Klaus Kohl

Zusatzhandbuch

zum

KKF_PC

Version 1.2/0

- Hinweise zum KKF_PC

- Beschreibung der Zusatzprogramme

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-8905 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.

Das hier vorliegende Zusatzhandbuch dient, wie der Name schon sagt, als Ergänzung des Handbuches. Es beschreibt die in einzelnen KKF-Versionen voneinander abweichenden Befehlsparameter und -strukturen. Zusätzlich enthält es Hinweise zu den angepaßten Tools und zu den von der Hardware und dem Betriebssystem abhängigen Beispielprogrammen.

1.2 Das KK-FORTH für PC-Kompatible

Die Ausgangsbasis für alle KKF-Versionen ist das KKF_PC. Sowohl der Metakompiler als auch Editor und Terminalprogramm entstammen dieser KKF-Version.

Sowohl die Filestruktur als auch die Speicherverwaltung sind dem MSDOS entlehnt. Jedoch wurde bei der Definition der Befehle darauf geachtet, daß es ohne große Probleme auch für andere Betriebssysteme anpaßbar ist.

Neben dem KKF-Kern KKF_PC.COM , der nur die schon im Handbuch beschriebenen Befehle beinhaltet, gibt es auf der Diskette 1 ein File mit Namen KKF.COM . Es enthält neben Assembler, Disassembler, Debugger und Editor noch einige Befehle, um DOS-Befehle auszuführen (DIR, COPY ...) oder sogar auf eine eigene COMMAND-Oberfläche zu gelangen. Von dort aus können dann beliebige DOS-Befehle oder andere Programme ausgeführt werden. Nach der Eingabe von EXIT gelangt man dann wieder in das unveränderte KK-FORTH zurück.

1.3 Lieferumfang

Mit dieser Beschreibung erhalten Sie zwei Disketten (5.25" oder 3.5" - 360K). Die erste Diskette enthält das eigentliche KK-FORTH mit allen Zusatzfiles. Die zweiten Diskette wird sowohl zum KKF_PC als auch zu allen Terminalversionen mitgeliefert. Es enthält neben dem KKF_PC noch einen FORTH-Screeneditor und das Terminalprogramm.

Da außer dem KKF-Kern alle Sourcen mitgeliefert werden, können viele Vorgaben (z.B. Tastenbelegung) den eigenen Wünschen angepaßt werden.

KKF_PC-Diskette

```
READ     ME      2838   3.09.91   9.14
-KKF-PC- ---        0  21.12.90   9.34
KKF_PC   COM    16850  29.08.91  16.34  KKF-Kern
MAKEKKF  KKF      293  30.07.91  20.03  File zum Erstellen des KKF.COM
KKF      COM    35980  29.08.91  16.35  KKF mit Assembler ... Editor
--FILES- ---        0  21.12.90   9.34
ASM8086  SCR    21504   1.08.91  18.36  8086-Assembler
DIS8086  SCR    38912   1.08.91  13.35  8086-Disassembler
PC_DEBUG SCR    17408  10.08.91  13.58  8086-Debugger
PC_EXTRA SCR     8192  10.08.91  16.14  DOS-Tools
NECP6    SCR    21504  29.08.91   9.39  Druckertreiber für NEC-P6
PC_EDIT  SCR    43008   3.08.91  11.27  FORTH-Screeneditor
---HB--- ---        0  21.12.90   9.34
AUTO     SCR     2048  24.07.91   7.59  Autostart-Beispiel
ERRTRAP  SCR     6144  27.07.91  20.09  Errortrapping-Beispiel
FFT      SCR    10240  24.08.91  13.22  FFT-Analyse
SIEVE    SCR     3072  28.08.91  18.57  Sieb des Eratosthenes
--GRAFIK ---        0  21.12.90   9.34
G_BIOS   SCR    11264   3.09.91   9.09  BIOS-Grafikbefehle
G_VGA    SCR    20480   3.09.91   9.13  Grafik-Befehle für MDB10
GTEST    SCR     9216  24.08.91  11.06  VGA-Grafikmenü
PC_MOUSE SCR     6144  24.08.91  11.08  Ansteuerung einer Maus
```

KKF-Terminaldiskette

```
READ     ME      2732  29.08.91  16.46
-KKF-PC- ---        0  21.12.90   9.34
KKF_PC   COM    16850  29.08.91  16.34  KKF-Kern
ASM8086  SCR    21504   1.08.91  18.36  8086-Assembler
PC_EXTRA SCR     8192  10.08.91  16.14  DOS-Tools
NECP6    SCR    21504  29.08.91   9.39  Druckertreiber für NEC-P6
-MEMEDIT ---        0  21.12.90   9.34
MEM_EDIT SCR    47104  24.08.91  15.25  Source für FED.COM
FED      COM    23999  29.08.91  16.40  FORTH-Screeneditor
-TERMINA L--        0  21.12.90   9.34
PC_TERM  SCR    38912  29.08.91  16.43  Source für TERMINAL.COM
T_EXTRA  SCR     7168   2.08.91  14.19  Angepaßte DOS-Tools
T_EDIT   SCR    44032   5.08.91  23.20  Angepaßter Editor
FRAME    SCR    10240   9.08.91   8.14  Statuszeilen für Bildschirm
TERMINAL COM    28828  29.08.91  16.44  Terminalprogramm
```

1.4 Installation des KKF

Die erste Aktion nach dem Auspacken der Diskette sollte auf jeden Fall das Anbringen eines Schreibschutzes sein. Dadurch verhindert man sowohl eine ungewollte Veränderung der Sourcen als auch den Übergriff von Viren auf ddie Programme oder dem Bootsektor der Diskette.

Besitzt der PC/AT eine Festplatte, so richtet man am besten ein eigenes Unterverzeichnis mit Namen KKF_PC (bzw. KKF_TERM für zweite Diskette) ein. In dieses Unterverzeichnis kann man dann alle Files der ersten (zweiten) Diskette kopieren. Das Programm KKF_PC.COM bzw. KKF.COM sollte dann immer in diesem Unterverzeichnis gestartet werden, weil das FORTH keinen automatischen Pfadwechsel unterstützt.

