLD** und **#**) vorher erniedrigt.

<> (w1 w2 --)
Vergleich zweier 16Bit-Werte. Ein TRUE-Flag wird zurückgeliefert, wenn beide Werte ungleich sind.

<MARK (-- addr) E83
<MARK wird zur Markierung der aktuellen Dictionaryposition als Rücksprungadresse verwendet. Der Befehl wird von Befehlen für Kontrollstrukturen wie **BEGIN** und **DO** verwendet, verändert aber den Dictionarypointer nicht.
Anwendung: `: BEGIN <mark ; Immediate`

<RESOLVE (addr --) E83
Der Befehl **<RESOLVE** wertet die durch **<MARK** auf dem Datenstack abgelegte Rücksprungadresse aus. Vor der Ausführung von **<RESOLVE** muß immer ein Sprungbefehl (**?BRANCH** , **BRANCH** oder **NEXTBRANCH**) kompiliert worden sein, da **<RESOLVE** je nach Version die zusätzlich benötigte Sprungadresse korrigiert oder erst erzeugt.
Anwendung: `: UNTIL postpone ?branch <resolve ;`

= (w1 w2 -- f) 83
Vergleich zweier 16Bit-Werte. Ein TRUE-Flag wird zurückgeliefert, wenn beide Werte gleich sind.

> (n1 n2 -- f) 83
Vergleich zweier vorzeichenbehafteter 16Bit-Werte. Ein TRUE-Flag wird zurückgeliefert, wenn n1 größer als n2 ist.

>$ (addr u -- pad)
Die Umkehrung zu **COUNT** macht aus Adresse und Länge eines Strings wieder einen Counted-String mit 0-Ende. Dazu wird der String nach `PAD 1+` kopiert, das Längenbyte davor abgelegt und eine Null angehängt.

>$, (addr u --)
Der angegebene String wird als Counted-String mit 0-Ende im Dictionary abgelegt. Danach wird bei Bedarf die Dictionaryadresse aligned.

>BODY (cfa -- pfa) E83
Liegt die Codefeldadresse eines Befehls vor, so kann mit **>BODY** zu der ersten

Parameterfeldadresse gegangen werden. Jedoch sollte man sich vorher vergewissern, ob dies in der verwendeten KKF-Version bei dem Befehl überhaupt möglich oder nötig ist.

>ERRORTEXT (-- addr)

Die Variable **>ERRORTEXT** enthält einen Zeiger auf eine Tabelle mit allen verwendeten Fehlermeldungen. Der Zeiger wird von **ERRORTEXT@** verwendet und enthält entweder 0 oder zeigt auf einen Speicherbereich mit folgenden Aufbau:

addr+0	$0001	(erste Fehlernummer)
addr+2	$14	(Längenangabe zum String)
addr+3	"Unknown funktionscode"	(Text mit 0-Ende; evtl. Aligned
addr+x	$0002	(nächste Fehlernummer)
...		
addr+y	$0000	(Ende der Liste)

Normalerweise zeigt die Variable auf eine Tabelle mit den englischen Fehlermeldungen. Da die Tabelle immer am Ende des KKF-Kerns abgelegt ist, kann sie mit wenig Aufwand durch eine deutschsprachige Version ersetzt werden.

>HEAP? (-- addr)

Die Variable **>HEAD?** wird als Zähler für die auf den Heap zu kompilierenden Header verwendet. Ist die Variable ungleich 0, so legt **CREATE** den Befehlsnamen auf dem Heap ab und erniedrigt die Variable um eins. Durch **|** wird **>HEAP?** auf 1 gesetzt, falls es vorher 0 war. **-HEADERS** setzt die Variable auf -1 und **HEADERS** setzt sie wieder auf 0.

>IN (-- addr) U,83

Die USER-Variable **>IN** enthält den Offset des nächsten zu lesenden Zeichens aus dem Eingabepuffer. Die Variable wird von **LOAD** gespeichert und zurückgesetzt und von **WORD** erhöht.

>LABEL (name ; w --) I

Der Wert w wird wie bei einer Konstanten einem Namen zugewiesen. Da aber der Befehl vollständig im Heap abgelegt wird, bleibt der Dictionarypointer unverändert. Der erzeugte Befehlsname darf aber nicht kompiliert werden, da nach Löschen des Heap's die entsprechende Routinen zerstört ist. Da aber das erzeugte Wort als "immediate" deklariert wird, kann es selbst dafür sorgen, daß in :-Definitionen ein Inline-Literal gespeichert wird. Weil **>LABEL** selbst ein Immediate-Wort ist und die Verkettungszeiger direkt manipuliert, können >LABEL-Definitionen auch innerhalb von :-Definitionen gemacht werden.

>LINK (cfa -- lfa) E83

Liegt die Codefeldadresse eines Befehls vor, so kann mit **>LINK** die Linkfeldadresse ermittelt werden. Kann die zugehörige Linkfeldadresse nicht gefunden werden (z.B. bei einem versteckten Befehl), so wird 0 zurückgeliefert.

>MARK (-- addr) E83

>MARK wird zur Vorbereitung eines Vorwärtssprunges von Kontrollstrukturen wie **IF** oder **WHILE** verwendet. Vor der Ausführung von **>MARK** muß immer ein Sprungbefehl (**?BRANCH** , **BRANCH** oder **NEXTBRANCH**) kompiliert worden sein, da **>MARK** bei einigen KKF-Versionen (wie KKF_PC und KKF_8415) den Raum für den Offset reserviert. Auf dem Datenstack wird dann die aktuelle Dicionaryposition abgelegt. **>MARK** muß immer mit dem zugehörigen **>RESOLVE** aufgelöst werden.

Anwendung:
```
: IF  postpone ?branch  >mark ;
```

>NAME (cfa -- nfa) E83

Liegt die Codefeldadresse eines Befehls vor, so kann mit **>NAME** die Namensfeldadresse

ermittelt werden. Kann die zugehörige Namensfeldadresse nicht gefunden werden (z.B. bei einem versteckten Befehl), so wird 0 zurückgeliefert.

>NEXTTASK (task1 -- task2)
Aus der angegebenen Tastadresse wird die nächste Taskadresse ermittelt.

>NUMBER (du1 addr1 len1 -- du2 addr2 len2) ANSI
Der ab addr abgelegte String wird gemäß der aktuellen Zahlenbasis zu du1 akkumuliert. Bei einem Fehler wird die Adresse des nicht zu interpretierenden Zeichens und die Restlänge zurückgeliefert. Fehlerfreie Umwandlung wird durch len2 = 0 signalisiert.

>R (w -- ; R: -- w) 83 R
Der nur innerhalb einer :-Definition zulässige Befehl bringt einen 16Bit-Wert vom Datenstack zum Returnstack. Da auf dem Returnstack auch die Rücksprungadressen bei Aufruf anderer FORTH-Worte abgelegt werden, kann auf den Wert nur innerhalb der gleichen Definition mittels **R@** , **R>** und **RDROP** zugegriffen werden.

>RESOLVE (addr --) E83
Der Befehl **>RESOLVE** wertet die durch **>MARK** auf dem Datenstack abgelegte Adresse aus. Der aktuelle Dictionarypointer wird dabei nicht verändert.
Anwendung: `: THEN >resolve ;`

>TIB (-- addr) 83
>TIB enthält die Adresse des Terminal-Input-Puffers. Sie wird von **TIB** gelesen und enthält meistens die Adresse des für jeden Task getrennt verfügbaren Speichers zwischen USER-Bereich und Returnstack (Adresse: `TASKADDR@ SYSVAR &38 + @ + 2+`).

>VADDR (addr1 -- addr2)
addr2 ist die Variablenadresse, dessen Offset ab addr1 abgelegt ist. Da im KK-FORTH die Variablen und Teile der Vokabular-Verkettung getrennt vom Dictionary im Variablenbereich abgelegt sind, wird im Dictionary nur der Offset gespeichert.
Beispiel:
```
: Variable   ( name ; -- )
  Create  vlen @ ,  0 v,     ( Variable erzeugen )
  Does>   >vaddr ;           ( Variablenadresse ermitteln )
```

? (addr --)
Der Wert der 16Bit-Variable ab Adresse addr wird gelesen und vorzeichenbehaftet wie bei **.** ausgegeben.

?ALLOT (n --)
Test, ob noch n Adressen zwischen Dictionary und Variablenbereich verfügbar sind. Falls der Dictionarypointer im Heap liegt, wird kein Test durchgeführt. **?ALLOT** wird von allen ALLOT-Befehlen verwendet und liefert erst dann einen Fehler, wenn kein Speicher mehr für die n Adressen frei ist.

?BLK (u --)
Der Befehl testet, ob ein File offen ist und ob der angegebene Block im momentan aktiven File verfügbar ist.

?BRANCH (f --) 83 (I),R

?BRANCH ist eine Routine, die von vielen Kontrollstruktur-Befehlen mit Fallunterscheidung wie **IF** , **WHILE** und **UNTIL** verwendet wird. **?BRANCH** benötigt noch einen Offset oder eine Adresse, zu der gesprungen wird, wenn das Flag f Null ist. Da der Aufbau eines bedingten Sprunges von der KKF-Version abhängt, sollte er nur zusammen mit **>MARK** oder **<RESOLVE** verwendet werden.

Beispiele:
```
... postpone ?branch <resolve ...
... postpone ?branch >mark    ...
```

?DEPTH (n --)

Test, ob der Datenstack mindestens eine Tiefe von n 16Bit-Werten hat. Dieser Befehl dient zur Kontrolle der richtigen Datenanzahl.

?DNEGATE (d n -- d | -d)

Ist n negativ, so wird der 32Bit-Wert d durch sein Zweierkomplement ersetzt.

?DO (w1 w2 --) I,R
(C: -- addr 3)

?DO markiert den Anfang einer Schleifenstruktur. Anders als bei **DO** werden jedoch die beiden Parameter w1 und w2 miteinander verglichen. Sind beide Werte gleich, so wird das Programm sofort hinter **LOOP** bzw. **+LOOP** fortgesetzt.

?DUP (w -- w w) 83
(0 -- 0)

Nur wenn der Wert ungleich Null ist, wird er ein zweites mal auf dem Datenstack abgelegt.

?ERROR (f error | f csa -1 | f csa $7fff --)

Nur wenn das Flag f ungleich Null ist, wird die Fehlerbehandlung eingeleitet. Ansonsten wird sowohl das Flag als auch die Fehlernummer vom Stack entfernt. Der Befehl ist auch dann verwendbar, wenn das Flag -1 oder $7FFF und die Counted-String-Adresse der Fehlermeldung angegeben wird.

?EXIT (f --) R

Der aktuelle Befehl wird mit **EXIT** verlassen, falls das Flag f ungleich 0 ist.

?LEAVE (f --) R

Dieser Befehl darf nur innerhalb von **DO** ... **LOOP** - Schleifen verwendet werden. Falls das Flag f ungleich 0 ist, wird die Schleife sofort verlassen.

?NAME (-- csa)

Holt wie bei **NAME** den nächsten durch Leerzeichen begrenzten String aus dem Eingabepuffer. Falls der Puffer schon leer ist oder nur noch Leerzeichen enthält, wird eine Fehlerbehandlung eingeleitet.

?NEGATE (n1 n2 -- n1 | -n1)

Wenn der vorzeichenbehaftete 16Bit-Wert n2 negativ ist, wird das Zweierkomplement des ersten 16Bit-Wertes n1 zurückgeliefert. Dieser Befehl wird meist zur Übertragung eines Vorzeichens verwendet.

?OPEN (--)

Der Befehl **?OPEN** testet durch Abfrage der **FILE-ID** , ob gerade ein File geöffnet ist. Beim Systemstart oder nach **CLOSE** ist dieses Flag auf 0 und **?OPEN** führt zu einem Fehler.

?PAIRS (w1 w2 --)

Beide Werte werden miteinander verglichen. Nur wenn w1 ungleich w2 ist, so wird mit der Fehlermeldung ("Unstructured") abgebrochen.
Dieser Befehl wird von den meisten Kontrollstrukturen zum Test der nur während des Kompilierung auf dem Datenstack abgelegten Werte verwendet. Es werden aber dabei die Werte so dupliziert, daß selbst nach einem Fehler der Stack unverändert bleibt und das richtige Kontrollwort eingegeben werden kann.

?STACK (--)

Dieser Befehl dient zum Test des Datenstacks. Falls zuviele Werte vom Datenstack geholt wurden oder zuviele Einträge auf dem Datenstack liegen, so wird der Stack gelöscht und **ERROR** aufgerufen.
Da die meisten Prozessoren nur einen Speicherbereich für Dictionary, Daten- und Returnstack besitzen, kann ein Über-/Unterlauf zur Zerstörung wichtiger Daten führen. Im KK-FORTH ist ein Sicherheitsabstand von einigen Bytes (Wert im USER-Bereich gespeichert) zwischen Stackanfang und nachfolgenden Datenfeld (meist Taskbereich) eingebaut.
?STACK wird im KK-FORTH nur von **INTERPRET** verwendet und testet nach jedem interpretierten oder kompilierten Befehl den Datenstack. Falls in einer Schleife der Datenstack laufend verändert wird, so kann **?STACK** ohne Beeinflussung der Stacktiefe als Test eingebaut werden.

@ (addr -- w) 83

Ab der Adresse addr wird ein 16Bit-Wert gelesen und zum Datenstack gebracht. Die Reihenfolge der Bits im Speicher und die Anzahl der benötigten Adressen sind prozessorspezifisch. Um aufeinanderfolgende Einträge zu lesen, sollte die Adresse nur mit den Befehlen **CELL+** oder `n CELLS +` verändert werden. Bei einigen Prozessoren führt ein Wortzugriff auf ungeraden Adressen zu fehlerhaften Werten oder zu einer Fehlermeldungen.

ABORT (--) 83

Der im FORTH83-Standard aufgeführte Befehl dient zur erneuten Initialisierung des Systems und dem Start der Standardapplikation, welche normalerweise **QUIT** ist. Von **ABORT** wird sowohl Daten- als auch Returnstack gelöscht, der Interpretermodus aktiviert und die Variable **>HEAD?** auf 0 gesetzt.

ABORT" (... f --) I

?ERROR wird mit Fehlernummer $FFFF und nachfolgenden String aufgerufen und dadurch bei f ungleich 0 der Datenstack gelöscht und der nachfolgende Fehlerstring ausgegeben.

ABS (n -- u) 83

Der Absolutwert des vorzeichenbehaftete 16Bit-Wert n wird ermittelt.

ALIAS (cfa --)

Anwendung: `' <oldname> ALIAS <newname>`

ALIAS erzeugt einen neuen Namenseintrag für **<newname>** im Dictionary. Wie bei **DEFER** wird aber statt des Codefeldes nur der Zeiger auf die angegebene Codefeldadresse gespeichert. Da **>NAME** die Namensfeldadresse eines Befehles durch den Zeiger auf die Codefeldadresse ermittelt, wird immer die letzte ALIAS-Definition gefunden.

ALIGN (addr1 -- addr2) ANSI (I)

Einige Prozessoren (wie z.B. 680xx oder RTX-2000) können Befehle oder 16Bit-Werte nur an geraden Adressen lesen. Der Befehl **ALIGN** dient zur Anpassung der Dictionaryadresse und wird automatisch von **CREATE** und $, aufgerufen. Er sollte bei Datenstrukturen im Dictionary auch bei anderen Prozessoren verwendet werden, da dadurch die Programme leichter auf andere Systeme übertragen werden können. Falls kein **ALIGN** notwendig ist, wird dieser Befehl als Immediate-NOOP definiert und beansprucht dadurch keine Programmlaufzeit.

Anwendung:
```
: $,   ( char -- ) ( String zum Dictionary )
            word            ( Stringadresse holen )
            here  over c@ 1+  dup allot  move   ( kopieren )
            0 c,  align ;   ( 0-Ende und align )
```

ALIGNED (addr1 -- addr2) ANSI (I)

Einige Prozessoren erlauben nur dann einen 16Bit-Zugriff auf Werte, wenn die Anfangsadresse gerade ist. Der für das ANSI-FORTH vorgeschlagene Befehl dient der Korrektur der Adresse, falls es notwendig ist. Es wird dann addr1 auf den nächsten geraden Wert aufgerundet.

ALLOT (n --) 83

Veränderung des Dictionaryzeigers **DP** um n Bytes. Ist n negativ, so wird der Speicher wieder freigegeben. Mit positiven Werten können Bytes im Programmspeicher reserviert werden. Falls danach weiter kompiliert wird, sollte durch **ALIGN** eine Aufrundung auf die nächste gerade Speicheradresse durchgeführt werden.

ALSO (--) E83

Im KKF können bis zu 6 Vokabulare in der Suchreihenfolge angegeben werden. Mit **ALSO** wird das erste, auswechselbare Vokabular in die Liste der 5 immer zu durchsuchenden Vokabulare übernommen. Eine Fehlermeldung ("CONTEXT full") wird ausgegeben, falls dort schon 5 Vokabulare gespeichert sind. Nur wenn ein Vokabular mehrfach hintereinander angegeben wurde, wird es nur einmal durchsucht. Mit **ONLYFORTH** kann die ganze Suchreihenfolge bis auf das FORTH-Vokabular gelöscht werden.

AND (w1 w2 -- w3) 83

Es wird eine bitweise UND-Verknüpfung der beiden 16Bit-Werte w1 und w2 durchgeführt und das Ergebnis w3 wieder auf dem Stack abgelegt. Nur wenn die entsprechenden Bits beider Werte auf 1 stehen, wird auch das Bit im Ergebnis auf 1 gesetzt.

Beispiel:
```
          1011110010101101
    AND   0100101001000100
          ----------------
          0000100000000100
```

ASCII (String; -- char) I

Der ASCII-Wert des ersten Zeichen im String nach ASCII wird ermittelt und auf dem Datenstack abgelegt. Im Kompiliermodus wird der Wert als Literal direkt in die Definition übernommen.

ASHIFT (n1 n2 -- n3)

Der 16Bit-Wert n1 wird arithmetisch um n2 Bits nach links (wenn n2 positiv) oder nach rechts (wenn n2 negativ) geschoben. Beim Schieben nach rechts wird das Bit 15 (Vorzeichenbit) sowohl in Bit 14 übernommen als auch wieder in Bit 15 kopiert.

AT (x y --) UV

Der Cursor wird an die angegebene Position (Spalte x, Zeile y) gebracht. Bei **AT** handelt es sich um einen über **OUTPUT** vektorisierten Befehl, der für andere Ausgabegeräte als den Textscreen erst angepaßt werden muß. Da im KKF_PC die Positionierung direkt über BIOS-Funktionen realisiert werden, kann es bei Umleiten der Ausgaben zu Probleme kommen.

AT? (-- x y) UV

Die Umkehrung des Befehls **AT** liefert die aktuelle Position des Cursors auf dem Bildschirm zurück. Es handelt sich ebenfalls um einen über **OUTPUT** vektorisierten Befehl. Da im KKF_PC die Position über BIOS-Funktion direkt ermittelt wird, kann es beim Umleiten der Ausgabe zu Problemen kommen. Falls bei einem Ausgabegerät keine Möglichkeit zur Abfrage der Position existiert (z.B. beim Plotter), so sollte eine Variable verwendet werden. Diese Variable muß dann bei jeder Ausgabe mit **AT** , **CR** , **DEL** , **EMIT** oder **TYPE** aktualisiert werden.

BASE (-- addr) U,83

Die USER-Variable **BASE** enthält den Wert der aktuellen Zahlenbasis. Da intern immer mit binären Werten gerechnet wird, dient **BASE** nur zur Angabe der Zahlenbasis bei Ein-/Ausgabe von Werten. Es sollten nur die Ziffern 0-9 und die Buchstaben A-Z verwendet werden, deshalb sind nur Werte von 2 bis 36 in **BASE** sinnvoll. Am häufigsten werden 2 (binär), 8 (oktal), 10 (dezimal) und 16 (hexadezimal) verwendet.

BEGIN (--) 83 I,R
(C: -- 2 addr 2)

Anfang einer Kontrollstruktur, die mit **REPEAT** oder **UNTIL** abgeschlossen wird. Während der Programmausführung wird der Datenstack nicht verändert. Beim Kompilieren des Strukturbefehls werden das Anfangsflag 2, die aktuelle Dictionaryadresse und wieder 2 als Endflag abgelegt. Der Befehl darf nur in :-Definitionen verwendet werden.
Beispiel: `... BEGIN (Teil 1) f WHILE (Teil 2) REPEAT ...`
Nach Ausführung des Teil 1 wird das Flag f getestet. Falls es 0 ist, wird die Kontrollstruktur verlassen und das Programm hinter **REPEAT** fortgesetzt. Ansonsten wird Teil 2 ausgeführt und danach zu **BEGIN** zurückgesprungen und Teil 1 erneut ausgeführt. Falls **WHILE** fehlt, wird die Programmschleife solange wiederholt, bis durch Aufruf von **EXIT** der gesamte Befehl abgebrochen wird.
Beispiel: `... BEGIN (Teil 1) f1 WHILE (Teil 2) f2 UNTIL ...`
Auch hier wird nach Ausführung des Teil 1 das Flag f1 getestet. Falls es 0 ist, wird die Kontrollstruktur verlassen und das Programm hinter **UNTIL** fortgesetzt. Ansonsten wird Teil 2 ausgeführt und danach das Flag f2 getestet. Nur wenn das Flag 0 ist, wird zu **BEGIN** zurückgesprungen und Teil 1 erneut ausgeführt. Ansonsten wird die Kontrollstruktur verlassen. Auch in diesem Beispiel darf der **WHILE** -Teil fehlen.

BELL (--) UV

Erzeugt einen kurzen Pfeifton am Ausgabegerät. Dazu wird bei den meisten KKF-Versionen einfach der ASCII-Wert 7 ausgegeben. **BELL** ist ein über **OUTPUT** vektorisierter Ausgabebefehl.

BINARY (--)

Die USER-Variable **BASE** wird auf 2 gesetzt und damit die Ein-/Ausgabe von Zahlen auf binäre Werte eingestellt. Bei der Interpretation von Zahlen führen alle Ziffern größer 1 und alle Buchstaben zu Fehlermeldungen, falls nicht mittels Prefix ($ oder &) die Zahlenbasis temporär verändert wird.

BLANK (addr len --) 83

Ein len Bytes langer Speicherbereich ab Adresse addr wird mit dem ASCII-Wert für Leerzeichen ($20) gefüllt.

BLK (-- addr) U,83

In der USER-Variablen **BLK** wird die Nummer des gerade zu interpretierenden Screens gespeichert. Da der Wert 0 als Flag zur Interpretation des Terminal-Input-Puffers verwendet wird, kann der Screen 0 nicht für Programmtext verwendet werden. **LOAD** speichert beim Interpretieren anderer Screens neben **>IN** , **>TIB** und **#TIB** auch **BLK** .

BLOCK (u -- addr) 83

Dieser Standardbefehl des FORTH-83 dient zur Ermittlung der Speicheradresse eines Diskblockes. Dazu wird nach Test, ob das File geöffnet ist und ob die Blocknummer verfügbar ist, die Adresse eines freien Diskpuffers ermittelt (siehe **BUFFER**) und der entsprechende Block aus dem File geladen. Falls man danach der Befehl **UPDATE** ausgeführt, wird diese Blocknummer als verändert gekennzeichnet und vor dem Laden anderer Blöcke zum File zurückgeschrieben.

Achtung: Im KKF wird meist nur ein Diskpuffer verwendet.

BODY> (pfa -- cfa) E83

Zu der angegebenen Parameterfeldadresse pfa wird die Codefeldadresse cfa ermittelt.

BOOT (--)

Nach dem Befehlsheader von **BOOT** ist die Startroutine des KK-FORTH abgelegt. Diese Routine kopiert nach Ermittlung der zu verwendenden Speicheradressen alle Task-, Heap- und Variablenbereiche an ihre Zieladressen. Dabei werden alle Zeiger entsprechend korrigiert. Nach Umschalten auf Singletask, Initialisieren der Ein-/Ausgabe und Vorbereitung des Diskinterface wird zuerst '**BOOT** und dann '**COLD** aufgerufen. Danach startet **BOOT** durch Aufruf von **ABORT** die Interpreterscheife.

Da bei RAM-Versionen des KK-FORTH die BOOT-Routine den ursprüngliche Programmaufbau zerstört, wird der Header von **BOOT** so verändert, daß auch bei Eingabe von **BOOT** nur **COLD** ausgeführt wird.

BOUNDS (start count -- limit start)

BOUNDS errechnet aus dem Anfangswert und der Anzahl der Werte die für eine **DO** ... **LOOP** -Schleife benötigten Anfangs- und Endwerte.

Beispiel:
```
: $.  ( csa -- ) ( String ausgeben )
   count bounds  DO  i c@ emit  LOOP ;
```

BRANCH (--) E83 (I),R

Der Befehl **BRANCH** ist ein von den Strukturbefehlen **ELSE** und **REPEAT** kompilierter unbedingter Sprung. Leider ist im Standard angegeben, daß dieser Befehl immer mit `COMPILE BRANCH` und einem nachfolgenden Offset-Literal kompiliert werden soll. Da im KK-FORTH unbedingte Sprünge z.B. beim RTX-2000 auch durch spezielle ("Assembler"-)Befehle realisert werden, ist immer `POSTPONE BRANCH` zu verwenden. Die Ermittlung und Einbindung der Zieladresse zum BRANCH-Befehl ist immer mit den Befehlen **>MARK** und **>RESOLVE** bzw. **<MARK** und **<RESOLVE** durchzuführen.

Beispiel:
```
: ELSE    ( addr1 1 -- addr2 -1 )
   1 ?paris  postpone branch >mark -1
   rot >resolve ;   immediate restrict
```

BUFFER (u -- addr) 83

Es wird die Adresse eines Diskpuffers ermittelt. Falls der Diskpuffer noch durch ein mit

UPDATE markierten Block belegt ist, wird dieser vorher gespeichert. Der Inhalt des durch **BUFFER** angeforderten Puffers ist aber undefiniert, da die angegebene Blocknummer nicht geladen wird.

BYE (--) 83

Mit **BYE** wird das KK-FORTH verlassen. Vorher werden noch alle Diskpuffer zurückgeschrieben und das geöffnete File geschlossen. Nach **'BYE** werden dann alle von **BOOT** veränderten Systemvariablen zurückgesetzt und das Programm verlassen.
Ist KK-FORTH das einzige Programm im System, so wird ein Neustart des Systems durchgeführt und dadurch das FORTH wie beim Reset gebootet. Da aber kein Reset ausgeführt wurde, haben die nicht vom KK-FORTH verwendeten Register und Ports noch ihren alten Wert.

C! (w addr --) 83

Die niederwertigen 8 Bit (entspricht einem Byte) des 16Bit-Wertes w wird ab der Speicheradresse addr abgelegt. Um die nächste Byteadresse zu ermitteln, sollte **CHARS+** eingesetzt werden.

C, (char --)

Die niederwertigen 8 Bit des Wertes char werden im Dictionary abgelegt und davor die Adresse in **DP** mit **ALLOT** entsprechend erhöht. Mit **CREATE** , **C,** und **,** können im Dictionary eigene Datentypen definiert werden. Werden dabei sowohl 8- als auch 16Bit-Werte abgelegt, so muß darauf geachtet werden, daß einige Prozessoren 16Bit-Werte nur an geraden Adressen lesen können.
Siehe dazu auch die Befehle **ALIGN** , **CELL+** und **CHAR+** .

C@ (addr -- w) 83

Die Umkehrung zu **C!** ließt ab der Speicheradresse addr einen 8-Bit-Wert und legt in auf dem Datenstack ab. Die oberen 8 Bit von w werden mit 0 aufgefüllt.

CAPACITY (-- u)

Bei einem geöffneten File kann mittels **CAPACITY** die Anzahl der verfügbaren Screens ermittelt werden. Dabei ist darauf zu achten, daß bei einem Wert von 3 nur die Screens 0 bis 2 verfügbar sind und daß auf einen unvollständigen Screen am Fileende nicht zugegriffen werden kann.

CAPS (-- addr) V

Bei einem Wert gleich Null in der Variable **CAPS** wird von **FIND** der Befehl **UPPER** nicht aufgerufen und damit Groß-/Kleinschrift in den Befehlen berücksichtigt. Da **CREATE** nur mittels FIND die Umwandlung in Großschrift durchführt, werden auch bei neu erzeugten Befehlen Groß-/Kleinschrift unterschieden. Wird aber dann nachträglich das Flag CAPS wieder ungleich Null gesetzt, können keine in Kleinschrift eingetragenen Befehle mehr aufgerufen werden.

CASE (--) I,R
(C: -- 5 5)

Der Anfang einer **CASE** ... **OF** ... **ENDOF** ... **ENDCASE** -Struktur dient nur zur Ablage zweier Kontrollwerte auf dem Datenstack.

CASE? (n1 n2 -- n1 0 | -1)

Die beiden Werte n1 und n2 werden verglichen. Sind beide identisch, so wird nur das

TRUE-Flag (-1) zurückgeliefert. Sind die Werte verschieden, so bleibt der erste Wert n1 auf dem Stack und ein FALSE-Flag (0) bleibt statt des zweiten Wertes auf dem Datenstack.
Beispiel:
```
: MENU   ( -- ) ( Ende mit E )
   BEGIN  key  ascii e case? ?exit   ...  REPEAT ;
```

CELL+ (addr1 -- addr2) ANSI

Um ohne Kenntnisse des verwendeten Prozessors die Adresse des nächsten 16Bit-Wertes zu ermitteln, wurde der Befehl **CELL+** definiert. Er erhöht die angegebene Adresse auf die nächste Speicherzelle einer Wortadresse. Bei den meisten Prozessoren liegt diese Adresse addr2 um zwei Bytes höher als addr1. Deshalb wird **CELL+** meist als **ALIAS** für **2+** definiert.

CELLS (n -- addr) ANSI

CELLS liefert die Anzahl der Adressen, die durch n 16Bit-Werte belegt werden. Bei byteadressiertem Speicher ist **CELLS** eine ALIAS-Definition von **2*** .

CHAR+ (addr1 -- addr2) ANSI

Der im ANSI-FORTH vorgeschlagene Befehl **CHAR+** dient zur Anpassung einer FORTH-Version an verschiedene Prozessoren. Mit **CHAR+** wird die nächste Byteadresse im Speicher ermittelt. Meistens handelt es sich bei **CHAR+** um eine ALIAS-Definition von **1+** .

CHARS (n -- len) ANSI

Statt wie bei CHAR+ nur die Adresse zu erhöhen kann mit **CHARS** die Speichergröße von n Zeichen ermittelt werden. Bei den meisten Prozessoren belegt ein Zeichen auch nur eine Adresse und len ist deshalb identisch mit n. Um Rechenzeit und Programmplatz zu sparen, wird dann **CHARS** als ALIAS-Befehl für **NOOP** definiert und mit **IMMEDIATE** gekennzeichnet.

CLOSE (--)

Im KK-FORTH wird nur ein File für die Blockbefehle verwaltet. Mit CLOSE wird nach dem Zurückschreiben der mit **UPDATE** markierten Blöcke das File geschlossen.

CLS (--) UV

Löschen des Bildschirms und Setzen des Cursors an die oberste linke Position. Im KKF-PC wird diese Funktion durch direktes Aufrufen des BIOS realisiert und kann deshalb nicht umgeleitet werden.

CMOVE (addr1 addr2 len --) 83

Ein Speicherbereich von len Bytes wird von Adresse addr1 nach addr2 kopiert. Da dabei die alte Information ab addr2 überschrieben wird, können mit **CMOVE** überlagernde Speicherbereiche nur zu niedrigeren Adressen verschoben werden. Ist addr2 größer als addr1, so kann statt **CMOVE** der Befehl **CMOVE>** verwendet werden. Mit **MOVE** wird automatisch der richtige Befehl gewählt.

CMOVE> (addr1 addr2 len --) 83

Wie bei **CMOVE** werden len Bytes ab Adresse addr1 nach addr2 kopiert. Da aber das Kopieren beim letzten Byte beginnt, wird die Adresse addr2 zuletzt überschrieben.

CODE? (-- addr) V

In der Variable **CODE?** wird durch die Befehle **,A** und **,C** die aktuelle Dictionaryadresse

gespeichert. **CODE?** wird z.B. bei einem optimierenden Kompiler zur Kombination mehrerer Befehle verwendet.

COLD (... --)

COLD dient als Warmstartroutine. Nach **SINGLETASK** und erneuter Initialisierung von Disk-Interface und Ein-/Ausgabe wird Dictionary, Heap, Variablen- und USER-Bereich auf dem Zustand des letzten SAVE zurückgestellt und nach '**COLD** über **ABORT** die Interpreterschleife aufgerufen.
Da aber die Diskpuffer nicht gesichert und die Files nicht geschlossen werden, kann es dabei zum Verlußt von Daten kommen.

COMMAND! (command -- error) X

Die angegebene Befehlsnummer wird zum Terminal geschickt und die 16Bit-Fehlernummer als Antwort erwartet. Ein Fehler wird auch dann zurückgeliefert, wenn ein Zeichen vor oder während der Datenübertragung geschickt wird.

COMPILE (--) 83 R

Der nach **COMPILE** folgende Befehl wird in den gerade zu kompilierenden Befehl übernommen und der Programmzeiger entsprechend korrigiert. Dadurch können mittels eines Definitionswortes beliebige Programmteile in andere Befehle kompiliert werden. Verwendet wird **COMPILE** meistens in eigene Kontrollstrukturen. Alternativ zu `COMPILE Name` kann auch `POSTPONE Name` verwendet werden, falls der nachfolgende Name keine IMMEDIATE-Befehl ist.
Beispiel:

```
: i2*   ( -- i*2 ) ( Schleifenwert * 2 )
   compile i   compile 2*  ;   immediate restrict
```

CONSTANT (Name ; w --) 83
(-- w) Ausführung von Name

Der 16Bit-Wert w wird als Konstante mit angegebenen Namen definiert. Jede Ausführung von Name legt den 16Bit-Wert w wieder auf dem Datenstack ab. Im KK-FORTH sollte eine **CONSTANT** -Definition immer dann eingesetzt werden, wenn ein Zahlenwert über das gesamte Programm hinweg konstant bleibt aber bei Übertragung auf andere Geräte geändert werden muß. Beispiele dazu sind Portadressen, Maximalanzahl von Daten oder feste Grenzwerte für Abfragen.

CONTEXT (-- addr) E83

Die im USER-Bereich abgelegte Adresse des auswechselbaren Teils der Suchreihenfolge wird durch **CONTEXT** geliefert. Da die Anzahl der zu durchsuchenden Vokabulare veränderbar ist, ändert sich auch die Adresse addr. Die in **CONTEXT** abgelegte Adresse zeigt auf den Offset-Eintrag des entsprechenden Vokabulars im Dictionary.

CONVERT (d1 addr1 -- d2 addr2) 83

Der ASCII-Text ab addr1+1 wird entsprechend der aktuellen Zahlenbasis (siehe **BASE**) zu d1 akkumuliert. Dazu wird d1 mit der Zahlenbasis multipliziert und der Wert des Zeichens addiert. Falls die USER-Variable DPL auf einem Wert ungleich -1 steht, wird für jedes Zeichen **DPL** erhöht. Die Routine wird beendet, sobald ein Zeichen nicht bearbeitet werden kann. Die Adresse addr2 zeigt dann auf dieses Zeichen. Der Befehl ist nur wegen seiner Festlegung im FORTH83-Standard vorhanden. Statt **CONVERT** sollte im KK-FORTH das mächtigere **>NUMBER** verwendet werden.

COUNT (csa -- addr len) 83

Aus einem Counted-String kann mittels **COUNT** die Anfangsadresse und die Länge des Strings ermittelt werden. Bei einem byteorientierten Prozessor wird dieser Befehl oft dazu

mißbraucht, um bei Ausgabeschleifen das Byte aus der csa zu lesen und gleichzeitig die Adresse csa um eins zu erhöhen.

COUNT>0 (addr -- addr len)

COUNT>0 kann bei einem Filenamen oder bei einem String mit ASCII-Wert 0 als Endemarkierung zur Ermittlung der Länge verwendet werden. Dazu wird ab Adresse addr die Anzahl der Zeichenohne dem 0-Byte gezählt und der Wert zusätzlich auf dem Datenstack abgelegt.

CR (--) UV,83

An das aktuelle Ausgabegerät wird ASCII-Wert 13 (Wagenrücklauf) und der ASCII-Wert 11 (Zeilenvorschub) geschickt. Dadurch erfolgt die nächste Ausgabe am Anfang der nächsten Zeile. Beim Bildschirm wird der Cursor ebenfalls in die nächste Zeile gesetzt oder der Rest des Bildschirms hochgerollt. **CR** ist ein Ausgabebefehl, der über **OUTPUT** vektorisiert wurde und damit an andere Ausgabegeräte angepaßt werden kann.

CREATE (name ; --) DX,83

Die Grundroutine jeder Befehlsdefinition legt den nachfolgenden Namen als neues FORTH-Wort im Dictionary an. Danach liefert **NAME** die Adresse des nachfolgenden Speicherplatzes im Dictionary. Es wird aber kein zusätzlicher Speicher reserviert. Ein Fehler wird ausgelöst, falls nach **CREATE** kein Name angegeben wird. **CREATE** sucht mittels FIND nach einem gleichlautenden Befehlsname. Dabei wird die Umwandlung der Eingabe in Großschrift durchgeführt, falls **CAPS** ungleich 0 ist. Bei Namensgleichheit mit vorhandenen Befehlen wird eine Warnung ausgegeben (falls Bit 1 in **SFLAG** auf 0 steht), aber der Befehl trotzdem erstellt. Ist die Variable **>HEAD?** ungleich 0, so wird der Header zum Heap gebracht und **>HEAD?** um eins erniedrigt. In beiden Fällen wird bei Bedarf der Dictionary- oder Heap-Zeiger vorher auf die nächste gerade Adresse gesetzt.

Beispiel:
```
: Constant   ( w -- ; -- w )
   Create ,
   Does>  @ ;
```

CS@ (-- ptr)

Das KK-FORTH arbeitet meistens nur mit einem Speicher von 64KByte. Mittels **DS@** wird ein Pointer auf den Anfang des Codebereiches geliefert.

CURRENT (-- addr) 83

In der USER-Variablen **CURRENT** wird ein Zeiger auf das Offset-Feld des Vokabulars abgelegt, in das alle Neudefinitionen eingetragen werden. Beim Start des KK-FORTH wird hier das Vokabular FORTH eingetragen. Mit DEFINITION wird das aktuelle **CONTEXT** - Vokabular in **CURRENT** übernommen.

D+ (dw1 dw2 -- dw3) 83

Die beiden 32Bit-Werte dw1 und dw2 werden addiert und die Summe dw3 wieder auf dem Datenstack abgelegt. Da im KK-FORTH das Zweierkomplement verwendet wird, kann dieser Befehl sowohl bei Werten mit als auch ohne Vorzeichen verwendet werden. Ein Überlauf des verwendeten Zahlenbereiches wird nicht erkannt, sondern nur der niederwertige 32Bit-Teil des Ergebnisses zurückgeliefert.

D- (dw1 dw2 -- dw3) E83

Der 32Bit-Wert dw2 wird von dw1 abgezogen und die Differenz dw3 wieder auf dem Datenstack abgelegt. Wie auch bei **D+** können sowohl vorzeichenbehaftete als auch vorzeichenlose Werte bearbeitet werden. Eine Überschreitung des Zahlenbereiches wird ignoriert.

D. (d --) E83
Der vorzeichenbehaftete 32Bit-Wert d wird linksbündig mit nachfolgenden Leerzeichen in der aktuellen Zahlenbasis ausgegeben.

D.R (d n --) E83
Der vorzeichenbehaftete 32Bit-Wert d wird linksbündig in einem Feld mit n Zeichen ausgegeben. Es wird dazu die aktuelle Zahlenbasis verwendet. Ist der Wert n negativ oder kleiner als die Länge des Ausgabestrings, so wird n ignoriert und der String ohne zusätzliche Leerzeichen ausgegeben.

D0< (d -- f) 83
Falls der vorzeichenbehaftete 32Bit-Wert d negativ ist, wird ein TRUE-Flag (-1) zurückgeliefert.

D0<> (dw -- f)
Nur wenn der 32Bit-Wert dw ungleich 0 ist, wird ein TRUE-Flag zurückgeliefert.

D0= (dw -- f) 83
Die Gegenfunktion zu D0<> liefert ein TRUE-Flag, falls der 32Bit-Wert dw 0 ist.

D0> (d -- f)
Ein TRUE-Flag wird zurückgeliefert, wenn der vorzeichenbehaftete 32Bit-Wert d positiv und größer als 0 ist.

D2* (dw1 -- dw2)
Der 32Bit-Wertes dw1 wird um eine Stelle nach links geschoben und das freie Bit0 mit 0 aufgefüllt. Diese Funktion kann auch als Verdopplung eines 32Bit-Wertes angesehen werden, solange dabei keine Überschreitung des Darstellungsbereiches auftritt.

D2/ (dw1 -- dw2)
Der 32Bit-Wert dw1 wird um eine Stelle nach rechts geschoben, wobei das Bit31 den alten Wert beibehält. Diese Funktion wird auch arithmetisches Schieben genannt, da ein 32Bit-Wert ohne Veränderung seines Vorzeichens durch zwei geteilt wird.

D< (d1 d2 -- f)
Ein TRUE-Flag wird zurückgeliefert, wenn der 32Bit-Wert d2 größer als d1 ist.

D<> (dw1 dw2 -- f)
Ein TRUE-Flag wird zurückgeliefert, falls die beiden Werte ungleich sind. Der Befehl kann sowohl auf Bitmuster als auch auf vorzeichenbehaftete und vorzeichenlose 32Bit-Werte angewandt werden.

D= (dw1 dw2 -- f)
Die Umkehrung zu **D<>** liefert ein TRUE-Flag, falls beide 32Bit-Werte identisch sind.

D> (d1 d2 --)
Ein TRUE-Flag wird zurückgeliefert, wenn der vorzeichenbehaftete 32Bit-Wert d1 größer als d2 ist.

D>PTR (d -- ptr)

Im KK-FORTH kann auch der zusätzliche Speicher außerhalb des FORTH-Bereiches angesprochen werden. Aus Kompatibilitätsgründen sollte dabei intern eine durchgehende 32Bit-Adressierung gewählt werden. Um nun die tatsächliche Adressierung (z.B. bei PC) zu ermitteln, wird der 32Bit-Wert in einen Pointer umgesetzt. Dabei muß (z.B. beim 8086-FORTH) die Befehlsfolge `ptr>d d>ptr` nicht unbedingt wieder den gleichen Pointer liefern.