Falls mit nur einer Diskette gearbeitet werden soll, so verwendet man entweder eine DISKCOPY-Kopie der ersten Diskette oder übernimmt alle Programme mit XCOPY auf eine größere Diskette (720K oder 1.2M). Das KKF_PC ist schnell genug, um sogar noch auf einem langamen Laptop vernünftiges Arbeiten zu erlauben. Dabei gab es auch keine Probleme mit anderen Betriebssystemen (z.B. DRDOS).

Obwohl der mitgelieferte Editor FED.COM direkt auf den Bildschirmspeicher greift, dürfte es nur wenige Probleme mit der verwendeten Hardware geben. Es wird entweder der Bildschirm-Modus 3 (80 Zeichen, mindestens 25 Zeilen) oder der Herkules-Modus 7 unterstützt. Jedoch müssen vom BIOS die entsprechenden Variablen an der Speicheradresse $0040:$0049 (Bildschirmmodus: 3 oder 7) und $0040:$004A (Spaltenanzahl: mindestens 80) gesetzt sein.

1.5 Das Arbeiten mit dem KKF_PC

Bei der Erstellung oder beim Test neuer Programme solle immer das KKF.COM verwendet werden. Da es den Editor schon enthält, entfällt das lästige Nachladen. Man kann dann so vorgehen, wie es im KKF-Handbuch beschrieben wurde.

Sollten Autostart-Programme erzeugt werden, so ist das kürzere KKF_PC.COM vorzuziehen. Bei Bedarf kann man sich dann währende des Laden des Programmes noch den 8086-Assembler auf den Heap laden. Das mit SAVESYSTEM gespeicherte Programm sollte immer mit der Extension .COM versehen werden. Es ist, wie auch das KKF_PC.COM, ein direkt aus der COMMAND-Oberfläche aus startbares Programm.

Kapitel 2

Beschreibung

2.1 Allgemeines zum KKF_PC

Das KKF_PC ist ein vollständig im RAM laufendes COM-Programm. Dies bedeutet, daß sowohl Programm, die Daten als auch beide Stacks im gleichen Segment untergebracht sind. Es wird immer ein Speicherbereich von 64KByte reserviert, wobei das Programm selbst erst bei Adresse $0100 beginnt.

Beim Aufruf von COM-Programmen werden zusätzliche Eingaben vom Betriebssystem erkannt und dem Programm an Speicheradresse $0080 mitgegeben. Das KK-FORTH erkennt dies und verwendet eine vorhandenen Text als erste Eingabezeile. Dadurch lassen sich KKF-Programme aus BATCH-Prozeduren mit Parameter versorgen.

2.2 Speicheraufteilung

Die folgenden Adressen beziehen sich auf das KKF_PC V1.2/0. Programme sollten aber diese Angaben nie direkt verwenden, sondern immer aus Variablen oder Konstanten ermitteln.

$0000		Übliche Daten eines COM-Programmes
$0080		DOS-Zeilenübergabepuffer (mit Längenbyte)
$0100	SYSCON	Anfang des SYSCON-Bereiches
$0110	SYSVAR	Anfang des SYSVAR-Bereiches
$0160		Dictionary-Anfang
$418E	HERE	Anfang des freien Programmbereiches
$F228	VDP @	Anfang des Variablenbereiches
$F26C	HDP @	Anfang des Heap (beim Start leer)
$F26C	TDP @	Anfang des Taskbereiches
$F5C6	TASK0	Anfang des USER-Bereiches im TASK0
$F91E	FIRST	Diskpuffer
$FD20	SYSVAR $2C + @	Anfang des Puffers für 8 Befehlszeilen
$FFB0	SYSVAR $30 + @	Anfang des Anwender-Arbeitsbereich (leer)
$FFB0	(SYSVAR	Kopie des SYSVAR-Bereiches (für SAVE)
$0000	LIMIT	Programmende+1

2.2.1 Systemkonstanten

Die System-Konstanten enthalten Werte, die nur beim Systemstart berücksichtigt werden. Dabei ist beim KKF_PC nur die Angabe des RAM-Ende veränderbar.

SYSCON	**Sprung auf BOOT-Routine**
SYSCON + 4	**reserviert**
SYSCON + 8	**Speicheradresse des SYSVAR-Bereiches**
SYSCON + 10	**Anfangsadresse des Zusatzimage (nicht verwendet)**
SYSCON + 12	**Anfangsadresse des RAM's (nicht verwendet)**
SYSCON + 14	**Endadresse + 1 des RAM's (veränderbar)**

Da bei RAM-Versionen des KK-FORTH die zusätzlichen Befehle direkt an das Dictionary angehängt werden, ist die Angaben für das Zusatzimage nicht nötigt.

Die Anfangsadresse des Programmes ist durch seine COM-Struktur fest mit $0100 vorgegeben. Der im SYSCON-Bereich gespeicherte Wert wird deshalb ignoriert. Er dient wie die Angabe der SYSVAR-Adresse nur zur Information.

Die Endadresse+1 des RAM's ist bei der ausgelieferten Version mit $0000 (entspricht vollem Speicher) angegeben. Sie kann jedoch verändert werden, um dann den restlichen Speicher für eigene Zwecke zu verwenden. Dabei ist aber darauf zu achten, daß noch genügend Speicher für das KK-FORTH bleibt. Um mit der geänderten Speicheraufteilung zu Arbeiten, muß man das Programm erst mit SAVESYSTEM Filename speichern und dann neu starten.

2.2.2 Systemvariablen

Beim Start des KKF_PC werden die Sytemvariablen (ab Adresse $0110) zuerst nach **(SYSVAR** kopiert und erst dann verwendet.

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

SYSVAR + 4 **Prozessorkennung**
$0111 System läuft auch auf einem 8088

SYSVAR + 6 **Hardwarekennung/Betriebssystem**
$0211 System verwendet MSDOS V2.11

SYSVAR + 8 **Versionsnummer**
$1200 Version 1.2/0

SYSVAR + 10 **Attribut-Wort**

```
%0001001100000010
 ^^                    RAM-Version
   ^^                  Segmentierter Speicherverwaltung
     ^^                Ohne Floatingpoint-Unterstützung
       ^               Ein-/Ausgabe über Betriebssystem
        ^              Fileinterface über Betriebssystem
         ^^            Returnstack im Programmsegment
           ^^          Datenstack im Programmsegment
             ^         Kein Align beim 32Bit-Zugriff notwendig
              ^        Kein Align beim 16Bit-Zugriff notwendig
               ^^      Reihenfolge der Daten: d2 d3 d0 d1
```

SYSVAR + 12 **Zeiger auf Kopie der Systemvariablen**
$FFB0 Kopie der Systemvariablen liegen am Speicherende

SYSVAR + 14 **VOC-LINK**
$2156 Zeigt auf das als einziges vorhandene FORTH-Vokabular

SYSVAR + 16 **DP**
$4135 Der KKF_PC-Kern hat eine Länge von 16356 Bytes

SYSVAR + 18 **VDP**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 20 **VLEN**
$0042 Im KKF_PC-Kern werden 82 Bytes für Variablen verwendet

SYSVAR + 22 **HDP**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 24 **HLEN**
$0000 Der Heap ist beim Start leer

SYSVAR + 26 **TDP**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 28 **TASK-LINK**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 30 **TASK**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 32 **Enthält Adresse von TASK0**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 34 **TASKS**
$0001 Es ist nur ein Task definiert

SYSVAR + 36 **TLEN**
$0054 Es sind 84 Bytes des USER-Bereiches verwendet

SYSVAR + 38 **TMAXLEN**
$0100 Jeder Task-USER-Bereich hat eine Länge von 256 Bytes

SYSVAR + 40 **Enthält Adresse für FIRST**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 42 **Enthält Größe des Diskpuffer**
$0402 Platz für Blocknummer und einen Screen

SYSVAR + 44 **SWORK (Adresse des System-Arbeitsspeichers)**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 46 **SLWEN (Länge des System-Arbeitsspeichers)**
$0290 Platz für 8 Zeilen mit je 82 Bytes

SYSVAR + 48 **UWORK (Adresse des Anwender-Arbeitsspeichers)**
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 50 **UWLEN (Länge des Anwender-Arbeitsspeichers)**
$0000 Wird vom KKF-Kern nicht verwendet

SYSVAR + 52 **Enthält Länge einer gespeicherten Eingabezeile**
$0052 Eine Zeile hat 80 Zeichen und zwei Werte (Position und Länge)

SYSVAR + 54 **Enthält die Anzahl der zu speichernden Zeilen**
$0008 Es werden 8 Eingabezeilen verwaltet

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**
$0003 Für 38400 Baud vorbereitet (wird von KKF_V20S verwendet)

SYSVAR + 64 **Reserviert für SIO-Portadresse**
$03F8 Portadresse der SIO1 (wird von KKF_V20S verwendet)

SYSVAR + 66 **Reserviert für versionsabhängige Daten**
$0000 Steht beim KKF_PC ebenfalls zur freien Verfügung

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 wird beim KKF_PC nicht genutzt

SYSVAR + 78 **UFLAG**
Frei für Anwender, wird dem Betriebssystem zurückgeliefert

Die 4 nicht verwendeten Wörter unterhalb von **SFLAG** können für eigene Zwecke verwendet werden. Sie haben den Vorteil, daß ihre Speicheradressen (auch in anderen KKF-Versionen) unveränderlich sind.

2.2.3 Der Taskbereich

Im KKF_PC ist nur ein Task eingerichtet. Dieser Task enthält neben den beiden, jeweils für 256 Einträge vorbereiteten Stacks noch Speicher für **WORD** und der Zahlenstringausgabe (84 Bytes ab **WDP@**), dem Stringbereich (256 Bytes ab **PAD**) und dem Eingabepuffer (zuerst 2 reservierte Bytes, dann 80 Bytes ab **TIB**). Um auch noch Stackunterläufe abzufangen, sind überhalb beider Stacks noch ein kleiner Sicherheitsbereich mit je 6 Bytes eingerichtet.

Da aber alle Speicheradressen beim Kaltstart aus dem im USER-Bereich gespeicherten Angaben ermittelt werden, sind diese Werte im weiten Bereich veränderbar. Jedoch werden keine Tests durchgeführt und falsche Werte bei TASKADDR@+&16 bis TASKADDR@+&32 führen zu undefinierten Reaktionen. Die oben genannten Werte des KKF_PC führen zu folgender Speicheraufteilung:

$F26C	WDP@	Anfang des Arbeitsbereiches für diesen Task - Ablagebereich für WORD
	…	
		- Arbeitsbereich für <# bis #>
$F2C0	PAD	Anfang des Stringbereich - 256 Bytes für Strings
	…	
		- Platz für 256 Stackeinträge
$F5C0	S0 @	Ende des Datenstacks
	…	(danach 6 Bytes frei)
$F5C6	TASKADDR@	Anfangsadresse des USER-Bereiches - Programm/Daten für Taskwechsel
	…	
$F5C6	TASKADDR@ + $0100	Eingabepuffers für **QUERY** (= TIB - 2)
	…	
		- Platz für 256 Returnstackeinträge
$F918	R0 @	Ende des Returnstacks (danach 6 Bytes frei)

2.2.4 Diskpuffer, System- und Anwender-Arbeitsspeicher

Am Ende des verwendeten Speichers befinden sich der Diskpuffer und der Arbeitsspeicher für das System und für den Anwender.