D>S (d -- n)

Der 32Bit-Wert d wird vorzeichenrichtig in den 16Bit-Wert n umgewandelt. Da im KK-FORTH bei 32Bit-Werten der höherwertige 16Bit-Teil als oberster Stackeintrag vorliegt ist **D>S** eine **ALIAS** -Definition zu **DROP** und liefert nur dann den richtigen Wert, wenn d im Bereich von -32768 bis 32767 liegt.

D? (addr --)

Analog zum 16Bit-Befehl ? wird aus einer Variablen oder aus einem Datenfeld ein 32Bit-Wert mittels **2@** gelesen und vorzeichenbehaftet mit **D.** ausgegeben.

DABS (d - du)

Der vorzeichenbehaftete 32Bit-Wert d wird in seinen Absolutwert umgewandelt. Bei d=$8000.0000 bleibt der Wert unverändert, da 0-d wieder $8000.0000 ergibt.

DCLEAR (... --)

Alle auf dem Datenstack abgelegten Werte werden durch Zurücksetzten des Datenstackzeigers auf seinen ursprünglichen Wert (siehe **S0**) gelöscht.

DECIMAL (--) 83

Durch Setzten der USER-Variable **BASE** auf den Wert 10 wird die Zahlenbasis für Ein-/Ausgabe von Zahlenstrings auf Dezimal gesetzt.

DEFER (name ; --)

DEFER legt den nachfolgend angegebenen Name als Befehl im Dictionary an. Solange dieser Name noch nicht mittels **IS** auf einen anderen Befehl umgeleitet wurde, löst der Aufruf eine Fehlermeldung ("DEFER not defined") aus. Der Befehl wird meistens dann verwendet, falls ein erst später definierbarer Befehl schon am Anfang benötigt wird. Es können auch statusabhängige Aktionen ohne zeitraubende Fallabfragen realisiert werden.

Beispiel:

```
DEFER Menü   ( Neustart des Menü's )
: menü_error   ( error -- )
   page ." Menüfehler : " hex u. decimal  cr
   Menü ;           ( wieder zum Menü )
...                 ( weitere Definitionen )
: (Menü        ( -- ) ( Hauptroutine ) ... ;
' (Menü  IS  Menü   ( Menü umleiten )
```

DEFINITIONS (--) 83

Das aktuelle **CONTEXT** -Vokabular wird in die USER-Variable **CURRENT** übernommen. Alle nachfolgenden Befehle werden in das von **CURRENT** angegebene Vokabular eingetragen.

Beispiel:

```
Laden eines Assemblers
Vocabulary Assembler   Assembler Definitions
... ( hier folgen die Befehle des Assemblers )
Forth Definitions
... ( hier folgt der FORTH-Teil mit  Code  ... )
```

DEL (--) UV

Der durch **OUTPUT** vektorisierte Ausgabebefehl sorgt für das Zurückstellen des Cursors um eine Position nach Links und dem Löschen des Zeichens unter dem Cursor. Diese Funktion ist nur dann verwendbar, wenn der Cursor nicht schon am Zeilenanfang steht.

DELETE (Name ; --)

Das File mit dem nachfolgend angegebenen Name wird gelöscht. Zur Sicherheit wird vorher das aktuelle File geschlossen und erst danach wieder geöffnet, falls dies möglich ist. Da dieser Befehl nicht mit **IMMEDIATE** gekennzeichnet wurde, ist bei Verwendung von **DELETE** in :-Definitionen darauf zu achten, daß der Filename immer erst bei Ausführung des Befehls geholt wird.

DEPTH (... -- ... n)

Die Anzahl der auf dem Datenstack liegenden 16Bit-Einträge wird ermittelt und auf dem Stack abgelegt. Falls dieser Befehl nach einem Stackunterlauf ausgeführt wird, können auch negative Werte auftreten.

DISC (-- addr) U

Die USER-Variable **DISC** zeigt auf eine mit **UTABLE:** erzeugte Tabelle, in der die Codefeldadressen der zu den einzelnen Diskbefehlen gehörenden Routinen stehen. Dadurch kann mit einem Befehl die Umstellung der im System vordefinierten Routinen (z.B. Fileinterface über RS232) auf eine eigene Fileverwaltung vorgenommen werden.. Beim Start von KK-FORTH oder nach einem Fehler wird durch Aufruf des Befehls **STANDARD-IO** die Schnittstelle neu initialisiert. Durch SAVE wird auch der aktuelle Wert von DISC gesichert und damit dauerhaft umgestellt.

	addr-4	Offset von DISC im USER-Bereich
	addr-2	Länge des Datenfeldes ab addr
DISC -->	addr	CFA der Initialisierungsroutine
	addr+2	CFA der Routine (FILE?
	addr+4	CFA der Routine (FILE_CREATE
	addr+6	CFA der Routine (FILE_DELETE
	addr+8	CFA der Routine (FILE_OPEN
	addr+10	CFA der Routine (FILE_CLOSE
	addr+12	CFA der Routine (FILE_SIZE
	addr+14	CFA der Routine (FILE_POS!
	addr+16	CFA der Routine (FILE_POS@
	addr+18	CFA der Routine (FILE_READ
	addr+20	CFA der Routine (FILE_WRITE
	addr+22	CFA der Routine (FILE_FREE
	addr+24	CFA der Routine (FILE_FIRST
	addr+26	CFA der Routine (FILE_NEXT

DMAX (d1 d2 -- d1 | d2)

Nur der größere der beiden vorzeichenbehafteten 32Bit-Werte verbleibt auf dem Datenstack.

DMIN (d1 d2 -- d1 | d2)

Nur der kleinere der beiden vorzeichenbehafteten 32Bit-Werte verbleibt auf dem Datenstack.

DNEGATE (d n -- d | -d) E83

Falls der 16Bit-Wert n negativ ist, wird der 32Bit-Wert d in sein Zweierkomplement umgewandelt. Ansonsten bleibt d unverändert.

DO (n1 n2 -- ; R: -- addr n3 n4) 83 I,R
(C: -- addr 3)

DO markiert den Anfang einer Programmschleife, die mit **LOOP** oder **+LOOP** abgeschlossen wird. Während des Kompilierens wird die zugehörige Runtime-Routine im Programm abgelegt und mittels **>MARK** eine Vorwärtsreferenz erstellt. Das Flag 4 dient zur Kontrolle der Programmstruktur durch **+LOOP** oder **LOOP**.
Bei Ausführung der Runtimeroutine im Programm werden Anfangswert n2 und Schleifenendwert n1 vom Datenstack entfernt und in einer prozessorspezifischen Weise auf dem Returnstack abgelegt. Deshalb kann innerhalb einer **DO** ... **LOOP** -Schleife auf vorher mit **>R** auf dem Returnstack abgelegte Werte nicht zugegriffen und ein Programm erst nach **UNLOOP** mit **EXIT** verlassen werden.
Beispiel:

```
: Zähler  100 0  DO  i .  LOOP  ;
```

DOES> (-- pfa) 83 I,R
(C: --)

Bei **DOES>** wird der Definitionsteil einer eigenen Datenstruktur abgeschlossen und die Codefeldadresse des zuletzt erzeugten Befehls verändert. Bei Aufruf des so veränderten Befehls wird dessen Parameterfeldadresse (Adresse des ersten Wertes) auf dem Datenstack abgelegt und der nach **DOES>** folgende Programmteil des Definitionswortes ausgeführt. Dadurch können aufwendige Datenstrukturen ohne Hilfe von Assemblerroutinen realisiert werden.
Beispiel:

```
: Feld:   ( Definition eines Feldes mit 16Bit-Werte )
   Create ( x y -- )      ( Definitionsteil )
    over ,  *  2* allot  ( X-Länge merken, Allot )
   Does>  ( x y addr -- ) ( Ausführung bei Aufruf )
    >r    r@ @ * 2*  +   ( y * X-Länge + x )
    r> 2+            + ; ( + Offset )
3 4  Feld: Matrix  ( Matrix definieren )
1 1  Matrix        ( Liefert Adresse von x=1, y=1 )
3 2  Matrix        ( Liefert Adresse von x=3, y=2 )
```

DP (-- addr) U

Die USER-Variable **DP** enthält die aktuelle Endadresse des Dictionary. Der Wert wird durch **HERE** ausgelesen und zum Datenstack gebracht. Im KK-FORTH wird der Wert nur selten direkt, sondern meisten mittels **ALLOT** verändert.

DPL (-- addr) U

In der USER-Variablen **DPL** wird ein Zähler mitgeführt, der nur bei der Interpretation der Zahleneingabe durch **CONVERT** , **>NUMBER** oder **NUMBER?** verändert wird. Bei der Eingabe von Zahlen wird zur Erkennung von 32Bit-Werten ein Punkt an beliebiger Stelle des Zahlenstrings eingefügt. Nach **NUMBER?** enthält **DPL** die Anzahl der Ziffern nach einem Punkt oder den Wert -1, falls es sich um eine 16Bit-Zahl handelt. Andere negative Werte in **DPL** können zur Kennung eigener Datentypen verwendet werden.

DROP (w --) 83

Der oberste 16Bit-Wert des Datenstacks wird entfernt.

DS@ (-- ptr)

Die Anfangsadresse des Datenbereiches wird durch **DS@** zurückgeliefert. Normalerweise sind Programm, Daten und Stack im gleichen Segment und liefern deshalb den gleichen Pointer.

DU2/ (du1 -- du2)

Wie bei **D2/** wird der 32Bit-Wert dw1 um ein Bit nach rechts geschoben. Das freie Bit31

wird aber mit 0 aufgefüllt, was einer Division eines vorzeichenlosen 32Bit-Wertes durch zwei entspricht.

DU< (du1 du2 -- f)
Ein TRUE-Flag wird auf dem Datenstack abgelegt, wenn der vorzeichenlose 32Bit-Wert du1 kleiner als du2 ist.

DU> (du1 du2 -- f)
Das Flag f ist wahr (-1), wenn der vorzeichenlose 32Bit-Wert du1 größer als du2 ist.

DUMAX (du1 du2 -- du1 | du2)
Der größere der beiden vorzeichenlosen 32Bit-Werte verbleibt auf dem Datenstack.

DUMIN (du1 du2 -- du1 | du2)
Der kleinere der beiden vorzeichenlosen 32Bit-Werte verbleibt auf dem Datenstack.

DUMP (addr len --) E83
Für die Analyse eines Datenfeldes kann der Speicherbereich als Hex-Dump mit nachfolgender ASCII-Ausgabe angezeigt werden. Nach der Ausgabe der Adresse werden jeweils 16 Bytes als zweistellige Hex-Zahl und anschließend als ASCII-Zeichen ausgegeben. Falls das Zeichen nicht ausgebbar ist (z.B. ASCII-Werte unter 32), wird nur ein Punkt angezeigt. Die Ausgabe kann mit beliebiger Taste gestoppt und mit #BRK-Taste abgebrochen werden.

DUP (w -- w w) 83
Der oberste 16Bit-Wert wird dupliziert.

DWITHIN (dw1 dw2 dw3 -- f)
Ein TRUE-Flag wird auf dem Datenstack abgelegt, wenn die Differenz zwischen dw1 und dw2 kleiner als die Differenz wischen dw1 und dw3 ist. **DWITHIN** kann sowohl bei vorzeichenlosen als auch bei vorzeichenbehaftenen 32Bit-Werten verwendet werden, da der Test mit **DU<** durchgeführt wird.

EDITSTRING (addr maxlen pos len -- pos2 len2) UV
Ein sehr mächtiger, ebenfalls über **INPUT** vektorisierter Befehl wird zum Editieren der Eingaben verwendet. Die maximale Länge des Strings wird in der USER-Variablen **SPAN** gespeichert. Danach erfolgt die Ausgabe des Strings (Länge len) ab der aktuellen Cursorposition und Setzten des Cursors an die Position pos. Danach kann der Text mit folgenden Tasten bearbeitet werden:

<- oder CTRL+S	Cursor nach links
-> oder CTRL+D	Cursor nach rechts
Backspace	Zeichen vor dem Cursor löschen
DEL oder CTRL+G	Zeichen unter dem Cursor löschen
Zeichen	Einfügen des Zeichens beim Cursor
ENTER	Abschluß der Eingabe

Falls das Bit 2 von **SFLAG** gelöscht ist, dient der Befehl zur Eingabe einer Befehlszeile. Deshalb kann mit der ESC-Taste oder Cursor hoch die letzten Eingabezeilen zurückgeholt und editiert werden. Danach wird die neue Eingabezeile wieder in diesen Zeilenpuffer gebracht.

ELSE (--) 83 I,R
(C: addr1 1 -- addr2 -1)

ELSE leitet den Falsch-Teil einer **IF**...**ELSE**...**THEN** -Struktur ein. Beim Kompilieren wird ein unbedingter Sprung auf das Ende der Struktur in den Programmcode eingefügt und die Zieladresse des **IF** -Sprunges korrigiert. Als Kontrolle des richtigen Strukturaufbaues wird das von **IF** abgelegte Flag mit Wert 1 verwendet und in -1 gewandelt.

EMIT (char --) UV,83

Der zum **OUTPUT** -Vektor gehörender Befehl **EMIT** gibt den auf dem Stack liegende Zeichencode aus. Dazu wird meistens der niederwertige 8-Bit-Teil des Stackwertes verwendet. Deshalb können auch Zeichencodes unter ASCII-Wert 32 ausgegeben werden.

EMIT? (-- f) UV

EMIT? dient zur Überprüfung der Bereitschaft des aktuellen Ausgabemediums. Falls das Flag gleich 0 ist, warten alle Ausgabebefehle solange, bis wieder ein Zeichen ausgegeben werden kann. **EMIT?** gehört ebenfalls zum **OUTPUT** -Vektor und wird z.B. bei Druckerspooler im Hintergrundbetrieb oder für Terminalprogramme verwendet.

EMPTY (--)

Nach Ausführung von **EMPTY** sind alle Befehle, die seit dem letzten **SAVE** oder SAVESYSTEM eingegeben wurden, wieder gelöscht. Zusätzlich wird der Heap-, Variablen- und USER-Bereich wieder freigegeben.
Meistens wird der Befehl bei mehrfacher Kompilierung von Programmteilen zur schnellen Zurückstellung auf den alten Systemzustand verwendet.

EMPTY-BUFFERS (--) 83

Der im FORTH83-Standard vorgegebene Befehl des Block-Diskinterfaces dient zur Freigabe des im KK-FORTH verwendeten Diskspeichers. Dies geschieht durch das Einschreiben der nicht verwendbaren Blocknummer 32767 in die erste Zelle ab **FIRST** . Da vorher der Puffer nicht zurückgeschrieben wird, ist ein geänderter Inhalt verloren. Sollte vorher eine Sicherung durchgeführt werden, so ist statt **EMPTY-BUFFERS** der Befehl **FLUSH** anzuwenden.

ENDCASE (n --) I,R
(C: 5 ... 5 --)

Der Abschluß einer CASE-Struktur kompiliert **DROP** und korrigiert dann die Adressen aller durch **ENDOF** abgelegten Sprünge. Falls eine der OF-Bedingungen zutrifft, wird das Programm nach **ENDOF** sofort bei **ENDCASE** fortgesetzt. Ansonsten wird der Suchwert vom Datenstack entfernt.

ENDOF (--) I,R
(C: addr1 -5 -- addr2 -2 5)

Das Ende eines OF-Zweiges einer CASE-Struktur kompiliert einen unbedingten Sprung und korrigiert die Adresse des OF-Sprunges.

ERASE (addr len --)

ERASE füllt einen Speicherbereich von len Bytes ab Adresse addr mit Wert 0.

ERROR (0 | error | csa -1 | csa $7fff --)

Einer der wichtigsten Befehle der im KK-FORTH neu strukturierten Fehlerbehandlung ist **ERROR** . Alle Systembefehle verwenden für den Aufruf der Fehlerbehandlungsroutine diesen Befehl. **ERROR** erwartet auf dem Datenstack entweder das Flag 0, die

Fehlernummer error oder das Flag $7FFF (oder $FFFF) mit Counted-String-Adresse der Fehlermeldung. Ist das Flag auf dem Stack 0, so wird der aktuelle Befehl nicht unterbrochen. Bei allen Werten ungleich 0 wird die Routine aufgerufen, auf welche USER-Variable **ERRORHANDLER** zeigt. Die Fehlernummern $7F00 bis $7FFF sind für den KKF-Kern reserviert. Fehlernummer $0001 bis $00FF werden vom Betriebssystem des verwendeten Rechners geliefert.

ERROR" (f --) I
(C: --)

ERROR" dient zur Ausgabe einer Fehlermeldung und zum Restart des FORTH, falls das auf dem Datenstack abgelegte Flag ungleich 0 ist. Anders als bei **ABORT"** wird aber der Datenstack vor der Fehlerausgabe nicht gelöscht.

ERRORHANDLER (-- addr)

Die USER-Variable **ERRORHANDLER** zeigt auf die von **ERROR** aufgerufene Fehlerbehandlungsroutine. Im KK-FORTH zeigt diese Variable auf **(ERRORHANDLER** . Es wird dann bei einem Fehler der Eingabestring und der entsprechende Fehlertext ausgegeben.

ERRORTEXT@ (error | csa $7fff | csa $ffff -- addr len) DX

Der DEFER-Befehl **ERRORTEXT@** zeigt auf die unmittelbar danach folgende versteckte Routine. Diese Routiene liefert Länge und Adresse der Fehlermeldung. Dazu wird entweder die angegebene Counted-String-Adresse ausgewertet oder den zu der Fehlernummer gehörenden String aus der Tabelle ab **>ERRORTEXT** zurückgeliefert. Ist die Fehlernummer nicht vohanden, so wird der String "Error $xxxx" zurückgeliefert.

EVALUATE (addr len --) ANSI

Der im ANSI-FORTH forgeschlagene Befehl **EVALUATE** interpretiert oder kompiliert den angegebenen String. Dazu werden die USER-Variablen **BLK** , **>IN** , **#TIB** und **>TIB** gespeichert und entsprechend verändert.

EXECUTE (cfa --) 83

Auf dem Datenstack wird die Codefeldadresse des gewünschten Befehls erwartet. **EXECUTE** entfernt die Adresse vom Datenstack und führt diese Routine aus. Es gibt bis auf die Ausführungszeiten und dem fehlenden Test auf RESTRICT-Bit keinen funktionellen Unterschied zwischen den zwei nachfolgenden Befehlssequenzen:

```
... Name ...
... ['] Name EXECUTE ...
```

EXIT (--) (R: addr --) 83 R

Der Befehl **EXIT** dient zum Verlassen des aktuellen Wortes und Fortsetzung des Programmes im aufrufenden Wort. **EXIT** wird automatisch durch **;** kompiliert, kann aber z.B. bei Fallunterscheidungen zusätzlich angegeben werden. Da die Rücksprungadresse als oberster Returnstackeintrag erwartet wird, darf **EXIT** ohne spezielle Behandlung nicht in **DO**...**LOOP** -Schleifen verwendet werden (siehe **UNLOOP**). Um einen Systemabsturz im Interpretermodus zu verhindern, ist der Befehl als **RESTRICT** gekennzeichnet worden und darf deshalb nur in :-Definitionen verwendet werden.

EXPECT (addr len --) UV,83

Der zum INPUT-Vektor gehörende Befehl **EXPECT** erwartet die Speicheradresse und die maximale Länge des einzugebenden Strings. Anschließend wird auf die Eingabe der Zeichen gewartet. Da für die Eingabe der Befehl **EDITSTRING** verwendet wird, können auch alle dort beschriebenen Tasten verwendet werden.

Erst nach Betätigung der ENTER-Taste wird die eingegebene Zeilenlänge in der USER-Variable SPAN gespeichert.

FALSE (-- 0) Con

Die Konstante **FALSE** liefert den bei allen Vergleichen gelieferten Wert, wenn die Bedingung nicht erfüllt wird. Dieser Wert 0 ist auch der einzige Wert, bei dem der bedingte Sprung **?BRANCH** ausgeführt wird.

FILE-FCB (-- addr) V

Die Variable **FILE-FCB** enthält einen Zeiger auf die FCB-Adresse des aktuellen Files. Falls das File mit **(OPEN** geöffnet wurde, kann der zugehörige String auch zerstört sein und dadurch bei **INCLUDE** oder **LOADFROM** einen Fehler verursachen.

FILE-ID (-- addr) V

In der Variablen **FILE-ID** wird die Handlenummer des aktuellen Screenfiles gespeichert. Bei allen Werten ungleich 0 wird bei **?OPEN** kein Fehler ausgegeben.

FILE-LINK (-- addr) SV

Die Variable **FILE-LINK** beinhaltet einen Vektor auf den Verkettungszeiger des zuletzt geöffneten Files. Beim Anlegen neuer Filenamen wird die Verkettung fortgesetzt und bei **REMOVE** korrigiert. Deshalb kann mit **FILES** die Liste aller verfügbarer Filenamen ausgegeben werden.

FILE. (--)

Falls ein File geöffnet ist, wird der Name des Files ausgegeben. Siehe dazu auch **FILE-FCB** .

FILES (--)

Die Liste aller im KK-FORTH verwendeter Filenamen wird ausgegeben. Siehe dazu auch **FILE-LINK** .

FILL (addr len char --) 83

Ein Speicherbereich von len Bytes ab der Adresse addr wird mit dem Byte-Wert char gefüllt.