Diskpuffer

Im KK-FORTH wird nur ein Diskpuffer mit einer Länge von insgesamt 1026 Bytes verwendet. Die ersten zwei Bytes nach **FIRST** enthält dabei die Nummer des nachfolgenden Blockes mit 1024 Zeichen. Um bei Veränderung dieses Screens eine nötige Speicherung zu markieren, wird das höchste Bit dieser Nummer als UPDATE-Flag verwendet.

Eine Sonderbedeutung hat dabei die Screennummer 32767 (entspricht $7FFF). Sie kennzeichnet, daß dieser Screen leer ist. Solange kein File geöffnet ist, darf diese Screennummer für die Anforderung eines Arbeitsspeichers verwendet werden. Dabei muß man darauf achten, daß dieser Speicher nicht auf Diskette zurückgeschrieben werden kann und bei Filezugriffe überschrieben wird.

Sytem-Arbeitsbereich

Wenn man das unveränderte KKF_PC verwendet, so werden die letzten 8 Befehlszeilen gespeichert. Der Speicher dazu liegt noch oberhalb des Diskpuffers und ist auf die im SYSVAR-Bereich angegebene Länge von 8*82=656 Bytes gesetzt. Bei der Eingabe wird dann dieser Puffer um 82 Bytes nach oben geschoben und dann die neue Eingabezeile zusammen mit der Längenangabe und der Position des Cursors ab der SWORK-Adresse abgelegt.

USER-Arbeitsbereich

Der vom KKF-Kern unterstützte aber bis jetzt noch nicht verwendete USER-Arbeitsbereich hat eine Länge von 0 Bytes. Dieser Wert läßt sich nur durch die direkte Manipulation der Speicheradresse SYSVAR+&50 ändern. Da aber diese Änderung nur beim Kaltstart berücksichtigt wird, muß man dann das Programm erst mit `SAVESYSTEM Filename` speichern und erneut starten.

Normalerweise wird dieser Speicher durch den KKF-Kern nicht verändert. Da aber beim Programmstart der Stackzeiger an das Ende des Arbeitsspeichers zeigt, werden bei Interrupt einige Einträge benötigt und dabei verändert.

2.3 Aufbau der Befehle

Die folgenden Angaben beschreiben den genauen Aufbau der unterschiedlichen FORTH-Worte. Jedoch sollte man sich bei der Berechnung der entsprechenden Adressen immer auf die Befehle **L>NAME** , **N>LINK** , **>LINK** , **>NAME** , **>BODY** , **LINK>** , **NAME>** und **BODY>** berufen.

Ein Befehl besteht aus:

LFA	Linkfeldadresse	Verkettung zur LFA des nächsten Befehls
NFA	Namensfeldadresse	Filename mit Längenangabe und 3 Flagbits
CFA	Codefeldadresse	Zeiger oder Adresse der Assemblerroutine
PFA	Parameterfeldadresse	Enthält Daten des Befehls

Die höchsten drei Bits in der Namensfeldadresse werden für die Eigenschaft des Befehls verwendet. Die unteren 5 Bits geben dann die Länge des folgenden Befehlsnamen an. Es können deshalb bis zu 31 Zeichen verwendet werden.

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

Bei den angegebenen Befehle (NEST bis (DIC handelt es sich um relativ kurze Assemblerroutinen im KKF-Kern. Sie sind für die Tätigkeit des Befehls verantwortlich.

:-Definitionen am Beispiel :

```
1F7C:28F8  DA 28 01 3A 7F 01 10 05
```

$28F8/9	$28DA	Verkettung zum vorherigen Befehl
$28FA	$01	Namensfeldadresse (Länge: 1 Zeichen)
$28FB	$3A	Befehlsname :
$28FC/D	$017F	Adresse von (NEST
$28FE/F	$0510	Highlevel-Teil (beginnt mit CURRENT)

Konstanten am Beispiel #B/BLK

```
0956:034D  43 03 06 23 42 2F 42 4C 4B C0 01 00 04
```

$034D/E	$0343	Verkettung zum vorherigen Befehl
$034F	$06	Namensfeldadresse (Länge: 6 Zeichen)
$0350 ...	$23 ...	Befehlsname #B/BLK
$0356/7	$01C0	Adresse von (CON
$0358/9	$0400	Wert der Konstanten

32Bit-Konstanten am Beispiel PI

```
3.1415 2Constant pi

0956:418E  C1 3D 02 50 49 B2 01 15 14 03 00
```

$418E/F	$3DC1	Verkettung zum vorherigen Befehl
$4190	$02	Namensfeldadresse (Länge: 2 Zeichen)
$4191/2	$50 $49	Befehlsname PI
$4193/4	$01B2	Adresse von (2CON
$4195..8	$0003.1415	Wert der 32Bit-Konstanten

Variablen am Beispiel FILE-ID

```
0956:05EC  E0 05 07 46 49 4C 45 2D 49 44 A4 01 00 00
```

$05EC/D	$05E0	Verkettung zum vorherigen Befehl
$05EE	$07	Namensfeldadresse (Länge: 7 Zeichen)
$05EF ...	$46 ...	Befehlsname FILE-ID
$05F6/7	$01A4	Adresse von (VAR
$05F8/9	$0000	Offset in den Variablenbereich

32Bit-Variablen am Beispiel COUNTER

```
2Variable counter

0956:4199  8E 41 07 43 4F 55 4E 54 45 52 A4 01 44 00
```

$4199/A	$418E	Verkettung zum vorherigen Befehl
$419B	$07	Namensfeldadresse (Länge: 7 Zeichen)
$419C ...	$43 ...	Befehlsname COUNTER
$41A3/4	$01A4	Adresse von (2VAR (identisch mit (VAR)
$41A5/6	$0044	Offset in den Variablenbereich

USER-Variablen am Beispiel >IN

```
0956:0536  2C 05 03 3E 49 4E 0F 02 3A 00
```

$0536/7	$052C	Verkettung zum vorherigen Befehl
$0538	$03	Namensfeldadresse (Länge: 3 Zeichen)

$0539 ...	$3E ...	Befehlsname >IN
$053C/D	$020F	Adresse von (USER
$053E/F	$003A	Offset in den USER-Bereich

Vokabulare am Beispiel FORTH

```
0956:2068  4B 20 05 46 4F 52 54 48 1D 02 00 00 1E 00
```

$2068/9	$204B	Verkettung zum vorherigen Befehl
$206A	$05	Namensfeldadresse (Länge: 5 Zeichen)
$206B ...	$46 ...	Befehlsname FORTH
$2070/1	$021D	Adresse von (VOC
$2072/3	$0000	Verkettung zum nächsten Vokabular (fehlt)
$0274/5	$001E	Offset in den Variablenbereich
(VDP @ $1E + ergibt $F246)		
$F246/7	$3DC1	Zeiger zur BOOT-Linkfeldadresse

DEFER-Definitionen am Beispiel ERRORTEXT@

```
0956:39B8  A7 39 0A 45 52 52 4F 52 54 45 58 54 40 CA 01 32 00
```

$39B8/9	$39A7	Verkettung zum vorherigen Befehl
$39BA/B	$0A	Namensfeldadresse (Länge: 10 Zeichen)
$39BC ...	$45 ...	Befehlsname ERRORTEXT@
$39C5/6	$01CA	Adresse von (DEFER
$39C7/8	$0032	Offset in den Variablenbereich
(VDP @ $32 + ergibt $F25A)		
$F25A/B	$39C9	Zeigt auf die versteckte Routine

Die versteckte Routine zu **ERRORTEXT@** beginnt unmittelbar hinter dem Befehlsheader. Sie hat aber trotzdem eine eigene Codefeldadresse.

LABEL-Definitionen am Beispiel HEREADDR

```
label hereaddr

1F7C:F25F  BB 8C A8 48 45 52 45 41 44 44 52 5B F2
1F7C:F25B  28 02 F0 8C
```

$F25F/60	$8CBB	Verkettung zum vorherigen Befehl
$F261	$A8	Namensfeldadresse (Länge: 8 Zeichen) (Befehl ist Immediate und Indirect)
$F262 ...	$48	Befehlsname HEREADDR
$F26A/B	$F25B	Zeiger auf Codefeldadresse (im Heap)
$F25B/C	$0228	Adresse von (LABEL
$F25D/E	$8CF0	aktueller Wert des Dictionarypointers

Die im Prinzip wie eine Konstante wirkende Routine zur LABEL-Definition ist unterhalb seines Headers untergebracht. Dadurch wird sie automatisch beim Löschen des Namens entfernt.

ALIAS-Definitionen am Beispiel ALIGNED

```
0956:13E0  CD 13 A7 41 4C 49 47 4E 45 44 67 01
```

$13E0/1	$13CD	Verkettung zum vorherigen Befehl
$13E2	$A7	Namensfeldadresse (Länge: 7 Zeichen) (Befehl ist Immediate und Indirect)
$13E3 ...	$41	Befehlsname ALIGNED

$13EA/B	$0167	Codefeldadresse von NOOP

CREATE-DOES>-Konstruktion am Beispiel 3D-Constant

```
: 3D-Constant   ( x y z -- ; -- x y z )
   Create  rot ,  swap ,  ,
   Does>   dup @  swap cell+  dup @  swap cell+  @ ;
1 2 3 3D-Constant 123

1F7C:8CE4  5F F2 0B 33 44 2D 43 4F 4E 53 54 41 4E 54 7F 01
1F7C:8CF4  B0 27 D0 08 6B 13 BE 08 6B 13 6B 13 37 29 E8 E5
1F7C:8D04  74 38 09 58 0F BE 08 98 0B 38 09 58 0F BE 08 98 ...
```

$8CE4/5	$F25F	Verkettung zum vorherigen Befehl (im Heap)
$8CE6	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8CE7 ...	$33	Befehlsname 3D-CONSTANT
$8CF2/3	$017F	Adresse von (NEST
$8CF4 ...	$27B0	Create-Teil (endet mit (;code)
$8D02...5	$E8 $E5 $74	(does> call,
$8D06 ...	$0938	Does>-Teil (endet mit exit)

```
1F7C:8D19  E4 8C 03 31 32 33 02 8D 01 00 02 00 03 00
```

$8D19/A	$8CE4	Verkettung zum vorherigen Befehl
$8D1B	$03	Namensfeldadresse (Länge: 3 Zeichen)
$8D1C ...	$31	Befehlsname 123
$8D1F/20	$8D02	Zeiger auf Does>-Teil von 3D-CONSTANT
$8D21/2	$0001	x-Wert
$8D23/4	$0002	y-Wert
$9D25/6	$0003	z-Wert

VCREATE-VDOES>-Konstruktion am Beispiel 3D-Variable

```
: 3D-Variable   ( -- ; n -- addr )
   VCreate 0 v,  0 v,  0 v,
   VDoes>  swap cells + ;
3D-Variable vector1

1F7C:8D27  19 8D 0B 33 44 2D 56 41 52 49 41 42 4C 45 7F 01
1F7C:8D37  84 28 65 02 00 00 A9 13 65 02 00 00 A9 13 65 02
1F7C:8D47  00 00 A9 13 37 29 E8 AB 74 BE 08 72 0A 2C 0B 8D
```

$8D27/8	$8D19	Verkettung zum vorherigen Befehl (im Heap)
$8D29	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8D2A ...	$33	Befehlsname 3D-VARIABLE
$8D35/6	$017F	Adresse von (NEST
$8D37 ...	$2884	Create-Teil (endet mit (;code)
$8D4D...F	$E8 $AB $74	(vdoes> call,
$8D50 ...	$08BE	VDoes>-Teil (endet mit exit)

```
1F7C:8D58  27 8D 07 56 45 43 54 4F 52 31 4D 8D CC 00
```