FIND (csa -- cfa f | csa 0) DX,83

Der Suchbefehl **FIND** zeigt auf die unmittelbar danach folgende, versteckte Routine und erwartet auf dem Datenstack die Adresse des gesuchten Strings. Falls das Flag CAPS einen Wert ungleich 0 besitzt, wird dieser String in Großschrift umgewandelt. Danach werden alle CONTEXT-Vokabulare nach diesem Befehlsname abgesucht. Wird der Befehl gefunden, so liefert **FIND** die Codefeldadresse des Befehls und ein Flag zur Art des Befehls:

Betrag von f = $01	Standardbefehle
Betrag von f = $02	Befehl ist Restrict
f negativ	Befehl ist Immediate

Ansonsten bleibt die Counted-String-Adresse und zusätzlich ein FALSE-Flag auf dem Datenstack.

FIRST (-- sys)

FIRST liefert die erste von der Fileverwaltung verwendete Speicheradresse. Sie ist vom verwendeten System und von der Anzahl der für den Diskpuffer reservierten Adressen abhängig. Normalerweise wird im KK-FORTH nur ein Diskpuffer verwendet und die erste Zelle für die Blocknummer der nachfolgenden 1024 Bytes verwendet. Bit15 markiert dabei, daß dieser Block mit **UPDATE** für die Speicherung markiert wurde.

FLIP ($xxyy -- $yyxx)

Vertauschung der höherwertigen 8 Bit mit den niederwertigen 8 Bit. Der Befehl kann z.B. zur schnellen Multiplikation eines 8-Bit-Wertes mit 256 oder zur Aufteilung eines 16Bit-Wertes in zwei 8-Bit-Werte verwendet werden.

FLUSH (--) 83

Wie bei **EMPTY-BUFFERS** gibt der Befehl **FLUSH** alle vom KK-FORTH verwendeten Diskpuffer wieder frei. Falls Screens mit **UPDATE** markiert worden sind, werden sie vorher zurückgeschrieben.

FOR (n --) I,R
(C: -- addr 4)

Die von Charles Moore speziell für den FORTH-Chip NC4000 entwickelte Befehlsstruktur **FOR** ... **NEXT** erzeugt eine schnelle Scheife mit n+1 Wiederholungen. Das durch **FOR** kompilierte **>R** bringt den Schleifenwert zum Returnstack und kann von dort mit **R@** abgefragt werden. Die während der Kompilierung ablegten Adresse addr und der Wert 4 dient zur Strukturkontrolle und Korrektur des Rücksprung bei **NEXT** .

FORGET (name; --) 83

FORGET erwartet den Befehlsnamen, der vergessen werden soll. Alle danach definierten Befehle und Vokabulare werden durch den in **FORGET** aufgerufenen Befehl **REMOVE** ebenfalls entfernt. In der Suchreihenfolge ist danach nur noch das Vokabular **FORTH** zu finden. Folgende Fehler sind bei `FORGET Name` möglich:

- Der Name wurde nicht gefunden
- Dictionaryzeiger zeigt in den Heap
- Der Befehl ist geschützt

FORTH (--) Voc,83

In dem Vokabular **FORTH** sind alle FORTH83-Befehle definiert. Beim Start des KK-FORTH oder nach dem Vergessen einzelner Befehl wird durch **ONLYFORTH** das **FORTH** als einziges zu durchsuchendes Vokabular sowohl in den festen als auch in den variablen Teil von **CONTEXT** eingetragen.

H, (w --)

Ein 16Bit-Wert wird in den Heap gebracht. Dazu wird zuerst der Heap um eine Zelle (meist 2 Bytes) nach unten erweitert und dann w vor dem Heap abgelegt. Meistens muß dazu der Variablenbereich verschoben werden.

HALIGN (--)

Ähnlich **ALIGN** führt **HALIGN** eine Korrektur des Heap-Zeigers aus, falls der verwendete Prozessor keinen Wortzugriff auf gerade Adressen durchführen kann.

HALLOT (n --)

Die Anfangsadresse des Heap's wird um n Bytes verändert. Positive Werte vergrößern den Heap durch Herabsetzen der Anfangsadresse. Negative Werte setzen die Anfangsadresse des Heap wieder herauf. Da im KK-FORTH der Heap überhalb den Variablen liegt, muß bei **HALLOT** der gesamte Variablenbereich verschoben werden. Interruptroutinen, die Daten im Variablenbereich ablegen, sind deshalb während der Kompilierung zu deaktivieren.

HC, (char --)

Das niederwertigen Byte aus char wird zum Heap gebracht. Dazu wird der Heap um ein Zeichen erniedrigt und deshalb meistens auch der Variablenbereich verschoben.

HCLEAR (--)

Alle auf dem Heap abgelegten Namen werden aus der Befehlsliste entfernt und der Heap vollständig gelöscht. Da auch der Speicher wieder für andere Zwecke freigegeben wird, dürfen die verbleibenden Befehle keine Adressen oder Befehl des Heap's kompiliert haben. Bei dem oft nur temporär auf dem Heap abgelegten Assembler ist dies der Fall, da die kompilierten Befehle nur zur Erzeugung der Assemblerroutine herangezogen werden.
Da **HCLEAR** den Befehl **REMOVE** verwendet, gelten alle dort angegebenen Aktionen und Fehlermöglichkeiten.

HDP (--) SV

Die Systemvariable **HDP** enthält die Adresse des aktuellen Heapanfangs. Da evtl. auch temporäre Programme im Heap abgelegt werden, bleibt die Speicheradresse des Heap meistens unverändert. Die einzige Ausnahme ist die Vergrößerung des Taskbereiches zur Generierung weiterer Tasks. Jedoch muß dazu vorher der Heap gelöscht oder die Verkettungszeiger korrigiert werden.

HEADERS (--)

Der Gegenbefehl zu **-HEADERS** setzt die Variable **>HEAD?** wieder auf den Wert 0 und sorgt damit dafür, daß alle nachfolgenden Befehlsheader wieder im Dictionary abgelegt werden.

HEAP (-- addr)

Die unterste durch den Heap belegte Adresse liefert **HEAP** . Da der Bereich in Richtung niedrigerer Adresse wächst, ist addr gleichzeitig der zuletzt für den Heap reservierte Speicher. **HEAP** ermittelt die Adresse aus der Systemvariable **HDP** .

HEAP? (cfa -- f)

Das Flag f wird wahr, wenn die angegebene Adresse im Heap liegt.

HERE (-- here) 83

HERE liefert durch Auslesen der USER-Variable **DP** die Adresse des ersten Bytes oberhalb des FORTH-Dictinary. Beim Erzeugen neuer Befehle wird sowohl der Header als auch die einzelnen Befehle immer ab **HERE** abgelegt und der Zeiger **DP** entsprechend erhöht.

HEX (--)

Durch das Setzen der USER-Variable **BASE** auf den Wert 16 wird die hexadezimale Zahlenein-/ausgabe aktiviert. Bei allen Zahlenein-/ausgaben werden dann die Buchstaben A bis F für die "Ziffern" 10 bis 15 verwendet.

HIDE (--)

Innerhalb einer :-Definitionen soll der aktuelle Befehl noch nicht aufrufbar sein. Dazu setzt **HIDE** den Zeiger des in **CURRENT** angegebenen Vokabulars auf das vorhergehende Wort und versteckt dadurch den zuletzt mit **CREATE** definierten Befehl. Da aber **CREATE** die Linkfeldadresse des zuletzt erzeugten Befehls in der Variablen LAST speichert, kann dann bei Abschluß der :-Definition mittels **;** der Befehl **REVEAL** die alte Suchreihenfolge wieder herstellen.
Eine Fehlermeldung wird ausgegeben, falls bei Aufruf von **HIDE** die Variable **LAST** auf 0 steht und deshalb kein Befehl zu verstecken ist.

HLD (-- addr) UV

Die USER-Variable **HLD** enthält den Zeiger auf das zuletzt mit **HOLD** gespeichert Zeichen. Es wird im KK-FORTH bei allen Zahlenstring-Erzeugungen verwendet und zeigt

dabei immer in den Speicherbereich um **PAD** . **HLD** wird durch **<#** auf die Adresse von **PAD** gesetzt und vor der Speicherung eines Zeichens erniedrigt. Selbst wenn die Eingabe im Interpretermodus erfolgt, können mit **<# # #S SIGN** und **HOLD** noch Strings mit einer Gesamtlänge von 52 (= 84-32) Bytes erzeugt und ausgegeben werden. Da die mit " eingegebenen Strings erst ab **PAD** abgelegt werden, verändern sie nicht den Zahlenstring.

HLEN (-- sys) SV

Die Systemvariable **HLEN** enthält die Gesamtlänge des aktuellen Heapbereiches.

HOLD (char --) 83

Der Zeiger in der USER-Variable **HLD** wird erniedrigt und der auf dem Datenstack befindliche ASCII-Wert an dieser Adresse abgelegt. **HOLD** wird von den Befehlen **#** , **#S** und **SIGN** zur Erzeugung eines Zahlenstrings verwendet. Da der Zahlenstring von Hinten nach Vorne erzeugt wird, muß das letzte Zeichen zuerst angegeben werden.

I (-- w) 83

In einer **DO** ... **LOOP** -Schleife kann der aktuelle Index der Schleife mit **I** auf den Datenstack gebracht werden. Da die dazu benötigten Schleifenwerte ab einem festen Offset erwartet werden, dürfen vor **I** keine zusätzlichen Parameter auf dem Returnstack abgelegt werden.

I' (-- w) ANSI

Mit **I'** wird der Endwert der innersten Schleife auf den Datenstack gebracht. Wie bei **I** darf auch hier kein anderer Wert auf den Returnstack liegen.

IDENT (--)

Beim Start des KK-FORTH wird nach der Initialisierung, aber noch vor dem Start des Interpreters, eine Einschaltmeldung ausgegeben. Diese durch **IDENT** ausgegebene Meldung hat folgendes Format:

```
KK-FORTH_PC V1.2/0
- (C) Klaus Kohl -
```

Da **IDENT** nur durch die Einbindung in **'ABORT** ausgeführt wird, kann sie vom Anwender durch eigene Meldungen oder sogar eigene Programme ersetzt werden.

IF (f --) 83 I,R
(C: -- addr 1)

Anwendung: `... f IF ... ELSE ... THEN ...`

Der auf **IF** folgende Programmteil bis zum **ELSE** oder (bei fehlendem **ELSE**) **THEN** wird dann ausgeführt, wenn das auf dem Datenstack liegende Flag f einen Wert ungleich Null besitzt. IF darf nur in :-Definitionen verwendet werden und kompiliert einen bedingten Sprung. Auf dem Stack werden zusätzlich zu der Adresse des Sprungs noch ein Kontrollflag abgelegt.

IMMEDIATE (--) 83

Der immer nach dem Abschluß einer Definition verwendete Befehl IMMEDIATE setzt im Header des zuletzt generierten Befehls ein Bit. Dieses Bit veranlaßt dann den Kompiler, selbst innerhalb einer :-Definitionen den neuen Befehl nicht zu kompiliert, sondern sofort auszuführen. Nur dadurch sind überhaupt Kontrollstrukturen mit Test der Parameter möglich.

Beispiel: `: IF postpone ?branch >mark 1 ; IMMEDIATE`

INCLUDE (name ; --)

Falls noch ein File offen ist, wird die Filekennung (FCB) gespeichert und das File geschlossen. Danach wird das File mit dem nachfolgenden Namen geöffnet, Screen 1 geladen und danach das File wieder geschlossen. Die gespeicherte Filekennung dient dann wieder zum Öffnen des alten Files. Falls das vorher verwendete File ohne Erzeugung eines Befehlsnamen geöffnet wurde, kommt es zu einem Fehler, wenn der Speicherbereich der angegebenen Filekennung überschrieben wurde.
Falls weitere Screens aus dem angegebenen File geladen werden müssen, ist ein entsprechender LOAD -Befehl im Screen 1 unterzubringen. Der INCLUDE -Befehl gibt selbst während seiner Ausführung noch Meldungen aus und legt den Filename auch als neuen Befehl an.
Beispiel:
```
Include : Filename
  >>>>>                    ( für jeden Block ein Pfeil )
End-Include : Filename
```

INDENT (--)

Speziell für die formatierte Ausgabe der Blocknummer (oder ">") und INCLUDE-Meldungen beim Laden von Files ist für die richtige Einrückung der Befehl **INDENT** eingeführt worden. **INDENT** gibt die in der Variablen **INDENT?** gespeicherte Anzahl von Leerzeichen aus. **INDENT** sollte bei Kommentarausgaben während des Laden von Files immer nach **CR** eingefügt werden.
Beispiel:
```
cr indent .( Assembler geladen)
```

INDENT? (-- addr) V

Die Variable **INDENT?** enthält die Anzahl der Leerzeichen, die von **INDENT** ausgegeben werden. **INDENT?** wird von QUIT auf 0 gesetzt und bei jedem **LOADFROM** (oder **INCLUDE**) um zwei erhöht und danach wieder erniedrigt.

INDIRECT (--)

Wie bei **IMMEDIATE** wird auch der Befehl **INDIRECT** erst nach der Erstellung eines Befehls angegeben und setzt ebenfalls ein Bit im Header dieses Befehls. Dieses gesetzte Bit veranlaßt dann das FORTH, bei der Erkennung des Befehls die Codefeldadresse nicht direkt zu verwenden. Statt dessen wird der Inhalt der CFA als Zeiger auf die tatsächliche CFA herangezogen. Meistens wird **INDIRECT** bei **ALIAS** -Definitionen verwendet, weil das Programm dadurch kompakter und auch schneller wird.

INPUT (-- addr) U

Die USER-Variable **INPUT** zeigt auf eine mit **UTABLE:** erzeugte Tabelle mit den Codefeldadressen aller gerade verwendeten Eingabebefehle. Die Tabelle hat folgenden Aufbau:

	addr-4	Offset von INPUT im USER-Bereich
	addr-2	Länge des Datenfeldes ab addr
INPUT -->	addr	CFA der Initialisierungsroutine
	addr+2	CFA der Routine KEY?
	addr+4	CFA der Routine KEY
	addr+6	CFA der Routine STRING
	addr+8	CFA der Routine EDITSTRING
	addr+10	CFA der Routine QUERY

INTERPRET (--)

Der Befehl **INTERPRET** interpretiert oder kompiliert die aktuelle Eingabezeile. Dabei verwendet es den Befehl **WORD** zum Holen eines Befehlsnamen. Bei **BLK** = 0 wird die Eingabezeile aus **TIB** mit Länge **#TIB** verwendet. Bei Werten ungleich 0 wird der entsprechnde Block mit 1024 Zeichen interpretiert oder kompiliert. **>IN** wird als aktueller Zeiger verwendet und ist deshalb am Anfang auf 0 zu setzen.
Die Unterscheidung, ob ein Befehl interpretiert oder kompiliert wird, erfolgt durch den

DEFER-Vektor **PARSER** . Je nach Zustand zeigt es auf **INTERPRETER** oder **COMPILER** . Diese beiden Wörter sind nicht direkt verfügbar, weil sie automatisch durch die im FORTH zu verwendenden Befehle **[** und **]** gesetzt werden. Dabei wird auch die USER-Variable **STATE** entsprechend dem Modus (0 für Interpreter, -1 für Kompiler) gesetzt.
Nach jedem Befehl wird auch noch der Datenstack mit **?STACK** untersucht. Sowohl Datenstacküber- und -unterläufe als auch ein nicht mehr ausreichender Speicher für das Dictionary wird dabei erkannt und beendet durch eine Fehlerausgabe **INTERPRET** .

IS (Name ; cfa --) I
DEFER-Befehle sind nach ihrer Definition mit **NODEFER** vorbelegt. Deshalb liefern sie beim Aufruf eine entsprechende Fehlermeldung. Mit `' name1 IS name2` wird die mit ' ermittelte Codefeldadresse des Befehls name1 in den nach IS folgenden Befehl name2 übernommen. Eine Fehlermeldung wird ausgegeben, falls es sich bei name2 nicht um ein DEFER-Befehl handelt.
IS kann auch in :-Definitionen verwendet werden, da dann automatisch eine entsprechende Runtimeroutine und die Codefeldadresse des nachfolgenden Befehls abgespeichert wird. Es muß dann aber zur Ermittlung der CFA statt ' der Befehl **[']** verwendet werden.

J (-- n) 83
In zwei ineinander geschachtelten **DO** ... **LOOP** -Schleifen kann der aktuelle Index der äußersten Schleife mit J auf den Datenstack gebracht werden. Da die dazu benötigten Schleifenwerte ab einem festen Offset erwartet werden, sind keine weiteren Parameter auf dem Returnstack zulässig.

J' (-- n) ANSI
Mit **J'** wird der Endwert der nächst äußersten Schleife auf den Datenstack gebracht. Wie bei J darf auch hier kein anderer Parameter auf dem Returnstack sein.

KEY (-- char) UV,83
Der über **INPUT** vektorisierte Eingabefehl wartet auf das nächste Zeichen und bringt den ASCII-Wert des Zeichens zum Datenstack.

KEY? (-- f) UV
Der über **INPUT** vektorisierte Eingabebefehl liefert ein Flag, ob mindestens ein Zeichen vom aktuellen Eingabemedium verfügbar ist.

L! (w ptr --) X
Der 16Bit-Wert w wird ab der Speicheradresse ptr abgelegt. Sowohl die Reihenfolge der Bytes im Speicher als auch die Bedeutung der 32Bit-Adresse ptr ist Systemabhängig.

L2! (dw ptr --) X
Der 32Bit-Wert dw wird ab der Speicheradresse ptr abgelegt. Sowohl die Reihenfolge der Bytes im Speicher als auch die Bedeutung der 32Bit-Adresse ptr ist Systemabhängig.

L2@ (ptr -- dw) X
Die Umkehrung des Befehls **L2!** holt den abgelegten Wert wieder zum Datenstack.

L>NAME (lfa -- nfa) E83
Aus der Linkfeldadresse wird die Namensfeldadresse eines Befehls ermittelt. Normalerweise folgt die NFA nach der LFA. Trotzdem sollte dieser Befehl verwendet werden, weil nur

dadurch eine Veränderung der Dictionarystruktur ohne Auswirkung auf das Programm durchgeführt werden kann.

L@ (ptr -- w) X
Die Umkehrung des Befehls **L!** bringt den an der 32Bit-Adresse ptr abgelegten 16Bit-Wert w wieder zum Datenstack.

LABEL (name ; --) I
Die aktuelle Dictionaryposition wird mit **>LABEL** ohne Veränderung als Konstante vollständig auf dem Heap abgelegt.

LAST (-- addr) V
Die Variable **LAST** enthält die Linkfeldadresse des letzten mit **CREATE** definierten Befehls. **LAST** wird sowohl von **HIDE** und **REVEAL** zum Verstecken eines Befehls als auch von **IMMEDIATE** , **RESTRICT** und **INDIRECT** zur Markierung bestimmter Funktionen des letzten Befehl verwendet.

LC! (char ptr --) X
Der 8Bit-Wert char wird ab der 32Bit-Adresse ptr abgelegt.

LC@ (ptr -- char) X
Die Umkehrung des Befehls **LC!** bringt den an der 32Bit-Adresse ptr abgelegten 8Bit-Wert char wieder zum Datenstack.

LCMOVE (ptr1 ptr2 len --) X
Analog dem Befehl **CMOVE** können bis zu 65535 Bytes ab Adresse ptr1 nach ptr2 gebracht werden. Bei einigen Versionen des KK-FORTH (z.B. PC) müssen bestimmte Beschränkungen bei der Adressierung beachtet werden.

LCMOVE> (ptr1 ptr2 len --) X
Wie bei **LCMOVE** werden len Bytes von ptr1 nach ptr2 gebracht. Jedoch beginnt der Befehl mit der höchsten Adresse und erlaubt dadurch bei überlappenden Speicherbereiche eine Verschiebung zu niedrigeren Adressen.

LDUMP (seg addr len --) X
Wie bei **DUMP** werden mindestens len Bytes sowohl Hexadezimal als auch in ASCII-Form ausgegeben, wobei eine 32Bit-Adresse anzugeben ist.

LEAVE (--) R
Der nur innerhalb von **DO** ... **LOOP** -Schleifen zulässige Befehl entfernt alle Schleifenparameter vom Returnstack und setzt das Programm hinter der Schleife fort.

LFILL (ptr len char --) X
Analog dem Befehl **FILL** wird ein Speicherbereich von len Bytes ab der 32Bit-Adresse ptr mit dem Zeichen char gefüllt. Bei einigen KKF-Versionen sind Beschränkungen für die Parameter zu beachten.

LIMIT (-- sys)
Die systemabhängige Konstante **LIMIT** gibt die Speicheradresse des ersten nicht mehr von

KK-FORTH verwendeten Bytes an. Bei einigen EMUF-Versionen des KK-FORTH werden noch Interrupt-Tabellen außerhalb des FORTH-Arbeitsbereiches angelegt.

LINK> (lfa -- cfa) E83

Aus der Linkfeldadresse eines Befehls wird durch **LINK>** die Codefeldadresse ermittelt. Der Befehl berücksichtigt auch das Indirect-Bit des Headers und liefert deshalb bei derartig markierten Namen die tatsächliche Codefeldadresse.

LIST (u --)

Um auch ohne Editor ein Programm ansehen zu können, wurde LIST definiert. Es erwartet auf dem Stack die Nummer des gewünschen Blockes und gibt diesen formatiert aus. Dabei wird in der Überschrift die Nummer des Blockes wiederholt und vor jeder Programmzeile die Nummer (0..15) ausgegeben. Es werden immer 64 Zeichen pro Zeile ausgegeben, wobei nicht druckbare Zeichen als Punkt dargestellt werden. Die Ausgabe kann mit beliebiger Taste gestoppt oder mit #BRK-Taste abgebrochen werden.