$8D58/9	$8D27	Verkettung zum vorherigen Befehl
$8D5A	$07	Namensfeldadresse (Länge: 7 Zeichen)
$8D5B ...	$56	Befehlsname VECTOR1
$8D62/3	$8D4D	Zeiger auf VDoes>-Teil von 3D-VARIABLE
$8D64/5	$00CC	Offset in den Variablenbereich

CODE-Definition am Beispiel DROP

```
0956:09D3  B8 09 04 44 52 4F 50 DC 09 5B AD 97 FF 25 D1 09
```

$09D3/4	$09B8	Verkettung zum vorherigen Befehl
$09D5	$04	Namensfeldadresse (Länge: 4 Zeichen)
$09D6 ...	$44 ...	Befehlsname DROP
$09DA/B	$09DC	Zeigt auf nachfolgenden Code
$09DC	$5B	tos push,
$09DD	$AD $97 $FF $25	next,
$09E1/2	$09D1	Verkettung zum nächsten NEXT,

PROC-Definition am Beispiel DOSCALL

```
Proc doscall   ( -- )  ok
    $21 int,  ret,  ok
  End-Proc  ok

1F7C:8D66  58 8D 07 44 4F 53 43 41 4C 4C 9A 01 CD 21 C3
```

$8B66/7	$8D58	Verkettung zum vorherigen Befehl
$8B68	$07	Namensfeldadresse (Länge: 7 Zeichen)
$8B69 ...	$44 ...	Befehlsname DOSCALL
$8B70/1	$019A	Adresse von (DIC
$8B72/3	$CD $21	$21 int,
$8B74	$C3	ret,

2.4 Informationen zu KKF_PC-Befehle

Dieses Kapitel enthält Informationen zu Befehlsgruppen wie externer Speicher oder Portzugriffe. Da dabei die Hardware oder das Betriebssystem des verwendeten Rechners genutzt werden muß, unterscheiden sich diese Befehlsaufrufe bei den einzelnen KKF-Versionen geringfügig.

2.4.1 32Bit-Adressen

Alle Prozessoren der 80x86/88er-Familie besitzen (im Real-Modus) eine segmentierte Verwaltung. Solange nur mit den 64KBytes des FORTH-Segmentes gearbeitet wird, ist dies nicht ersichtlich. Nur bei dem Befehl **DUMP** wird auch die Segmentnummer angezeigt.

In der Befehlsliste sind viele Befehle zu finden, die als Parameter eine 32Bit-Adresse (genannt ptr) erwarten. Beim KKF_PC ist dabei immer die kombinierte Angabe von Segment und Adresse gemeint. Dabei wird zuerst die Segmentnummer und dann die Adresse angegeben. Dies hat den Vorteil, daß die Adresse schnell ändern werden kann.

(ptr --) entspricht (seg addr --)

Befehle wie **L2@** oder **L2!** können zur direkten Adressierung eines bestimmten Speichers (z.B. Interrupt-Tabelle) verwendet werden. Dabei ist aber zu beachten, daß wie bei **2@** und **2!** zuerst der oberste Stackeintrag (also die Adresse) und dann erst das Segment abgelegt wird.

Man kann die gleichen Befehle aber auch zu einer echten 32Bit-Adressierung heranziehen. Dabei müssen aber immer die Befehle **D>PTR** und **PTR>D** zur Umrechnung der 32Bit-Adresse in die Pointerangabe und zurück herangezogen werden. Beim KKF_PC werden dabei folgende Umrechnungen durchgeführt:

D>PTR	seg = d / $10 ; addr = d mod $10
PTR>D	d = seg * $10 + addr

Bei allen Befehlen mit ptr-Angabe ist darauf zu achten, daß das Segment nicht geändert wird. Nach der Adresse seg:$FFFF wird die entsprechende Aktion (kopieren, lesen ...) bei seg:$0000 fortgesetzt. Die Umrechnungsbefehle sorgen aber dafür, daß mindestens $FFF0 Bytes verschoben werden können.

2.4.2 Adreß-Align

Obwohl dadurch eine höhere Geschwindigkeit bei der Programmausführung erreicht werden könnte, wird beim KKF_PC auf eine Anpassung an gerade Adressen verzichtet.

Alle Befehle dieser Wortgruppe (**ALIGN** , **ALIGNED** ...) sind deshalb als ALIAS-Definition von **NOOP** angelegt. Da sie zusätzlich als Immediate-Befehle gekennzeichnet sind, fehlen sie in :-Definitionen.

2.4.3 Länge der Datentypen

Die 80x86/88-Prozessoren besitzen einen byteadressierten Speicher. Die kleinste zugreifbare Zelle besteht also aus 8 Bits oder einem Byte.

Jedes Zeichen belegt bei Speicherzugriff mit **C@** und **C!** genau eine Adresse. Gemäß dem IBM-Zeichensatz werden dabei alle Bits genutzt.

Bei Strings wird wie in allen anderen KKF-Versionen zuerst ein Längenbyte, dann die Zeichenkette und zum Schluß noch eine Null abgelegt. Nach Erhöhung der Adresse des Längenbytes um 1 kann der String sofort für alle Filebefehle herangezogen werden.

16Bit-Worte haben eine Länge von zwei Adressen. Dabei dürfen auch ungerade Nummern als Anfangsadresse angegeben werden. Es wird immer das niederwertige Byte zuerst abgelegt.

32Bit-Worte belegen insgesamt 4 Adressen. Da zuerst das höherwertige und dann erst das niederwertige Wort ablegt oder geholt wird, ergibt sich folgende Speicherbelegung:

addr	Bit 16-23
addr+1	Bit 24-31 (höchstwertigste Byte)
addr+2	Bit 0-7 (niederwertigste Byte)
addr+3	Bit 8-15

2.4.4 Portbefehle

Es können sowohl 8- als auch 16Bit-Portzugriffe durchgeführt werden. Deshalb werden die Befehle **P@** und **P!** für 16Bit- und **PC@** bzw. **PC@** für 8Bit-Zugriffe verwendet. Wie bei normalen Speicherzugriffe wird auch hier bei 16Bit-Werte zuerst das niederwertige Byte an der niedrigeren Adresse abgelegt oder von dort geholt.

2.4.5 Filebefehle

Alle Filebefehle des KKF_PC sind direkte Aufrufe der entsprechenden (MS)DOS-Befehle. Als Wert wird dabei eine Handlenummer geliefert oder erwartet. Files auf Diskette oder Harddisk beginnen dabei erst ab Handlenummer 5. Die maximale Anzahl der gleichzeitig geöffneten Files ist von der Angabe in CONFIG.SYS (z.B. FILES=20) abhängig. Für die Handlenummern 0 ... 4 sind schon bestimmte Geräte vorgesehen, die weder geöffnet noch geschlossen werden müssen:

0	Standard-Eingabegerät (CON; umleitbar)
1	Standard-Ausgabegerät (CON; umleitbar)
2	Standardgerät zur Ausgabe von Fehlermeldungen
3	Serielle Schnittstelle (AUX)
4	Standarddrucker (PRN)

Es können bis zu 65535 Bytes vom Speicher auf das File übertragen oder vom File in den Speicher geladen werden. Dabei ist zu beachten, daß die Segmentnummer nicht verändert wird.

Beim Setzen des Filepointers mit **(FILE-POS!** wird neben dem (auch negativen) Offset noch die Angabe der entsprechenden Richtung dir erwartet. Dabei sind folgende Werte zulässig:

dir=0	Offset ab Anfang des Files
dir=1	Offset ab aktueller Position
dir=2	Offset ab Fileende

2.4.6 Fehlermeldungen

Die Texte zu den Fehlermeldungen sind am Ende des KKF_PC-Kerns untergebracht. Man kann deshalb, wie im Handbuch beschrieben, durch Veränderungen von **>ERRORTEXT** und Freigabe des belegten Speichers mit **ALLOT** Platz für eigene Fehlermeldungen schaffen.

2.5 Zusatzhinweise

Ein Blick hinter die "Kulissen" des KKF-Systems soll in dem letzten Kapitel der Beschreibung des KKF_PC-Kerns die Initialisierungssequenz und die Ein-/Ausgabe verdeutlichen. Die Realisierung der Schnittstellenbefehle im KKF_PC ist als Listing (Files IO_MSDOS.SCR) im Anhang E angegeben.

2.5.1 Initialisierung des Systems

Beim Start des KKF_PC werden durch die BOOT-Routine zwei Interruptvektoren und ein Flag des Betriebssystems gespeichert und auf eigene Routinen umgeleitet. Erst beim Verlassen des Programms werden die beiden Vektoren und das Flag wieder auf den alten Wert gesetzt.

Speicherallokierung

Beim Start eines Programmes wird vom Betriebssystem immer der gesamte Speicher übergeben. Um aber auch andere Programme über dem KK-FORTH zu starten oder Speicher für eigene Daten nutzen zu können, werden in der BOOT-Routine nur noch 64KByte für das KK-FORTH zurückgehalten und der Rest wieder freigegeben. Er kann dann mit **(MALLOC** angefordert, mit **(MRELOC** in der Größe verändert und mit **(MFREE** wieder freigegeben werden. Dabei kann ein angeforderter Speicher nur dan vergrößert werden, wenn keine weiteren Datenfelder belegt worden sind. Der Grund dazu ist, daß jeder Speicher einen Header mit 16 Bytes bekommt, in dem Informationen zur Adresse weiterer Speichersegmente zu finden sind.

DIV0-Interrupt

Bei allen Divisionsbefehlen kann ein Werteüberlauf auftreten, wenn durch einen zu kleinen Wert oder durch 0 geteilt wird. Der Prozessor erkennt dies und leitet sofort eine Fehlerbehandlung ein. Der normalerweise gesetzte Interrupt-Vektor würde dabei das KK-FORTH mit einer Fehlermeldung abbrechen. Da dies nicht erwünscht ist, wird dieser Vektor auf eine eigene Routine umgeleitet, die nach einer entsprechenden Fehlermeldung wieder im FORTH-Interpreter/Kompiler landet.

CTRL+C-Interrupt

Die meisten Ein-/Ausgabebefehle des Betriebssystems haben die unangenehme Eigenheit, daß mit der Tastenkombination CTRL+C das Programm verlassen wird. Nur durch Verwendung bestimmter Ein-/Ausgabebefehle (siehe nächstes Kapitel) und der Veränderung eines Interrupt-Vektors (INT $23) und des CTRL+C-Flags kann dies verhindert werden. Eine Sicherung des

CTRL+C-Interrupt ist dabei nicht notwendig, da er bei Programmstart automatisch gespeichert wird.

2.5.2 Ein-/Ausgabe und das Fileinterface

Um dem Anwender das Schreiben eigener Ein-/Ausgaberoutinen zu erleichtern, ist das Listing der entsprechenden Routinen dem Handbuch angehängt. Da es für den Targetkompiler konzipiert ist, besitzt es Vorwärtsreferenzen und kann deshalb nicht direkt geladen werden.

Zeichen von Tastatur holen

Da es nicht möglich ist, über Handlenummern den Status der Tastatur abzufragen, muß die Tastatur über die INT$21-Funktion 6 gepollt werden. Da aber dieser Befehl nicht nur den Status, sondern auch ein vorhandenes Zeichen liefert, besitzt das KKF_PC einen Zeichenpuffer. Sowohl **KEY** als auch **KEY?** fragen deshalb zuerst den Zeichenpuffer ab und rufen erst dann die DOS-Funktion auf, wenn kein Zeichen vorhanden ist.

Das Betriebssystem kann diese Funktion auch auf Files umleiten. Da aber diese Umleitung für das KK-FORTH transparent ist und vom Betriebssystem nicht automatisch am Fileende wieder auf Tastatur zurückgestellt wird, müssen im File auch der Befehl für das Beenden des Programmes untergebracht sein. Ein Beispiel dafür ist das File MAKEKKF.KKF zur Erstellung der KKF-Vollversion.