LITERAL (w --) 83 I,R

Der für die Kompilierung von Zahlenwerten in :-Definitionen gedachte Befehl erwartet auf dem Datenstack einen 16Bit-Wert und kompiliert ihn als Inline-Literal. Da der Doppelpunkt und die Kontrollstrukturen der Datenstack ebenfalls verwenden, muß dieser Wert unmittelbar zuvor in eckigen Klammern errechnet werden. Meist wird **LITERAL** verwendet, um einen während des Programmablauf unveränderlichen Wert ohne zusätzliche Konstantendefinition und ohne ständige Neuberechnung verfügbar zu haben.

Beispiel:
```
$003 Constant X-Len
$004 Constant Y-Len
: XY+   ( addr1 -- addr2 ) ( Offset schnell addieren )
   [ x-len y-len + ] Literal  +  ;
```

LMOVE (ptr1 ptr2 len --) X

Durch **LMOVE** wird ein Speicherbereich von len Bytes ab ptr1 nach ptr2 kopiert. Dabei wird durch geeignete Wahl von **LCMOVE** oder **LCMOVE>** selbst bei überlappenden Speicherbereiche der ursprüngliche Inhalt übernommen.
Auch hier sind bei bestimmten Prozessoren Restriktionen bei Adresse und Länge zu beachten.

LOAD (u --) 83

Vom aktuellen Screenfile wird der angegebene Screen in den Speicher geladen und interpretiert. Dabei kann wegen der Verwendung der Nummer 0 als Kennung des TIB die Screennummer 0 nicht geladen werden und wird deshalb nur zu Kommentarzwecke herangezogen. Ein Fehler wird ausgegeben, wenn kein File geöffnet ist oder die Blocknummer die Filegröße überschreitet.

LOADFROM (name ; u --)

Der Name des aktuellen Files wird gespeichert und das File geschlossen. Danach wird der nachfolgende Name gelesen und das entsprechende File geöffnet. Nach Ausgabe einer Meldung ("Include : Filename") wird der angegebene Block geladen und interpretiert. Am Ende des Ladevorgangs wird nach einer weiteren Meldung ("End-Include : Filename") das File wieder geschlossen, das zuvor verwendete File geöffnet und die zusätzlich geänderten Variablen **BLK** und **>IN** auf den alten Wert zurückgesetzt. Bei einem Fehler bleibt das alte File geschlossen und es kann sofort der Editor aufgerufen werden.
Weitere Anweisungen zum Laden der restlichen Programmteile sind immer im gewählten Screen u anzugeben. Wie bei **LOAD** ist die Nummer 0 nicht erlaubt.

LOOP (--) 83 I,R
(C: addr 4 --)

Mit der Befehl **LOOP** wird eine Zählschleife abgeschlossen und die dazu benötigten Parameter wie Anfangsadresse und Kontrollflag vom Datenstack entfernt.
Die von **LOOP** kompilierte (nicht direkt verfügbare) Runtimeroutine **(LOOP** erhöht den aktuellen Schleifenwert um 1 und kehrt zum Anfang der Schleife zurück, wenn der Endwert noch nicht erreicht wird. Da alle Informationen zur Schleife auf dem Returnstack erwartet werden, dürfen bei **LOOP** keine weiteren Daten dort abgelegt sein.

LWFILL (ptr count w --) X

Der 16Bit-Wert w wird ab der 32Bit-Adresse ptr count mal abgelegt. Wie bei **LFILL** sind prozessorspezifische Restriktionen bei den Parametern zu beachten. Der Befehl kann für das schnelle Füllen von 16Bit-Tabellen verwendet werden.

M* (n1 n2 -- d)

d ist das 32Bit-Produkt zweier vorzeichenbehafteter 16Bit-Werte. Eine Überschreitung des Zahlenbereiches ist dabei ausgeschlossen.

M+ (dw1 n -- dw2)

Zum 32Bit-Wert dw1 wird der vorzeichenbehaftete 16Bit-Wert n addiert. **M+** dient oft zur Aufsummierung von Werten.

M- (dw1 n -- dw2)

Ebenfalls zur gemischtgenauen Arithmetik dient der Befehl **M-** . Dabei wird von dw1 der vorzeichenbehaftete 16Bit-Wert n abgezogen.

M/ (d n -- q)

Bei der vorzeichenbehafteten, gemischtgenauen Division eines 32Bit-Wertes durch ein 16Bit-Wert wird nur der Quotient zurückgeliefert. Bei Division durch 0 oder Überschreitung des Zahlenbereiches wird -1 zurückgeliefert oder eine Fehlerbehandlung eingeleitet.

M/MOD (d n -- r q)

Anders als bei **M/** liefert der Befehl **M/MOD** bei der Division des vorzeichenbehaften Wert d durch n auch den Rest zurück.

MAKE (name ; --)

Wie bei **OPEN** wird ein Filename erwartet und dann dieses File für die Bearbeitung geöffnet. Dabei wird aber ein neues File mit der Filelänge 0 angelegt. Falls der angegebene Filename schon existiert, wird es vorher ohne Warnung gelöscht.
Sollte mit diesem File gearbeitet werden, so sind für die Zugriffe auf Screens erst noch die entsprechenden Blöcke mit **MORE** zu erzeugen.

MAX (n1 n2 -- n1 | n2) 83

Der größere der beiden vorzeichenbehafteten 16Bit-Werte bleibt auf dem Datenstack.

MAXAT (-- x y) UV

Ein ebenfalls über **OUTPUT** vektorisierter Befehl ist MAXAT . Er liefert die Anzahl der auf dem aktuellen Ausgabemedium verfügbaren Zeilen y und Spalten x. Da bei bei einem Bildschirm die Zählung mit 0 in der obersten linken Ecke beginnt, ist die angegebene Position für die unterste rechte Bildschirmecke jeweils um 1 zu erniedrigen.

MAXTLEN@ (-- sys)

Der Befehl **MAXTLEN@** liefert die Anzahl der maximal im USER-Bereich verfügbaren Speicheradressen. Wird beim Anlegen weiterer USER-Variablen dieser Wert überschritten, so erfolgt eine Fehlermeldung.

MIN (n1 n2 -- n1 | n2) 83

Der kleinere der beiden vorzeichenbehafteten 16Bit-Werte bleibt auf dem Datenstack.

MOD (n1 n2 -- r) 83

Durch **MOD** wird nur der Rest der Division von n1 durch n2 zurückgeliefert. Bei positiven Werte von n2 bleibt dabei r im Bereich 0 ... n2-1 und erfüllt damit die Bedingung einer Modulo-Funktion.

MORE (u --) 83

Das aktuelle File wird um u Blöcke zu je 1024 Zeichen vergrößert. Das File kann aber nicht verkleinert werden.

MOVE (addr1 addr2 len --)

Durch **MOVE** wird ein Speicherbereich von len Bytes ab addr1 nach addr2 kopiert. Dabei wird durch geeignete Wahl von **CMOVE** oder **CMOVE>** selbst bei überlappenden Speicherbereiche der ursprüngliche Inhalt übernommen.

MULTITASK (--)

Nach Aufruf dieses Befehls ist **PAUSE** für die Umschaltung zum nächsten Task vorbereitet. **MULTITASK** führt dabei selbst keinen Taskwechsel durch, sondern verändert nur den Befehl **PAUSE** . Mit **SINGLETASK** wird diese Veränderung durch Deaktivierung von **PAUSE** wieder rückgängig gemacht.

N>LINK (nfa -- lfa) E83

Aus der Namensfeldadresse eines Befehls wird die Linkfeldadresse ermittelt. Normalerweise befindet sich im KK-FORTH die LFA eine Zelle vor der NFA. Trotzdem sollte zur leichteren Portierung auf andere FORTH-Versionen **N>LINK** verwendet werden.

NAME> (nfa -- cfa) E83

Unter Berücksichtigung des Indirect-Bit im Header ermittelt der Befehl **NAME>** die tatsächliche Codefeldadresse eines Befehls aus der Namensfeldadresse.

NEGATE (n -- -n) 83

Der Befehl **NEGATE** ermittelt das Zweierkomplement (entspricht 0-n) der vorzeichenbehafteten 16Bit-Zahl n.

NEXT (-- ; R: n -- n-1 |) I,R
(C: addr 4 --)

NEXT ist der Abschluß der von Charles Moore speziell für den NC4000 entwickelten **FOR**...**NEXT**-Struktur. Der von **NEXT** kompilierte **NEXTBRANCH** erniedrigt den obersten Returnstackwert und spring zu FOR zurück, falls n ungleich 0 war. Ansonsten wird der Returnstackwert entfernt und das Programm hinter **NEXT** fortgesetzt.

NEXT-LINK (-- addr) SV

(NEXT ist die in Maschinensprache geschriebene oder direkt in Hardware realisierte

Kernroutine des FORTH. Es ist dafür verantwortlich, daß der nächste Befehl ausgeführt wird. Bei einigen KKF-Versionen (wie beim Z80) ist diese Routine nur einmal im System vorhanden und alle Befehle springen nach der Ausführung an diese Adresse. Bei anderen Prozessoren (8086, 68000) ist die direkte Einbindung der Routine in die einzelnen Befehle günstiger, weil dadurch das FORTH erheblich schneller und trotzdem nicht wesentlich größer wird.
Trotzdem ist es z.B. für einen Tracer erforderlich, vor der Ausführung des nächsten Befehls eigene Routinen abzuarbeiten. Dies geschieht am einfachsten durch die Veränderung des **(NEXT** . Um nun auch bei KKF-Versionen mit direkt eingefügtem **(NEXT** ein Tracer zu implementieren, werden beim Assemblieren der (NEXT-Routine auch ein Verkettungszeiger im Programm abgelegt. Die Adresse des letzten Zeigers wird dann in der USER-Variablen **NEXT-LINK** gespeichert.
Beim Löschen von Befehlen wird neben **VOC-LINK** und den Verkettungen der Befehlsnamen auch dieser Zeiger zurückgesetzt.

NEXTBRANCH (-- ; R: n -- n-1 |) (I),R

Der von **NEXT** kompilierte **NEXTBRANCH** testet den obersten Returnstackwert. Ist der Wert 0, so wird das Programm dahinter fortgesetzt. Ansonsten wird n um eins erniedrigt und zum Anfang der Schleife zurückgesprungen. Da der Aufbau des Sprunges systemspezifisch ist, sollte er nur in Kombination mit **>MARK** oder **<RESOLVE** verwendet werden:
Beispiel:
```
: NEXT   ( addr 4 -- )
   dup 4 ?pairs drop         ( Test )
   postpone nextbranch  <resolve ; immediate restrict
```

NIP (n1 n2 -- n2)

NIP entspricht der Befehlsfolge `SWAP DROP` und entfernt damit den zweiten 16Bit-Eintrag vom Datenstack.

NODEFER (--)

Die Standardroutine zu neu generierten DEFER-Befehlen bewirkt die Ausgabe einer Fehlermeldung ("DEFER not defined").

NOOP (--)

Wie schon der Name sagt, tut dieser Befehl nichts. **NOOP** dient meistens als Platzhalter in Programmen oder DEFER-Routinen. Die Codefeldadresse von **NOOP** zeigt in den meisten KKF-Implementationen auf die (NEXT-Routine (siehe dazu **NEXT-LINK**).

NOT (w1 -- w2) 83

Durch die bitweise XOR-Verknüpfung des 16Bit-Wertes w1 mit -1 wird das Einerkomplement ermittelt. Jedes gesetzte Bit in w1 ist in w2 gelöscht und umgekehrt.

NOTFOUND (csa --)

NOTFOUND wird von ' und dem Interpreter aufgerufen, wenn der angegebene String nicht gefunden wurde. Nach Entfernen der Stringadresse wird **ERROR** mit Fehlernummer $7F11 (hat keinen Text, da schon von **(ERRORHANDLER** ein Fragezeichen ausgegeben wird) aufgerufen.

NUMBER (n -1 | d 0> -- n | d) D

Die vom Interpreter genutzte DEFER-Routine zeigt normalerweise auf **DROP** und läßt deshalb nur den eigentlichen Wert auf dem Datenstack. Falls durch veränderung von **NUMBER?** zusätzliche Datentypen erkannt werden, muß **NUMBER** ebenfalls korrigiert werden.

NUMBER, (n -1 | d 0> --) DX

Die vom Kompiler genutzte DEFER-Routine zeigt auf einen unmittelbar danach folgenden, versteckten Befehl. Diese kompiliert den Wert als 16Bit- oder 32Bit-Literal. Falls durch Veränderung von **NUMBER?** zusätzliche Datentypen erkannt werden, muß **NUMBER,** ebenfalls korrigiert werden.

NUMBER? (csa -- addr 0 | d 0> | n -1) DX

Die von **INTERPRET** genutzte DEFER-Routine zeigt auf einen unmittelbar danach folgenden, versteckten Befehl. Dieser erwartet eine Counted-String-Adresse auf dem Datenstack. Gemäß der aktuellen Zahlenbasis wird der Zahlenstring zu einem 16Bit- oder 32Bit-Wert akkumuliert. Falls das erste Zeichen "$" , "&" oder "%" ist, wird während der Ausführung des Befehls die Zahlenbasis auf 16, 10 oder 2 gestellt. Danach kann noch das Vorzeichen "-" angegeben werden. Positive Vorzeichen und Leerzeichen im Zahlenstring sind nicht zulässig.
Falls ein nicht zu interpretierendes Zeichen im String auftaucht, wird **NUMBER?** mit Angabe der entsprechenden Zeichenadresse und dem FALSE-Flag beendet. Falls alle Zeichen gültig waren, so wird unter dem Flag -1 der entsprechende 16Bit-Wert zurückgeliefert.
Um auch 32Bit-Werte eingeben zu können, ist die Verwendung eines Punktes an einer beliebigen Stelle innerhalb des Zahlenstrings zulässig. Auf dem Datenstack wird dann der vollständige 32Bit-Wert und die Anzahl+1 der nach dem Punkt noch folgenden Ziffern zurückgeliefert.
NUMBER? verwendet für die Zahlenumwandlung den Befehl **>NUMBER** und verändert deshalb die USER-Variable **DPL** .

OF (n1 n2 -- n1 |) I,R
(C: 5 -- addr -5)

Mit **OF** wird ein Bedingungszweig einer **CASE**-Struktur eingeleitet. Dazu kompiliert **OF** sowohl Vergleich als auch Sprung und legt die Adresse zusammen mit dem Flag -5 auf dem Datenstack ab.
Bei der Ausführung des Befehls werden beide Werte miteinander verglichen. Bei Gleichheit werden beide Werte vom Datenstack entfernt und das Programm nach OF fortgesetzt. Ansonsten wird nur n2 entfernt und hinter dem zugehörigen **ENDOF** gesprungen.

OFF (addr --)

Das 16Bit-Flag (oder eine Variable) mit der angegebenen Adresse wird durch Einschreiben des Wertes 0 abgeschaltet.

ON (addr --)

Das 16Bit-Flag (oder eine Variable) mit der angegebenen Adresse wird durch Einschreiben des Wertes -1 eingeschaltet.

ONLYFORTH (--)

Nach Ausführung des Befehls **ONLYFORTH** ist nur noch das Vokabular **FORTH** sowohl in **CURRENT** als auch im festen und variablen Teil von **CONTEXT** eingetragen. **ONLYFORTH** wird immer nach dem Entfernen von Befehlen verwendet.

OPEN (name ; --)

Falls der angegebene Name schon als Filebefehl vorhanden ist, wird er direkt aufgerufen und damit die zugehörige Runtimeroutine **(OPEN** ausgeführt.
Ansonsten wird zuerst geprüft, ob ein File mit entsprechenden Namen im aktuellen Pfad existiert. Anschließend erfolgt die Definition und Ausführung von name als neuer Filebefehl. Dazu wird zuerst der Filename als neuen Befehl im Dictionary eingetragen, der Filename als String im Parameterfeld des Befehls abgelegt und der DOES>-Teil von **OPEN**

als Runtimeroutine dieses Befehls angegeben.
(OPEN erhält in beiden Fällen jeweils die Counted-String-Adresse des Filenamens und Schließt vor der Suche nach dem neuen File ein noch geöffnetes File.

OR (w1 w2 -- w3) 83
Die beiden 16Bit-Werte werden bitweise miteinander OR-Verknüpft. Sobald einer der beiden korrespondierenden Bits von w1 oder w2 gesetzt ist, wird auch das Bit in w3 gesetzt.

ORDER (--) E83
Alle **CONTEXT** - und das aktuelle **CURRENT** -Vokabular werden durch **ORDER** ausgegeben.
Beispiel: `OnlyForth Tools also Forth order` liefert
`FORTH TOOLS FORTH FORTH`

OUTPUT (-- addr) U
Die USER-Variable **OUTPUT** zeigt auf eine mit **UTABLE:** definierte Tabelle, in der die Codefeldadressen der zu den einzelnen Ausgabebefehlen gehörenden Routinen stehen. Dadurch kann mit einem Befehl die Umleitung der im System vordefinierten Routinen (z.B. Ausgabe auf Terminal) auf ein eigenes Ausgabegerät (z.B. Drucker) durchgeführt werden. Beim Start des KK-FORTH wird durch **STANDARD-IO** die Ausgabe auf die letzte SAVE-Einstellung zurückgestellt. Die Tabelle hat folgenden Aufbau:

	addr-4	Offset von OUTPUT im USER-Bereich
	addr-2	Länge des Datenfeldes ab addr
OUTPUT -->	addr	CFA der Initialisierungsroutine
	addr+2	CFA der Routine EMIT?
	addr+4	CFA der Routine EMIT
	addr+6	CFA der Routine TYPE
	addr+8	CFA der Routine CLS
	addr+10	CFA der Routine CR
	addr+12	CFA der Routine DEL
	addr+14	CFA der Routine BELL
	addr+16	CFA der Routine MAXAT
	addr+18	CFA der Routine AT
	addr+20	CFA der Routine AT?

OVER (w1 w2 -- w1 w2 w1) 83
Der zweite 16Bit-Wert auf dem Datenstack wird kopiert und auf dem Stack erneut abgelegt.

P! (w addr --)
Ausgabe des Wertes w ab der Portadresse addr. Die Reihenfolge der Bytes und die Anzahl der benötigten Portadressen ist versionsabhängig.

P@ (addr -- w)
Der aktuelle Wert w wird ab der Portadresse addr gelesen. Die Reihenfolge der Bytes und die Anzahl der benötigten Portadressen ist versionsabhängig.

PAD (-- addr)
Der Befehl **PAD** liefert eine Adresse, ab der die mit " eingegebenen Strings abgelegt werden. **<#** verwendet ebenfalls diese Adresse als Ende eines formatierten Zahlenstrings bei Ausgabe. addr liegt im KK-FORTH meist 84 Adressen über dem Arbeitsspeicher **WDP@** und hat einen Länge von 256 Adressen.

PARSER (csa --) D

Jedes Befehlswort in einer Eingabezeile wird durch **INTERPRET** einzeln geholt und durch Aufruf des DEFER-Wortes **PARSER** entweder interpretiert oder kompiliert. Im KK-FORTH wird dieser DEFER-Befehl nur durch die Befehle **[** und **]** zusammen mit der USER-Variablen **STATE** verändert.

PAUSE (--) D

PAUSE ist der Kernbefehl des kooperativen Multitasker im KK-FORTH. Der Aufruf von **PAUSE** bewirkt eine Stillegung des aktuellen Tasks und die Aktivierung des nächsten bereiten Tasks in der verketteten Liste. **PAUSE** ist beim Start des KK-FORTH noch auf **NOOP** gesetzt. Erst der Befehl **MULTITASK** verändert seine Eigenschaften so, wie es oben beschrieben wurde. Mit **SINGLETASK** wird **PAUSE** wieder auf **NOOP** gesetzt und damit deaktiviert. Die für die Erzeugung von weiteren Tasks benötigten Befehle gehören nicht zum KKF-Kern und müssen deshalb nachgeladen werden.

PC! (char addr --) X

Ein 8Bit-Wert char wird ab Portadresse addr geschrieben. Dieser Befehl steht nur dann zur Verfügung, wenn im System auch ein 16Bit-Portzugriff möglich ist.

PC@ (addr -- char) X

Ein 8Bit-Wert char wird ab Portadresse addr gelesen. Dieser Befehl steht nur dann zur Verfügung, wenn im System auch ein 16Bit-Portzugriff möglich ist.

PERFORM (addr --) 83

Ab der Adresse addr ist die Codefeldadresse eines Befehls kompiliert. **PERFORM** liest diesen Wert aus und führt den entsprechenden Befehl wie mit **EXECUTE** aus. Der Befehl wird oft bei schnellen Fallunterscheidungen durch Auslesen der Codefeldadresse aus einer Tabelle verwendet.

Beispiel:
```
Create CFA-Feld ] Aktion0 Aktion1 Aktion2 Aktion3 [
: Aktion_n   ( n -- ) ( Aktion n ausführen )
   cells  CFA-Feld +  perform ;
```

PICK (nx ... n1 n0 n -- nx ... n1 n0 nn) 83

Kopieren des n-ten 16Bit-Wertes des Datenstacks. Dabei beginnt die Zählung bei 0.

Beispiele: `0 pick` entspricht **DUP**
`1 pick` entspricht **OVER**

POSTPONE (name ; --) ANSI I,R

Der im ANSI-Vorschlag neu aufgenommene Befehl **POSTPONE** soll die Konfusion bei der Verwendung von **COMPILE** und **[COMPILE]** vermeiden. Mit `POSTPONE Name` ist es jetzt egal, ob der nachfolgende Befehl "immediate" ist oder nicht. Es wird automatisch die tatsächlich gewünschte Kompilereigenschaft des Befehls übernommen. Ein versionsabhängier Befehle wie **?BRANCH** kann bei Verwendung von `POSTPONE ?BRANCH` in Kontrollstrukturen auch als Immediate-Befehl definiert werden, ohne das Anwenderprogramm zu ändern.

Beispiel:
... postpone (Immediate-)Wort ...
ersetzt ... [compile] Immediatewort ...
und ... compile Wort ...

PTR>D (ptr -- d)

Umrechnung der Pointeradresse in eine 32Bit-Adresse. Dadurch lassen sich unabhängig von der tatsächlichen Speicheraufteilung lineare Adressierung simulieren.

PUSH (addr --) (R: -- w addr pop) R

Der Wert und die Adresse einer 16Bit-Variable wird für die Dauer des aktuellen Befehls auf dem Returnstack gespeichert und danach automatisch wieder zurückgeschrieben. Dazu wird neben den Werten auch die Adresse der entsprechenden Routine auf den Returnstack gelegt.

Beispiel:
```
: HEX.   ( n -- ) ( n hexadezimal ausgeben )
   base push    ( Zahlenbasis merken )
   hex  u.  ;   ( Wert hexadezimal ausgeben )
```
Vor PUSH dürfen keine anderen Werte auf dem Returnstack abgelegt werden. Sobald **EXIT** (wird von **;** kompiliert) ausgeführt wird, bringt die danach automatisch gestartete Routine pop den alten Inhalt wieder zurück in die Variable.

QUERY (--)

Eine Eingabezeile wird vom Terminal übernommen und in den Speicherbereich **TIB** gebracht. Die USER-Variable **#TIB** enthält danach die Länge des Eingabestrings. **>IN** und **BLK** werden auf Null zurückgesetzt. Die Eingabe wird über **EXPECT** abgewickelt und erlaubt damit auch die dort beschriebenen Steuerungen (CTRL+X: Zeile löschen ...).
Bei einigen KKF-Versionen ist auch die Übergabe einer Befehlszeile beim Start des FORTH möglich. Es wird dann beim ersten Aufruf statt der Terminaleingabe der Rest der Eingabezeile übernommen. Ist Bit0 von **SFLAG** gesetzt, so wurde die Befehlszeile schon geholt oder es ist kein Text verfügbar.

QUIT (--)

Nach dem Löschen des Returnstacks und dem Zurücksetzen des Flag **INDENT?** beginnt eine Endlosschleife, in der mit **QUERY** eine Zeile geholt und mit **INTERPRET** interpretiert oder kompiliert wird. Vor der Wiederholung der Schleife erfolgt noch die Ausgabe der Meldungen "ok" oder "]" durch Aufruf von **.STATE** .

R# (-- addr) U

Die USER-Variable **R#** übernimmt bei einem Fehler während des Interpretieren eines Screens den aktuellen Wert von **>IN** . Durch Auswertung von **SCR** und **R#** ist dann nach einem Fehler durch **V** der Einsprung in den Editor (Cursor nach der Fehlerposition) möglich.

R0 (-- addr) U

Die USER-Variable **R0** enthält die Anfangswert des Returnstacks. Ob es sich bei dem Wert in **R0** um eine Adresse oder um einen Offset handelt, ist Versionsabhängig.

R> (-- w ; R: w --) 83 R

Der nur innerhalb einer :-Definition zulässige Befehl bringt einen 16Bit-Wert vom Returnstack zum Datenstack. Da auf dem Returnstack auch die Rücksprungadressen bei Aufruf anderer FORTH-Worte abgelegt werden, kann auf den Wert nur innerhalb der gleichen Definition mittels **R@** , **R>** und **RDROP** zugegriffen werden.

R@ (-- w ; R: w -- w) 83

Ein auf dem Returnstack befindlicher 16Bit-Wert wird zum Datenstack kopiert.

RCLEAR (--) R

Nach Aufruf von '**RCLEAR** wird der Returnstackzeiger wieder auf den in **R0** gespeicherten Wert gestellt und dadurch der Returnstack gelöscht. **RCLEAR** darf nur in Endlosschleifen wie **QUIT** verwendet werden, da danach kein Verlassen dieses Befehls mehr möglich ist.

RDEPTH (-- n)

Der Befehl **RDEPTH** liefert die Größe des aktuellen Returnstacks zurück. Dieser Befehl wird im KK-FORTH aber nicht verwendet und dient nur zu Kontrollzwecke.

RDROP (-- ; R: w --) R

Der nur innerhalb einer :-Definition zulässige Befehl entfernt einen 16Bit-Wert vom Returnstack.

RECURSE (--) I

Normalerweise kann der gerade in Definition befindliche Befehl nicht aufgerufen werden, weil sein Name aus der Suchliste entfernt worden ist. Für die rekursive Programmierung kann durchaus ein Neueinsprung in den gleichen Befehl sinnvoll sein. **RECURSE** holt die Codefeldfeldadresse der aktuellen Definition und kompiliert sie. Es muß bei der Anwendung darauf geachtet werden, daß Datenstack und der Returnstack begrenzt sind.

Beispiel:
```
: FAKULTAET   ( n -- u ) ( Nur n=0 bis 8 erlaubt )
   ?dup  IF  dup 1-  recurse  *  ELSE  1  THEN ;
```

REMOVE (dp heap --)

Das von allen Befehlen zum Löschen von Befehlsheader verwendete **REMOVE** erwartet auf dem Datenstack die neue Dictionaryadresse und die Heapadresse. Falls db kleiner heap ist, werden alle zwischen diesen beiden Adressen liegenden Befehlsnamen entfernt. Der Dictionaryanfang wird auf dp und der Heapanfang auf heap gesetzt. Falls der neue Dictionaryanfang unter der im (SYSVAR-Bereich gespeicherten Adresse liegt, sollte durch **SAVE** die neue Speicheraufteilung gesichert werden.

REPEAT (--) 83 I,R
(C: 2 ... addrn -2 addr1 2 --)

Mit **REPEAT** wird das Ende einer **BEGIN** ... f **WHILE** ... **REPEAT** -Schleife markiert. Es wird ein bedingungsloser Rücksprung zu **BEGIN** (Adresse addr1) kompiliert und alle anderen Sprünge auf die darauffolgende Adresse korrigiert. Die Werte 2 bzw. -2 dienen dabei sowohl als Kontrolle der richtigen Schleifenstruktur als auch zur Erkennung des Ende der Adreßliste.

RESTRICT (--)

Der nur unmittelbar nach einer Code- oder :-Definition sinnvolle Befehl markiert diesen als "restrict". Ein so markierter Befehl darf dann nur noch innerhalb einer :-Definition verwendet werden. Außerhalb der Definition wird ein Fehler ("Is restrict") ausgegeben und der Befehl nicht ausgeführt.

RESTRICT wird nur dann eingesetzt, wenn durch Verwendung dieses Befehls außerhalb von :-Definitionen ein Systemabsturz (z.B. bei **>R** , **R>** oder **RDROP**) oder eine fehlerhafte Befehlsstruktur verursacht wird.

REVEAL (--)

Die Umkehrung des Befehls **HID**E macht den versteckten Befehl wieder sichtbar. Dazu wird die in **LAST** gespeicherte Namensfeldadresse des Befehls geholt und dieser Befehl wieder in das **CURRENT** -Vokabular eingebunden. Ein Fehler ("No definition") wird ausgegeben, wenn der Inhalt von **LAST** Null ist.

ROLL (wn wm ... w0 n -- wm ... w0 wn) 83

Der n-te Datenstackeintrag wird zum neuen TOS. Da die Zählung mit 0 begonnen wird entspricht `1 ROLL` dem Befehl **DUP** und `2 ROLL` dem Befehl **ROT** .

ROT (w1 w2 w3 -- w2 w3 w1) 83
Der dritte Datenstackeintrag wird zum TOS rotiert.

RP! (w --) R
Der nur innerhalb einer :-Definition zulässige Befehl **RP!** setzt die aktuelle Position des Returnstackzeigers auf w. Dabei sollten nur der in **R0** gespeicherte oder der vorher mit **RP@** geholte Wert verwendet werden, da die Bedeutung von w Versionsabhängig ist.

RP@ (-- w)
Für Menüprogramme mit Wiedereinsprung nach Fehler muß auch der Returnstack korrigiert werden. Der aktuelle Wert des Returnstackzeigers ist mit **RP@** abfragbar. Die Bedeutung von w ist Versionsabhängig.

S0 (-- addr) U
Die USER-Variable **S0** enthält den Anfangswert des Datenstacks. Der Datenstack ist leer, wenn mit **SP@** dieser Wert geliefert wird. Die Bedeutung des in **S0** angegebenen Wert ist Versionsabhängig.

S>D (n -- d)
Umwandlung eines vorzeichenbehafteten 16Bit-Wertes in sein 32Bit-Equivalent. Dazu wird das Bit 15 von n auf die Bits 1631 übernommen.

SAVE (--)
Dieser Befehl löscht den Heap, übernimmt die USER-Variablen **INPUT** , **OUTPUT** und **DISC** in die Systemvariablen und speichert danach alle Systemvariablen ab (SYSVAR . **SAVE** wird auch von **SAVESYSTEM** verwendet und dient zur Speicherung der aktuellen Systemeinstellung für den Neustart.

SAVE-BUFFERS (--) 83
Falls noch ein mit **UPDATE** markierter Block im Speicher existiert, so wird dieser zurückgeschrieben und bei fehlerfreien Übertragung das UPDATE -Flag gelöscht.

SAVESYTEM (name ; --)
Nach dem Aufruf von **SAVE** wird das aktuelle FORTH-System als Programm oder als zusätzliches Bitimage mit nachfolgenden Name auf Diskette gespeichert. Falls vorher der DEFER-Vektor 'ABORT auf eigene Befehle umgeleitet wurde, wird beim Start des gespeicherten Systems dieser Befehl ausgeführt und dadurch eine Autostart-Applikation gestartet. Die Speicherung erfolgt immer in folgender Reihenfolge:

0 ... DP	RAM-Image des Programmkerns oder der Zusätze
danach	Variablenbereich
danach	Heapbereich (meist leer)
danach	zuletzt definierter Task ... Standardtask

SCAN (addr1 len1 char -- addr2 len2)
Der String mit Anfangsadresse addr1 und der Länge len1 wird nach dem ersten Auftreten des Zeichens char untersucht. Die Adresse dieses Zeichens und die Restlänge des Strings wird zurückgeliefert. Falls das Zeichen char nicht im String vorkommt oder die Stringlänge 0 ist, wird Adresse addr2 auf addr1+len1 und die Restlänge len2 auf 0 gesetzt.

SCAN> (addr len1 char -- addr len2)

Analog **SCAN** wird nach dem ersten Auftreten des Zeichens char gesucht. Jedoch beginnt **SCAN>** am Stringende und sucht Richtung Stringanfang.

SCR (-- addr) U

Die USER-Variable **SCR** enthält nach einem Fehler die Blocknummer des zuletzt Interpretierten/Kompilierten Programmes.

SEAL (--) E83

Die Umkehrung zu **ALSO** entfernt das letzte in die **CONTEXT**-Liste aufgenommene Vokabular. Dieser Befehl ist mit Vorsicht anzuwenden, da nach Entfernung aller Vokabulare kein Befehl mehr interpretiert und kompiliert werden kann.

SFLAG (-- sys) SV

Durch **SFLAG** wird die Adresse des Systemflags geliefert. Dieses Flag wird beim Systemstart aus der Kopie der Systemvariablen entnommen und selbst bei **SAVE** nicht zurückgeschrieben. Nur durch direkte Manipulation der Kopie (Adresse: `SFLAG SYSVAR - (SYSVAR +`) kann der Wert für den nächsten Systemstart verändert werden.

Jedes einzelne Bit von **SFLAG** hat eine spezielle Bedeutung:

Bit 0=1: Keine Befehlszeile (mehr) vom Betriebssystem zu holen
Bit 1=1: Keine Ausgabe der "exist"-Meldung durch **CREATE**
Bit 2=1: Es ist kein Zeilenpuffer für **EDITLINE** vorhanden
Bit 3=1: Ein-/Ausgabeschnittstelle wurde schon Initialisiert

SHIFT (w1 n -- w2) 83

Der 16Bit-Wert w1 wird logisch um n Bits nach rechts (n<0) oder links verschoben. Das freigewordene Bit wird immer mit 0 aufgefüllt. Nur die Werte -16<n<16 sind sinnvoll, da ansonsten nur noch 0-Bits in w2 enthalten sind.

SIGN (n --) 83

Nur wenn n negativ (n<0) ist, wird ein Minuszeichen vor den Zahlenausgabestring gesetzt.

SINGLETASK (--)

Die Umkehrung des Befehls **MULTITASK** verändert **PAUSE** so, daß keine Taskumschaltung mehr erfolgt. **SINGLETASK** wird bei jedem Programmstart durch **COLD** aufgerufen.

SKIP (addr1 len1 char -- addr2 len2)

Der String mit Anfangsadresse addr1 und der Länge len1 wird nach dem ersten Auftreten des von char abweichenden Zeichens untersucht. Die Adresse dieses Zeichens und die Restlänge des Strings wird zurückgeliefert. Falls in dem String nur das Zeichen char vorkommt oder die Stringlänge 0 ist, wird Adresse addr2 auf addr1+len1 und die Restlänge len2 auf 0 gesetzt.

SKIP> (addr len1 char -- addr len2)

Vom Stringende abwärts wird das erste von char abweichende Zeichen gesucht und die Länge des Strings entsprechend korrigiert. Falls der gesamte String aus Zeichen char besteht oder die Länge 0 besitzt, wird auch len2 zu 0. **SKIP>** wird im KK-FORTH nur durch **-TRAILING** zur Abschneidung der Leerzeichen am Stringende herangezogen.

SP! (w --)

SP! setzt die aktuelle Position des Datenstackzeigers auf w. Dabei sollten nur (wie von **DCLEAR**) der in **S0** gespeicherte oder der vorher mit **SP@** geholte Wert verwendet werden.

SP@ (-- w)

SP@ liefert die aktuelle Position des Datenstacks. Daraus kann dann die Datenstacklänge errechnet werden. Die Bedeutung von w ist Versionsabhängig.

SPACE (--) 83

Der durch **OUTPUT** vektorisierte Befehl **SPACE** gibt ein Leerzeichen (ASCII-Wert 32) aus.

SPACES (n --) 83

Falls n positiv ist, werden n Leerzeichen ausgegeben.

SPAN (-- addr) U,83

In der USER-Variablen **SPAN** speichert **EXPECT** die bei der Eingabe übernommen Zeichen und **EDITSTRING** die Maximallänge des zu editierenden Strings.

SS@ (-- ptr) X

Der Befehl **SS@** liefert einen Pointer auf den Anfang des Daten- und Returnstacks. In den meisten KKF-Versionen liefern **SS@** , **CS@** und **DS@** den gleichen Wert, da Programm, Daten und Stack in einem einzigen Segment zusammengefaßt sind. Bei Prozessoren mit Hardwarestack ist dieser Befehl nicht verfügbar.

STANDARD-IO (--)

Nach Aufruf von **STANDARD-IO** sind die USER-Variablen **INPUT** , **OUTPUT** und **DISC** auf ihre im SYSVAR-Bereich gespiecherten Werte zurückgesetzt und die entsprechenden Initialisierungsroutinen aufgerufen. Die Ein-/Ausgabe und das Fileinterface wird dann über die Standard-Schnittstelle abgewickelt. Bei **SAVE** werden die entsprechenden Einstellungen in den SYSVAR-Bereich übernommen und damit dauerhaft verändert.

STATE (-- addr) U

Die USER-Variable STATE enthält ein Flag, ob momentan die Befehle interpretiert (**STATE**=0) oder kompiliert werden sollen. Es genügt aber nicht die Veränderung von **STATE** zur Änderung der Eigenschaft. Es muß gleichzeitig der von **INTERPRET** aufgerufene DEFER-Vektor **PARSER** umgestellt werden. Im KK-FORTH wird die Umstellung von Interpretermodus in den Kompiliermodus und wieder zurück durch die beiden Befehle **[** und **]** abgewickelt.

STOP? (-- f)

Der von **DUMP** , **.S** , **WORDS** und **LIST** verwendete Befehl STOP? dient zum Anhalten der Ausgabe. Solange keine Taste gedrückt wird, liefert **STOP?** das Flag 0 und kehrt sofort zurück. Wurde eine Taste ungleich der #BRK-Taste gedrückt, so wird solange gewartet, bis der nächste Tastedruck erfolgt und dann ebenfalls ein Flag zurückgeliefert. Nur wenn das Zeichen #BRK erkannt wird, liefert **STOP?** das Flag -1.

STRING (addr len --) UV

In Umkehrung zum Befehl **TYPE** erwartet der über die USER-Variable **OUTPUT** vektorisierte Befehl **STRING** einen String mit angegebener Länge und bringt ihn zur

Adresse addr. Dabei wird aber keine Bearbeitung der Zeichen durchgeführt und der Befehl erst nach Empfang aller len Zeichen beendet.

SWAP (w1 w2 -- w2 w1) 83
Die obersten beiden 16Bit-Einträge werden vertauscht.

SYSCON (-- sys)
SYSCON liefert die Speicheradresse eines Bereiches mit wichtigen Systemparametern. Die Bedeutung der einzelnen Werte ist bei der Speicheraufteilung beschrieben.

SYSVAR (-- sys)
SYSVAR liefert die Speicheradresse eines Bereiches mit allen veränderlichen Systemvariablen wie Dictionary- und Variablenadresse. Diese Daten werden beim Systemstart initialisiert und danach (wie auch bei **SAVE**) in den Speicherbereich ab **(SYSVAR** kopiert. Der Aufbau des SYSVAR-Bereiches in der Speicheraufteilung beschrieben. Der Bereich hat eine Länge von **SYSVARLEN@** (meist $50) Adressen.

SYSVARLEN@ (-- sys)
Die Größe des für die Systemvariablen benötigten Speicherbereiches wird durch **SYSVARLEN@** zurückgeliefert. In der aktuellen Version belegt der Bereich 40 Zellen.

TASK-LINK (-- sys) SV
Die Systemvariable **TASK-LINK** enthält einen Zeiger auf den Anfang des USER-Bereich vom zuletzt definierten Task. Bei **SAVESYSTEM** werden nur jeweils **TMAXLEN@** Adressen aus jedem USER-Bereich gespeichert. Es können im KKF-Kern keine Tasks gelöscht werden, deshalb verändert **REMOVE** diesen Zeiger nicht.

TASK0 (-- addr)
TASK0 liefert die USER-Anfangsadresse des Standardtask. Mit diesem Task wird nach **BOOT** oder **COLD** das FORTH gestartet.

TASKADDR (-- sys) SV
Die Anfangsadresse des aktuellen Tasks enthält **TASKADDR** . In den darauffolgenden Adressen sind die Daten für die Taskumschaltung und alle USER-Variblen gespeichert. Ab Offset `MAXTLEN@ 2+` beginnt der Eingabepuffer.

TASKADDR@ (-- addr)
TASKADDR@ liefert die Anfangsadresse des aktuellen USER-Bereiches.

TASKS (-- sys) SV
Die Anzahl der bis jetzt definierten Tasks ist in der Systemvariable **TASKS** gespeichert. Dieser Wert wird nur von **SAVESYSTEM** zur Speicherung aller Taskbereiche und durch **BOOT** zur Ermittlung der Image-Endadresse herangezogen.

TDP (-- sys) SV
In der Systemvariablen **TDP** ist die Anfangsadresse des ersten Taskbereiches gespeichert. Da jeder Task einen eigenen Arbeits- und Datenstackbereich unterhalb der USER-Adresse besitzt, liegt die in **TDP** gespeicherte Adresse unterhalb von **TASK-LINK** . Falls **TDP** verändert werden soll, so muß der Heap gelöscht sein und der Variablenbereich verschoben werden.

THEN (--) 83 I,R
(C: addr ±1 --)
Mit **THEN** wird das Ende einer Fallunterscheidung markiert. Nach dem Test auf die richtige Struktur (Flag ±1) wird die Zieladresse des Sprunges (Adresse addr) korrigiert.

THRU (+n1 +n2 --)
Die Blöcke n1 bis einschließlich n2 werden vom gerade geöffneten File geladen.

TIB (-- addr)
Die Adresse des Eingabepuffers wird aus der USER-Variable **>TIB** ausgelesen und zum Stack gebracht.

TLEN (-- sys) SV
In der Systemvariablen **TLEN** wird die Anzahl der von den Tasks belegten Adressen für die USER-Variablen gespeichert. Bei jeder Definition neuer USER-Variablen wird dieser Offset verändert und eine Fehlerbehandlung ausgelöst, wenn der maximal verfügbare Speicher von **TMAXLEN@** Adressen überschritten wird.

TRUE (-- -1)
Die Konstante **TRUE** liefert den Wert -1. Dieser Wert wird von allen Vergleichsoperationen geliefert, wenn die Bedingung zutrifft.

TUCK (w1 w2 -- w2 w1 w2)
Der oberste Wert des Datenstacks wird unter den zweiten 16Bit-Eintrag kopiert. Dieser Befehl entspricht dem im volksFORTH verwendeten **UNDER** .

TYPE (addr len --) UV,83
Der durch **OUTPUT** vektorisierte Ausgabebefehl gibt einen String aus. Dabei werden alle Zeichen unverändert weitergegeben.

U. (u --)
Der vorzeichenlose 16Bit-Wert wird linksbündig mit einem nachfolgenden Leerzeichen ausgegeben.

U.R (u n --)
Der vorzeichenlose 16Bit-Wert u wird gemäß der aktuellen Zahlenbasis rechtsbündig in einem Feld von n Zeichen ausgegeben. Falls n negativ oder kleiner als der Ausgabestring ist, wird der ganze String ohne zusätzliche Leerzeichen ausgegeben.

U2/ (u1 -- u2)
Durch logisches Schieben der Bits in u1 nach rechts wird der Wert vorzeichenlos halbiert.

U< (u1 u2 --)
Die beiden vorzeichenlosen 16Bit-Werte werden verglichen. Falls u1 kleiner als u2 ist, wird das TRUE-Flag -1 zurückgeliefert.

U> (u1 u2 --)
Die beiden vorzeichenlosen 16Bit-Werte werden verglichen. Falls u1 größer als u2 ist, wird das TRUE-Flag -1 zurückgeliefert.

U? (addr --)

In Anlehnung an **?** gibt der Befehl **U?** einen in der 16Bit-Variablen ab addr gespeicherten Wert vorzeichenlos aus.

UFLAG (-- sys) SV

Die Systemvariable **UFLAG** wird wie **SFLAG** nur beim Systemstart durch **BOOT** übernommen und kann nur durch Manipulation der (SYSVAR-Kopie dauerhaft verändert werden. Das Flag wird vom KKF-Kern nicht verwendet, aber bei **BYE** nach Möglichkeit an das System zurückgeliefert und kann dadurch z.B. beim PC in Batch-Prozeduren ausgewertet werden.

UM* (u1 u2 -- ud) 83

Die beiden vorzeichenlosen 16Bit-Werte werden miteinander multipliziert und das 32Bit-Ergebnis wieder auf dem Stack abgelegt.

UM/MOD (ud u -- r q) 83

Nach der Division eines 32Bit-Wertes ud durch den 16Bit-Wert u werden Rest r und Quotient q wieder auf dem Stack abgelegt. Je nach KKF-Version wird bei einer Überschreitung des 16Bit-Bereiches von Quotient q (ud zu groß oder u1=0) entweder eine Fehlerbehandlung eingeleitet oder nur die Ergebnisse -1 für r und q zurückgeliefert.

UMAX (u1 u2 -- u1 | u2)

Der größere der beiden vorzeichenlosen 16Bit-Werte bleibt auf dem Datenstack.

UMIN (u1 u2 -- u1 | u2)

Der kleinere der beiden vorzeichenlosen 16Bit-Werte bleibt auf dem Datenstack.

UNLOOP (-- ; R: addr limit count --) ANSI R

Der Befehl **UNLOOP** sollte nur innerhalb einer **DO** ... **LOOP** -Schleife zur Bereinigung des Returnstacks vor Verlassen des Befehls mit **EXIT** aufgerufen werden.

UNTIL (f --) 83 I,R
(C: 2 ... addr2 -2 addr1 2)

Mit **UNTIL** wird das Ende einer **BEGIN** ... f **WHILE** ... **UNTIL** -Schleife markiert. Es wird ein bedingter Rücksprung zu **BEGIN** (Adresse addr1) kompiliert und alle anderen Sprünge auf die darauffolgende Adresse korrigiert. Die Werte 2 bzw. -2 dienen dabei sowohl als Kontrolle der richtigen Schleifenstruktur als auch zur Erkennung des Ende der Adreßliste. Im Programm wird dann zu **BEGIN** zurückgesprungen, wenn f=0 ist.

UPC (char1 -- char2)

Mit **UPC** wird ein Zeichen in Großschrift umgewandelt. Da im KK-FORTH auch die Umlaute in Großschrift umgewandelt werden, kann es zu Konflikten mit Sondertasten kommen.

UPDATE (--) 83

Der zuletzt mit **BLOCK** oder **BUFFER** angeforderte Diskpuffer wird als geändert markiert. Derart markierte Puffer werden vor dem Überschreiben zum File zurückgeschrieben. Da hier nur ein Flag gesetzt wird, ist der Befehl so schnell, daß er (z.B. bei einem Datenbankprogramm) auch mehrfach eingesetzt werden kann.

UPPER (addr len --)

In dem angegebenen String werden alle Kleinbuchstaben einschließlich Umlaute in Großschrift umgewandelt. Dieser Befehl wird im KK-FORTH nur von FIND vor dem Vergleich einer Eingabe mit der Befehlsliste verwendet, wenn die Variable **CAPS** einen Wert ungleich 0 hat.

USER (name ; --)
(-- addr) bei Aufruf von name

Der Definitionsbefehl **USER** übernimmt den nächsten Namen als neue USER-Variable mit dem im SYSVAR-Bereich gespeicherten Offset (siehe dort unter UDP). Der Offset wird danach um zwei erhöht. Ein Fehler wird ausgegeben, falls kein Name hinter **USER** folgt oder kein Speicher mehr im USER-Bereich verfügbar ist. Da die Adresse erst bei Ausführung des Programmes abhängig von der Anfangsadresse des aktuellen Tasks berechnet wird, hat jeder Task einen eigenen Satz von USER-Variablen. Deshalb können auch alle Tasks gleichzeitig einen Ausgabestring erzeugen oder unterschiedliche Ausgabemedien mit den Ausgabebefehlen ansprechen.

UTABLE: (name ; len user --)

Der neu für das KK-FORTH definierte Befehl **UTABLE:** dient zusammen mit **UVECTOR** zur Definition und einfachen Umleitung von Ein-/Ausgaben. Bei Aufruf des mit **UTABLE:** definierten Befehls wird die angegebene USER-Variable auf die nachfolgende Tabelle umgestellt und der erste Befehl in der Liste zur Initialisierung aufgerufen. Durch Angabe der Länge der nachfolgenden Befehlsliste werden Programmabstürze durch Aufruf der nicht definierten Einsprünge verhindert. Dabei wird die Initialisierungsroutine nicht gezählt. **UTABLE:** verhält sich wie **:** und muß deshalb mit **;** abgeschlossen werden. Es ist darauf zu achten, daß nur Befehle, aber keine Literals oder Sprünge verwendet werden.

Beispiel:
```
5 INPUT UTABLE: Standard-Input
  NOOP  ((KEY?  (KEY  ((STRING  ((EDITSTRING  ((QUERY ;
```

UVECTOR (name ; # user -- ???)

Der neu für das KK-FORTH neu definierte Befehl **UVECTOR** dient zur Definition von Befehlen, die mittels **UTABLE:**-Befehle schnell umgeleitet werden können. Dazu wird eine USER-Variable definiert und die entsprechenden Befehle definiert. Bei der Definition muß neben der USER-Variable auch die Nummer des Befehls in der Liste angegeben werden.

Beispiel:
```
User input
1 input  UVector  key?
2 input  UVector  key
3 input  UVector  string
4 input  UVector  editstring
5 input  UVector  query
```

V, (w --)

Der 16Bit-Wert w wird dem Variablenbereich angehängt. Dazu ist eine Verschiebung des gesamten Variablenbereiches notwendig.

VALIGN (--)

Bei einigen Prozessoren kann der 16Bit-Zugriff nur auf geraden Adressen erfolgen. **VALIGN** korrigiert den Offset des Variablenbereiches entsprechend.

VALLOT (n --)

Der Variablenbereich wird um n Adressen verändert. Meistens ist dabei auch eine Verschiebung des Datenbereiches notwendig.

VARIABLE (name ; --) 83
(-- addr) bei Aufruf von name

Der Definitionsbefehl **VARIABLE** übernimmt den nächsten Namen als neue 16Bit-Variable und reserviert dafür auch den nötigen Speicherplatz. Da alle Tasks auf die gleiche Variablenadresse zugreifen, können Variablen zur Übergabe von Daten oder Flags verwendet werden. Ist es notwendig, so führt **VARIABLE** ein **VALIGN** aus.

VC, (char --)

Das niederwertige Byte von char wird dem Variablenbereich angehängt. Meistens wird dabei eine Verschiebung des Variablenbereiches durchgeführt.

VCREATE (name ; --)

In Anlehnung an **CREATE** wird ein neuer Befehlsname angelegt und der Offset in den Variablenbereich gespeichert. Bei Ausführung des neu definierten Befehls wird automatisch die Variablenadresse auf den Stack gebracht.

Beispiel:
```
: Variable   ( -- )
   VCREATE  0 v, ;
```

VDOES> (-- addr) I,R
(C: --)

In Anlehnung an **DOES>** dient VDOES> zur Verwaltung von Datenfelder, die im Variablenbereich abgelegt wurden. Immer wenn veränderbare Daten in einer ROM-Version verwendet werden soll, müssen statt der im Standard angegebenen Befehle **CREATE , DOES>** , **ALLOT** , **ALIGN** , **C,** und **,** die für den Variablenbereich definierten Befehle **VCREATE** , **VDOES>** , **VALLOT** , **VALIGN** , **VC,** und **V,** verwendet werden.

Beispiel:
```
: Feld   ( x y -- ) ( x y -- addr )
 valign  VCreate  over v,  * cells vallot
         VDoes>   dup >r @  *  + 1+ cells  r> + ;
```

VDP (-- sys) SV

Die Systemvariable **VDP** enthält die aktuelle Anfangsadresse des Variablenbereiches. Bei Veränderung des Heaps oder des Variablenbereiches wird auch diese Variable verändert.

VHERE (-- addr)

VHERE liefert die Endadresse+1 des Variablenbereiches. Sie kann für das nachträgliche Löschen des im Variablenbereiches reservierten Speichers genutzt werden.

Beispiel:
```
VCreate meinfeld  $100 allot  vhere $100 -  $100 erase
```

VLEN (-- sys) SV

Die Systemvariable **VLEN** enthält die Länge des belegten Variablenbereiches. Dieser Wert wird bei allen Variablendefinitionen als aktueller Offset gespeichert und entsprechend erhöht.

VOC-LINK (-- sys) SV

Die Systemvariable **VOC-LINK** zeigt auf die Parameterfeldadresse des zuletzt definierten Vokabulars. Dort stehen dann ein Zeiger auf die Parameterfeldadresse des nächsten Vokabulars und ein Offset in den Variablenbereich, wo dann der Zeiger auf die Linkfeldadresse des ersten Befehls in diesem Vokabular gespeichert ist. Enthält einer der beiden Zeiger den Wert 0, so ist das Ende der Kette erreicht.

VOCABULARY (name ; --) 83
(--) bei Aufruf von name

Der Definitionsbefehl **VOCABULARY** übernimmt den nächsten Namen als neues Vokabular. Bei Aufruf von **name** wird dann dieses Vokabular als erstes durchsucht. Dazu wird die Adresse des Offsets in den Variablenbereich (siehe auch **VOCABULARY** oder **VOC-LINK**) in den veränderlichen Teil von **CONTEXT** geschrieben.

VOCS (--)

Eine Liste aller im KK-FORTH definierter Vokabular wird ausgegeben. Dabei beginnt die Ausgabe mit dem zuletzt definierten Vokabular. Falls der Befehlsname eines Vokabulars nicht mehr gefunden werden kann (z.B. bei headerloser Definition), dann wird ??? ausgegeben.

WDP@ (-- addr)

Abweichend vom Standard-FORTH hat jeder Task einen eigenen Arbeitsbereich, aber kein eigenes Dictionaryende **HERE** . Deshalb wird die Anfangsadresse des Arbeitsbereiches durch **WDP@** statt durch **HERE** geliefert. Auch die zur Zahlenstringumwandlung und der Stringablage verwendete Adresse **PAD** liegt oberhalb von **WDP@** (meist &84 Adressen darüber).

WHILE (f --) 83 I,R
(C: addr1 2 -- addr2 -2 addr1 2)

Innerhalb von **BEGIN** ... **REPEAT** oder **BEGIN** ... f **UNTIL** kann **WHILE** beliebig oft zum Test eines Flags eingefügt werden. Wenn dieses Flag 0 ist, wird sofort zum Ende der Schleife gesprungen. Dazu kompiliert **WHILE** nach Test des Kontrollflags 2 einen bedingten Sprung und legt dessen Adresse zusammen mit dem Flag -2 unter der von **BEGIN** abgelegten Rücksprungadresse addr1.

WITHIN (w w1 w2 -- f)

Ein TRUE-Flag wird zurückgeliefert, wenn w1¾w<w2 ist. Da in **WITHIN** nur die Differenzen w1-w und w2-w mittels **U<** getestet wird, können sowohl vorzeichenlose als auch vorzeichenbehaftete 16Bit-Werte verwendet werden.

WORD (-- wdp)

Der nächste durch Leerzeichen begrenzte String wird aus dem Eingabepuffer geholt, mit Längenangabe zu **WDP@** gebracht und noch eine ASCII-Null an den String angehängt. Falls der restliche Eingabestring aus Leerzeichen besteht oder die Länge 0 hat, steht in wdp ebenfalls nur die Länge 0.

WORDS (--)

Die Befehlsliste des obersten (veränderlichen) Vokabulars in **CONTEXT** wird ausgegeben. Befehle mit Header auf dem Heap werden durch | vor dem Befehlsname gekennzeichnet. Die Ausgabe kann mit beliebiger Taste angehalten und mit #BRK-Taste abgebrochen werden.

XOR (w1 w2 -- w3)

Es wird eine bitweise XOR-Verknüpfung der beiden 16Bit-Werte w1 und w2 durchgeführt und das Ergebnis w3 wieder auf dem Stack abgelegt. Nur wenn die entsprechenden Bits beider Werte ungleich sind, wird das Bit im Ergebnis w3 auf 1 gesetzt.

Beispiel:

```
      1011110010101101
XOR   0100101001000100
      ----------------
      1111011011101001
```

[(--) I

Durch Umsetzen des DEFER-Befehls **PARSER** auf **INTERPRETER** wird der Interpretermodus aktiviert. Zusätzlich wird die USER-Variable **STATE** auf 0 gesetzt.

['] (-- cfa) I,R
(C: name --)

Die Codefeldadresse des nach **[']** folgenden Befehlsnamen wird als Literal in die aktuelle Definition übernommen. Eine Fehlermeldung wird ausgegeben, wenn der Befehl nicht gefunden werden kann.

[COMPILE] (name ; --) I,R

Die Codefeldadresse des nach **[COMPILE]** folgenden Befehlsnamen wird in die aktuelle Definition kompiliert. **[COMPILE]** wird zur Kompilierung von Immediate-Wörter in Strukturbefehle verwendet, kann aber meistens durch **POSTPONE** ersetzt werden.

Beispiel:
```
: ENDIF   ( Redefinition von  THEN )
   [COMPILE] THEN ;   IMMEDIATE RESTRICT
```

\ (--) I

Der Rest der hinter **** folgenden Zeile wird als Kommentar betrachtet und übersprungen. Nach dem **** muß immer zuerst ein Leerzeichen folgen.

\NEEDS (name ; --)

Ist der nach **\NEEDS** folgende Befehlsname schon in der aktuellen Befehlsliste verfügbar, wird der Rest der Zeile wie bei **** übersprungen. **\NEEDS** dient zum Nachladen von Befehlsgruppen wie den Assembler, die nur einmal im System benötigt werden.

Beispiel:
```
\NEEDS ASSEMBLER    Include ASM8086.SCR
```

\\ (--) I

Der Rest des Screens oder der Eingabezeile wird übersprungen. Meistens werden mit **** einzelne Screens oder Kommentare im unteren Teil eines Screens ausmaskiert.

] (--)

Durch Umstellung des DEFER-Vektors **PARSER** auf **COMPILER** wird der Kompiliermodus aktiviert. Als Kennung wird zusätzlich die USER-Variable **STATE** auf -1 gesetzt.

| (--)

Falls die Variable **>HEAD?** auf 0 steht, wird sie um 1 erhöht. Dadurch wird das KK-FORTH bei der nächsten Befehlsdefinition veranlaßt, den Befehlsheader auf dem Heap abzulegen. Da dabei **>HEAD?** wieder erniedrigt wird, dient **|** zur headerlosen Kompilierung des nächsten Befehls.

Beispiel:
```
| Variable test
  : reset   0 test ! ;
  : inc     1 test +! ;
  : val.    test ? ;
```

Anhang C

Erklärung der Fachwörter

In diesem Anhang sollen einige der vielen Fachwörter erklärt werden. Die meisten davon werden auch außerhalb von FORTH in der Informatik verwendet.

Block

Die Verwaltung von Screen-File geschieht in sogenannten Blocks oder Screens. Ein Block ist dabei ein 1024 Bytes großes Segment, daß nur am Stück von Diskette in den Speicher geladen oder wieder auf Diskette geschrieben wird. Für die Programmeingabe wird dann dieser Block als eine Seite mit 16 Zeilen zu je 64 Zeichen dargestellt.

Codefeldadresse

Die Codefeldadresse oder kurz CFA enthält einen Zeiger auf die Speicheradresse der zur Ausführung des Befehls notwendigen Assemblerprogrammes. Bei INDIRECT-Befehlen enthält die CFA nur einen Zeiger auf das tatsächliche Codefeld.

Compiler siehe Kompiler

Counted-String

Im KK-FORTH werden alle Strings als sogenannte "Counted-Strings" mit angehängtem 0-Ende gespeichert. Ein "Counted-String" besteht dabei aus einem Längenbyte und dem eigenlichen Zeichenstring. Durch den Befehl **COUNT** kann dann aus der "Counted-String-Adresse" (kurz csa) sowohl Adresse als auch Länge des Strings ermittelt werden.

FORTH

FORTH sollte eigendlich FOURTH , also Programmiersprache der 4. Generation heißen. Da aber der verwendete Rechner nur 5 Buchstaben für Filenamen zugelassen hat, wurde es zu FORTH verstümmelt. Gleichzeitig sollte dadurch auf die andere Bedeutung von FORTH (vorwärts, weiter) angespielt werden. Als Befehl ist **FORTH** der Name des Standard-Vokabulars, in dem alle KKF-Kernbefehle aufbewahrt werden.

Glossar

Das Glossar ist eine Zusammenstellung aller in FORTH verfügbarer Befehle. Im Glossar ist neben dem Befehlsname meistens die Veränderung der Stacks und eine Beschreibung des Befehls zu finden.

Interpreter

Ein Interpreter holt einen Befehl und führt ihn sofort aus.

Kompiler

Anders als beim Interpreter wird beim Kompiler der Befehl normalerweise nicht ausgeführt, sondern in der aktuellen Definition gespeichert. Ausnahmen davon sind die sogenannten IMMEDIATE-Befehle.

Literal

Ein Literal ist in FORTH eine Konstante, die in ein Programm übernommen wird. Dies geschieht durch den Befehl **LITERAL** , der meistens nach einer eckigen Klammer verwendet wird und einen 16Bit-Wert als sogenanntes Inline-Literal in das Programm übernimmt.

MCB

Der "Memory-Control-Block" dient zur Verwaltung des angeforderten Speichers außerhalb des KKF-Systems. Aufbau und Anwendung sind Systemspezifisch.

Namensfeldadresse

Ab der Namensfeldadresse oder kurz NFA wird zuerst das Längenbyte mit drei Flags und der Befehlsname abgelegt. Zur Erleichterung der Eingabe werden bei gesetzten **CAPS** alle Befehle in Großbuchstaben gespeichert.

.i.NOS;

Der "Next of Stack" ist entweder der Wert oder die Position des zweiten Datenstackeintrages.

Parameterfeldadresse

Nach der Codefeldadresse folgt die Parameterfeldadresse oder kurz PFA, ab der die Daten des Befehles abgelegt werden.

Screen Siehe BLOCK

String

Ein String ist eine Kette von Zeichen. Sie wird im KK-FORTH entweder als Counted-String verwaltet oder durch Anfangsadresse und Länge spezifiziert.

TIB

Der "Terminal-Input-Buffer" ist ein für jeden Task getrennt verfügbarer Spiecher oberhalb des USER-Bereiches. Er wird automatisch von **QUIT** zur Ablage der eingegebenen Befehlszeilen verwendet.

TOR

Der "Top of Returnstack" ist entweder der Wert oder die Position des obersten Returnstackeintrages.

TOS

Der "Top of Stack" ist entweder der Wert oder die Position des obersten Datenstackeintrages.

Zelle

Eine Zelle ist der benötigte Speicher für einen 16Bit-Wert. Bei byteadressierten Speicher werden dazu zwei Adressen benötigt.

Zweierkomplement

Im Computer sind mehrere Arten der Darstellung für negative Zahlen möglich. Hier im KK-FORTH wird das sogenannte Zweierkomplement verwendet. Bei negativen Werten ist dabei immer das höchste Bit gesetzt. Beim Negieren der Werte müssen zuerst alle Bits invertiert werden und dann zum Ergebnis noch 1 addiert werden. Der Vorteil dieser Berechnungsart ist, daß sowohl vorzeichenbehaftete als auch vorzeichenlose Addition und Subtraktion mit den gleichen Routinen durchgeführt werden kann.

Beispiel:

```
 2 = %0000000000000010
 1 = %0000000000000001
 0 = %0000000000000000
-1 = %1111111111111111
-2 = %1111111111111110
```

Anhang D

Fehlerliste

Fehlernummer	Art
$0000	Kein Fehler (ERROR entfernt nur den Wert)
Bit 15=1 : Kompilermodus verlassen und Datenstack löschen	
$0001 ...	Fehler des Betriebssystem (hier PC)
$0001	Unbekannte Funktionsnummer
$0002	File nicht gefunden
$0003	Pfad nicht gefunden
$0004	Zu viele Files offen
$0005	Zugriff verweigert
$0006	Unbekannte Handlenummer
$0007	MCB (Speicherverwaltung-Datenblock) zerstört
$0008	Kein Speicher verfügbar
$0009	Unbekannter MCB
$7e01 ...	Warnungen oder KKF-Meldungen
$7e01	Datenstack-Unterlauf
$7e02	" exist"
$7e03	" Include : "
$7e04	" End-Include : "
$7f01 ...	KKF-Fehlernummern
$7f01	Datenstack-Unterlauf
$7f02	Datenstack-Überlauf
$7f03	Returnstack-Unterlauf
$7f04	Returnstack-Überlauf
$7f05	Zu wenig Parameter
$7f06	Unerlaubter Wert
$7f07	Arithmetik-Überlauf
$7f08	Nicht initialisierter Interrupt
$7f09	Fehlerhafte Adresse
$7f0a	DEFER wurde nicht definiert
$7f0b	Dictionary ist voll
$7f0c	USER-Bereich ist voll
$7f0d	CONTEXT-Bereich ist voll
$7f0e	DP liegt im HEAP
$7f0f	Befehl ist geschützt
$7f10	Name erwartet
$7f11	Name nicht gefunden
$7f12	Es wurde kein Befehl definiert
$7f13	Befehl ist nur in :-Definitionen zulässig
$7f14	Fehlerhafte Kontrollstruktur
$7f15	Es folgte kein DEFER-Befehl nach IS
$7f16	Kein File geöffnet
$7f17	Blocknummer zu groß
$7f18	Blocknummer nicht erlaubt
$7f19	Fehler bei Terminal-Befehlsübertragung
$7f1a	Fehler bei UVECTOR-Befehl (Tabelle zu kurz)
$7f1b	Befehl wird nicht unterstützt
$7fff	Fehlermeldung als CSA liegt auf dem Datenstack

Anhang E

Terminalbefehle

F1
ALT+H Anzeige der Hilfsinformation zur Tastenbelegung

ALT+P Ein-/Ausschalten des Druckers. In der unteren Statuszeile wird bei aktivem Drucker "P" ausgegeben. Da die Ausgabe parallel zum Bildschirm erfolgt, bleibt das Terminalprogramm stehen, bis der Drucker bereit ist. Es können aber trotzdem noch Zeichen empfangen werden.

ALT+L Beim Arbeiten mit dem Terminal können alle empfangenen Zeichen auch in ein Logfile geschrieben werden. Dadurch kann, wie beim Erstellen dieser Beschreibungen, das Arbeiten mitprotokolliert werden. Nach Drücken von ALT+L muß der Name des Files (Vorgabe: KKF.LOG) eingegeben werden. Dieses File wird dann mit Länge 0 geöffnet und alle danach empfangenen Zeichen darin gespeichert. Falls beim Speichern ein Fehler auftritt (Diskette voll oder nicht beschreibbar), so wird das Logfile geschlossen. Es kann aber auch durch erneute Betätigung von ALT+L geschlossen werden. In der Statuszeile wird dann das "L" wieder gelöscht.

ALT+D Das Directory des aktuellen Verzeichnis wird bei ALT+D angezeigt. Weder auf dem Drucker noch im Logfile sind diese Ausgaben sichtbar.

ALT+S Aufruf der COMMAND-Oberfläche des Betriebssystems. Diese Funktion erlaubt danach die Eingabe von DOS-Befehlen. Da aber das Terminalprogramm weiterhin im Speicher steht, dürfen weder die verwendeten Files noch die aktive Schnittstelle durch diese Befehle verändert werden. Nach der Eingabe von EXIT kehrt man wieder in das Terminalprogramm zurück.

ALT+X Terminalprogramm beenden. Dabei sollten alle vom KK-FORTH verwendeten Files geschlossen sein. Zur Sicherheit wird vorher abgefragt, ob das Programm wirklich verlassen werden soll.

ALT+Q Tasteneingaben können die Befehlsübertragung zwischen KK-FORTH und Terminalprogramm stören. Deshalb kann dies durch ein Kommando ($0002) oder über Tastatur verhindert werden. Bei blockierter Tastatur wird ein "X" in der Statuszeile angezeigt und die gedrückte Taste gespeichert. Bei Umstellung von Port oder Baudrate wird auch der Tastaturpuffer gelöscht.

ALT+C Die Nummer des COM-Ports kann nach ALT+C verändert werden. Dabei sind folgende Portadressen möglich:

COM1: $03F8	COM2: $02F8
COM3: $03E8	COM4: $02E8

ALT+B Die Baudrate kann beim PC-Terminalprogramm von 300 bis zu 115200 Baud verändert werden. Die Tasten 0 bis 9 sind dabei mit folgenden Baudraten belegt:

0:	115200	1:	57600	2:	38400
3:	19200	4:	12800	5:	9600
6:	2400	7:	1200	8:	600
9:	300				

ALT+T Um möglichst ohne Veränderung der unbenutzten Schnittstellen und ohne dauernde Umstellung der Vorgaben auszukommen, kann das aktuelle System unter einem beliebigen Manen abgespeichert werden.

ALT+E Durch Drücken von ALT+E kann der Empfangsspooler gelöscht werden. Dadurch werden die schon empfangenen Zeichen ignoriert und nicht mehr ausgegeben oder gespeichert.

Anhang F

Editor-Tastenbelegung

Cursorsteuerung:

^E oder Cursor_hoch	Eine Zeile höher
^X oder Cursor_tief	Eine Zeile tiefer
^S oder Cursor_links	Ein Zeichen links
^D oder Cursor_rechts	Ein Zeichen rechts
TAB	Zur nächsten 4er-Teilung
Shift+TAB	Zur vorherigen 4er-Teilung
^F	Zum nächsten Wortanfang
^A	Zum vorherigen Wortanfang
^Q B	Zum Zeilenanfang
^Q K	Zum Zeilenende-1
^Q E	Zum Screenanfang
^Q X	Zum Screenende-1
POS1	Zum ersten Zeichen der Zeile
ENDE	Hinter das letzte Zeichen der Zeile
^POS1	Zum ersten Zeichen des Screens
^ENDE	Hinter das letzte Zeichen des Screens
Return	Zum nächsten Zeilenanfang
^R oder Bild_hoch	Zum vorhergehenden Screen
^C oder Bild_tief	Zum nächsten Screen
^K R oder ^Bild_hoch	Zum ersten Screen
^K C oder ^Bild_tief	Zum letzten Screen
^Q G	Eingabe der gewünschten Screennummer
F4	Zum zweiten File / Cursorposition umschalten

Zwischenspeicherung von Zeichen und Zeilen:

F1	Zeichen speichern und löschen
F2	Zeichen speichern, Cursor rechts
F3	Zeichen einfügen
F5	Zeile speichern und löschen
F6	Zeile speichern, Cursor in die nächste Zeile
F7	Zeile einfügen

Steuerung der Zeicheneingabe und des Löschens:

^V	Insert-Modus umschalten (Anzeige: O oder I)
Einfg	Ein Leerzeichen einfügen
^G oder Entf	Zeichen unter dem Cursor löschen
^H oder Backspace	Zeichen vor dem Cursor löschen
F8	Rest der Zeile löschen
^Y	Aktuelle Zeile entfernen
^N	Leerzeile einfügen
^K N	Leerscreen einfügen
^K Y	Aktuellen Screen entfernen
^Backspace	Nächste Zeile ab Cursorposition übernehmen
^Return	Rest dieser Zeile in die nächste Zeile

Suchen und Ersetzen:

^Q F	String suchen (u=Option für Rückwärts-Suche)
^Q A	String suchen und ersetzen (mehrere Optionen)
^T	Nächstes Wort in Kleinschrift wandeln
^U	Nächstes Wort in Großschrift wandeln

Speicherung:

F10	Änderungen in diesem Screen rückgängig machen
F9	Eingabe der ID-Kennung (nicht bei EDITOR.COM)
ESC	File speichern, Editor verlassen

Zusätze bei EDITOR.COM:

^K S	File speichern, danach weiter editieren
^K D	Editor ohne Speicherung des Files verlassen

Anhang G

Programmlistings

ERRTRAP.SCR

```
         Screen # 0  - ERRTRAP.SCR
 0 \\ Errortrapping für KK-FORTH                  ( 27.07.91/KK )
 1
 2 System:         KK-FORTH V1.2/0
 3 Änderung:       27.07.91  KK: File an KKF angepaßt
 4
 5                      Hinweise:
 6  - Verwendet USER-Variablen ERP1 und ERP2 als Zeiger
 7  - Aufbau eines Error-Returnstack:
 8    ERP2 + 2 --> Wiedereinsprungsadresse bei Fehler
 9    ERP2 + 1 --> Zeiger auf nächsten ERP-Eintrag oder 0
10    EPR2 + 0 --> ERP-UNLINK ( löscht die beiden oberen Werte )
11  - Innerhalb von  { ... } wird vorletzter Rücksprung verwendet
12  - Beispielprogramm auf Screen 5
13
14
15


         Screen # 1  - ERRTRAP.SCR
 0 \ LOADSCREEN                                   ( 27.07.91/KK )
 1
 2   | User erp1                   \ 1. Zeiger für ERRORTRAP
 3   | User erp2                   \ 2. Zeiger für ERRORTRAP
 4   Defer 'errtrap   ' noop is 'errtrap
 5   Defer 'endtrap   ' quit is 'endtrap
 6
 7   &02 &04 thru                 \ Rest laden
 8
 9   &05 load                     \ Beispielprogramm
10
11
12
13
14
15


         Screen # 2  - ERRTRAP.SCR
 0 \ >*<                                          ( 27.07.91/KK )
 1 | Create -erp   ( Löscht letzten ERP-Eintrag )
 2   ] erp2 @  erp1 !  r> erp2 !  ( oberster ERP-Eintrag löschen )
 3     rdrop  exit [                        ( Einsprung löschen )
 4
 5   : >*<     ( -- ; R: -- erp -erp )
 6     r@                 ( Rücksprungadresse dabei aber merken )
 7      erp2 @ >r  erp1 @  erp2 !                   ( ERP-Eintrag )
 8      -erp >r                               ( Korrekturbefehl )
 9      rp@ erp1 !                          ( RStack-Tiefe merken )
10     >r  0 ;                                              restrict
11
12
13
14
15
```

```
         Screen # 3  - ERRTRAP.SCR
 0 \ {  }                                          ( 27.07.91/KK )
 1   : {       ( -- ; R: -- erp ) ( Letztes ERP ignorieren )
 2     r>
 3      erp1 @  >r                                ( ERP1 merken )
 4      erp2 @  erp1 !                ( EPR2 ist aktueller Wert )
 5     >r ; restrict
 6
 7   : }       ( -- ; R: -- erp )
 8     r>
 9      r> erp1 !                           ( ERP1 restaurieren )
10     >r ; restrict
11
12
13
14
15


         Screen # 4  - ERRTRAP.SCR
 0 \ (start  (errortrap  errortrap                 ( 27.07.91/KK )
 1 | : (start      ( ... -- )
 2     'errtrap   dclear  erp1 @  ?dup
 3     IF   rp!                        ( Returnstack zurückstellen )
 4          2r>  r@ -rot  2>r  >r      ( Wiedereinsprung als TOR )
 5          1                         ( Flag für Wiedereinsprung )
 6     ELSE 'endtrap         ( Keinen Rücksprung: QUIT ausführen )
 7     THEN ;
 8
 9 | : (errortrap  ( error | csa -1 | csa $7fff -- )
10     errortext@
11     cr  type  cr
12     (start ;
13
14   : errortrap   ( -- ) ( Initialisierung )
15      erp1 off  erp2 off  ['] (errortrap  errorhandler  ! ;


         Screen # 5  - ERRTRAP.SCR
 0 \ demo                                          ( 27.07.91/KK )
 1   : demo1       ( -- ) ( Beispielwort 1 )
 2     >*< ." Wort : DEMO1 ausgeführt" .s cr
 3     IF cr ." Verlasse DEMO1" cr  exit THEN      ( Nach Fehler )
 4     $4001 error ;                   ( Demo-Fehlernummer $4001 )
 5
 6   : demo2       ( -- ) ( Beispielwort 2 )
 7     >*< ." Wort : DEMO2 ausgeführt" .s cr
 8     IF cr ." Verlasse DEMO2" cr  exit THEN      ( Nach Fehler )
 9     { $4002 error } ;               ( Demo-Fehlernummer $4001 )
10
11   : demo        ( -- ) ( Hauptbefehl )
12     $1234 cr  >*< ." Wort : DEMO  ausgeführt" .s cr
13     IF  demo1  cr ." Verlasse DEMO" cr  exit THEN
14         demo2  cr ." -------------" cr ;
15
```

FFT.SCR

```
         Screen # 0 - FFT.SCR
 0 \\ FFT-Analyse für KK-FORTH V1.2               ( 27.07.91/KK )
 1
 2 System:         KK-FORTH V1.2
 3 Änderung:       27.07.91  KK: Anpassung an KKF
 4
 5                     Hinweise:
 6  - FFT-Analyse von Signalen mit 32 .. 4096 Punke
 7  - Verwendet Speicherbereich zwischen  HERE  und  VDP @
 8  - Punkteanzahl und deren Zweierlogarithmus in Screen # 2
 9    ( nur durch Speicherbedarf beschränkt )
10
11  - Beispiel im letzten Screen
12
13
14
15


         Screen # 1 - FFT.SCR
 0 \ Loadscreen                                   ( 27.07.91/KK )
 1
 2 \ Einige Systemkonstanten (veränderbar, solange Speicher reicht)
 3       8 Constant ld(p           \ 2^8 Punkte
 4   &0256 Constant points         \ sind 256
 5
 6 \ Rest des Programmes laden
 7   &02 &08 thru                  \ FFT-Analyse
 8   &09 load                      \ Beispielbefehl
 9
10
11
12
13
14
15


         Screen # 2 - FFT.SCR
 0 \ Variablen und Datenfelder                    ( 27.07.91/KK )
 1   Variable lz           \ Endwert des aktuellen Exponenten
 2   Variable nz           \ Endwert der aktuellen Meßwertnummer
 3   Variable zx           \ 1. Meßwertnummer
 4   Variable zy           \ 2. Meßwertnummer
 5
 6   : re(n        ( n -- addr ) ( Realteil )
 7               2*  cells  here + ;
 8   : im(n        ( n -- addr ) ( Imaginärteil )
 9     points +  2*  cells  here + ;
10
11
12
13
14
15
```

```
         Screen # 3  - FFT.SCR
 0 \ sin  cos                                        ( 27.07.91/KK )
 1 | : (sp     ( y -- sign &16383 0 | sign y -1 )
 2     dup abs  dup &16384 =
 3     IF   drop  &16383 0
 4     ELSE &16384  dup rot -  abs  -  -1
 5     THEN ;
 6   : sin     ( 0..16384..32767..49152 -- 0..32767..0..-32767 )
 7     (sp IF  2* 2* &25728 over dup um* nip
 8             &10518 over &1173     um* nip -
 9             um* nip -   um* nip
10          THEN swap ?negate ;
11   : cos     ( 0..16384..32767..49152 -- 32767..0..-32767..0 )
12     &16384 +  sin ;
13 \       y*4              y^2*16                    y^2*16
14 \ z = ----- * ( 25728 - ------- * ( 10518 - 1173 * ------- ) )
15 \      65536             65536^2                   65536^2


         Screen # 4 - FFT.SCR
 0 \ sqrt                                            ( 27.07.91/KK )
 1 ( Quadratwurzel einer 32-Bit-Integerzahl                        )
 2 ( Zulässiger Bereich: $0XXX.XXXX --> 0..16383 ( 14 Bit )
 3   : sqrt2*      ( ud -- u )
 4     0  0                         ( Summe und Wurzel )
 5     &15                          ( 16 mal )
 6     FOR  >r  2* over 0< -  2* over 2* 0< -      ( 2 Bits in S )
 7             >r  d2*  d2*  r> r>                 ( Quadrat*4 )
 8          2* 1+                   ( Wert*2+1 )
 9          2dup u<                 ( Summe kleiner Wurzel ? )
10          IF   1-                 ( dann nur Wert*2 )
11          ELSE dup >r - r> 1+     ( oder Summe=Differenz )
12          THEN
13     NEXT
14     nip nip nip ;
15


         Screen # 5 - FFT.SCR
 0 \ round  bitrev  reversel                         ( 27.07.91/KK )
 1   : round       ( d -- n )
 2     dup &32767 xor IF over 0< IF 1+ THEN THEN nip ;
 3   : bitrev      ( a -- a2 ) ( Bitumkehr )
 4     0  swap  ld(p 1-
 5     FOR  2*  tuck  points and  or  2/   swap  NEXT
 6     drop ;
 7   : reversel    ( -- )
 8     points 1-
 9     FOR  r@ bitrev  dup r@ u<
10        IF  r@ over >r  >r    r@ re(n  2@   r> im(n  2@
11            r@ im(n   >r   r@ 2@ 2swap  r> 2!  2swap
12            r> re(n   >r   r@ 2@ 2swap  r> 2!
13                           r@ re(n  2!   r@ im(n  2!
14        THEN drop
15     NEXT ;
```

```
         Screen # 6 - FFT.SCR
 0 \ Komplexe Multiplikation                          ( 27.07.91/KK )
 1   : kmu          ( -- )
 2     zx @    lz @ ?dup IF 1- FOR 2/ NEXT THEN        ( ZX/2^LZ )
 3     bitrev  [ 15 ld(p - ] Literal FOR 2* NEXT         ( --> P )
 4     dup sin swap cos                    ( Sinus u. Cosinus )
 5     2dup  zy @ re(n 2@ round m*
 6       rot zy @ im(n 2@ round m* d+ d2*                  ( S )
 7     2swap zy @ im(n 2@ round m*
 8       rot zy @ re(n 2@ round m* d- d2*                  ( T )
 9     zx @ im(n 2@ d2/  2over  d-  zy @ im(n 2!         ( IM(Y )
10     zx @ im(n 2@ d2/         d+  zx @ im(n 2!         ( IM(X )
11     zx @ re(n 2@ d2/  2over  d-  zy @ re(n 2!         ( RE(Y )
12     zx @ re(n 2@ d2/         d+  zx @ re(n 2! ;       ( RE(X )
13
14
15


         Screen # 7 - FFT.SCR
 0 \ FFT-Hauptroutine                                 ( 27.07.91/KK )
 1   : kfft         ( -- )
 2     points  nz !   ld(p  lz !   ld(p 1-
 3     FOR  0  zx !   nz @  2/  nz !   lz @  1-  lz !
 4        BEGIN  nz @ 1-
 5           FOR   zx @  nz @  +   zy !     ( ZY = ZX + NZ )
 6                 kmu                      ( Berechnung )
 7                 zx @  1+  zx !           ( ZX = ZX + 1 )
 8           NEXT zy @  1+  dup zx !   points u< 0=
 9        UNTIL
10     NEXT
11     reversel ;
12
13
14
15


         Screen # 8 - FFT.SCR
 0 \ FFT                                              ( 27.07.91/KK )
 1 \ Signal im re(n    ( 16-Bit-Werte, wird auf 32 Bit erweitert )
 2 \ wird in das Spektrum umgewandelt und zurückgeschrieben
 3   : fft          ( -- )
 4     0 im(n  points 2* cells  erase     ( Imaginärteil löschen )
 5     points 1-                    ( Realteil * 32768 )
 6     FOR  r@ re(n @  0 swap  d2/  r@ re(n 2!  NEXT
 7     kfft
 8     points 2/                     ( Amplitude aus Komplexwert )
 9     FOR  r@ re(n 2@ d2/ round dup  m*
10          r@ im(n 2@ d2/ round dup  m*  d+  sqrt2* 2* 2*
11          0  r@ re(n 2!   0.  r@ im(n 2!   ( Werte in Realteil )
12     NEXT points 2/  dup 1+ re(n  swap 1- 2* cells erase
13          points 2/  dup 1+ im(n  swap 1- 2* cells erase
14          0 re(n 2@  d2/  0 re(n 2! ;             ( DC-Anteil )
15
```

```
         Screen # 9 - FFT.SCR
 0 \ Beispiel für FFT-Analyse                        ( 27.07.91/KK )
 1   : ffttest     ( -- ) ( Oberwellen analysieren )
 2     &65536. points um/mod  nip              ( Grundschwingung )
 3     points 1-
 4     FOR   &1000                                 (  DC-Anteil    )
 5           over r@ * 2 * sin 2/    +             ( + 8192*SIN[2] )
 6           over r@ * 5 * cos 4 /   +             ( + 4096*SIN[5] )
 7           over r@ * 6 * sin 8 /   +             ( + 2048*COS[6] )
 8           r@ re(n !                             (  speichern    )
 9     NEXT drop
10     fft                                         ( FFT-Analyse )
11     cr cr ."      Analyse einer Schwingung: " cr
12     9 FOR  cr 9 r@ -  dup 3 u.r ."  .te Oberwelle : "
13                       re(n 2@ drop  6 u.r             NEXT cr
14   ;
15
```

Index