Zeichen ausgeben

Da bei allen anderen Ausgabe-Funktionen beim Drücken der CTRL+C-Taste das Programm beendet oder das Zeichen "^C" ausgegeben wird, mußte die Zeichenausgabe ebenfalls über die INT$21-Funktion 6 abgewickelt werden. Da aber dabei der Zeichencode $FF (=&255) schon für die Statusabfrage reserviert ist, kann dieses Zeichen nicht ausgegeben werden. Weil der Wert $FF sowieso kein druckbares Zeichen liefert, wird es (wie auch beim volksFORTH_PC) bei **EMIT** und **TYPE** durch $FE ersetzt.

Beim Aufruf des KKF_PC kann durch die Angabe >Filename oder >Device die Ausgabe auf ein File oder ein anderes Gerät umgeleitet werden. Da dies für das KKF_PC transparent ist, können weiterhin nur die Zeichen $00 bis $FE ausgegeben werden.

Filebefehle

Wie schon in Kapitel 2.4.5 angesprochen, sind alle Filebefehle durch direkten Aufruf der entsprechenden Handle-Befehle realisiert. Man muß immer dafür sorgen, daß alle im KK-FORTH geöffneten Filenummern auch wieder geschlossen werden. Ansonsten kann es zu der Fehlermeldung "too many open files" kommen.

Kapitel 3

Zusatzprogramme

In diesem Kapitel werden Tools und Beispielprogramme beschrieben. Da die Files AUTO.SCR , ERRTRAP.SCR und FFT.SCR schon im Handbuch beschrieben sind, fehlen sie hier.

Durch die Eingabe von

KKF_PC <makekkf.kkf

wird die Möglichkeit der Eingabeumleitung genutzt und alle Anweisungen zur Erstellung des Programmes KKF.COM aus dem File entnommen.

3.1 ASM8086.SCR

Bei dem 8086-Assembler handelt es sich um einen modifizierten volksFORTH-Assembler (Autor: Klaus Schleisiek, Hamburg), der dabei als Ausgangsbasis den F83-Assembler verwendet hat.

Um bei einem einheitlichen Stiel in allen KKF-Assemblern zu bleiben, wurde an alle Befehle das Komma angehängt und die von Klaus Schleisiek verwendeten Klammern durch entsprechende Kontrollstruktur-Befehle ersetzt.

Aufbau der Assemblerbefehle

Bei den Assemblerbefehlen kommt immer der eigentliche Opcode - mit angehängtem Komma - immer am Ende der Definition. Davor stehen jeweils die benötigten Register mit nachfolgenden Adressierungsart. Werden bei einem Befehl mehrere Angaben benötigt, so kommt zuerst die Quelle (oder der Wert) und dann das Ziel.

```
sti,
tos inc,
tos ) inc,
$1234 # tos ) mov,
tos ) ax mov,
```

80x86-Register

Im Assembler sind alle Register des Prozessors verfügbar. Einige davon werden aber schon für das KK-FORTH verwendet und müssen deshalb gerettet werden.

CS = DS = SS	Alle Segmente verwenden die gleiche Bank
SP	FORTH-Datenstackzeiger
BP	FORTH-Returnstackzeiger
SI	FORTH-Befehlszeiger
DI	Arbeitsregister enthälte aktuelle Codefeldadresse
BX	Enthält den obersten Datenstackwert

AX, CX, DX, DI und ES sind frei verwendbar

Für den Assembler sind alle Register definiert:

```
AX   CX   DX   BX ( = TOS )
SP ( = FSP )   BP ( = FRP )   SI ( = FIP )   DI ( = W )
AH   AL   BH ( = TOSH )   BL ( = TOSL )   CH   CL   DH   DL
ES   CS   SS   DS
```

Adressierungsarten

Der Assembler legt während der Definition die Angabe der Adressierungsart auf den Datenstack. Da aber sowohl Register als auch bestimmte Adressierungswerte damit angegeben werden, muß bei Zahlenwerte oder Adreßangaben immer ein # oder ein #) folgen.

AH ... DL	8Bit-Register
AX ... DI	16Bit-Register
ES ... DS	Segmentregister
n #	Zahlenwert
n #)	Inhalt einer Adresse
c*	Register CX wird als Zähler verwendet (nur bei Schiebebefehle)
)	Inhalt einer Adresse auf das ein 16Bit-Register zeigt.
I)	Indizierte Adressierung
D)	Adressierung mit Displacement
DI)	Indizierte Adressierung mit Displacement

Zusätzlich sind bei einigen Befehlen noch eine explizite Angabe der Wortlänge notwendig. Falls nichts angegeben ist, wird eine Wort-Adressierung angenommen.

byte	nur ein Byte wird übertragen
word	ein 16Bit-Wert wird übertragen (Low-Byte zuerst)
far	für Sprünge über Segmentgrenzen hinweg

Kontrollstrukturen

Bei der Verwendung von Sprüngen oder Unterprogrammaufrufen ist eine 16Bit-Adresse anzugeben. Sollte einmal innerhalb eines Befehls ein bedingter oder unbedingter Sprung eingebunden werden, so sind folgende Kontrollstrukturen zu verwenden:

```
Bedingung IF,  ( ELSE, )  THEN,
BEGIN,  Bedingung WHILE,  REPEAT,
BEGIN,  Bedingung UNTIL,
BEGIN,  AGAIN,
```

Folgende Bedingungen sind dabei definiert:

```
0=   0<>   0<   0>=
<   >=   <=   >
u<   u>=   u<=   u>
ov   nov
<>c0=   c0=   ?c0=   c0<>
```

Beispiele:

aaa,	Befehl ohne Adreßangabe
$1234 # ax mov,	Lade Register ax mit Wert $1234
vdp #) tos mov,	Lade TOS mit Inhalt der Systemvariable VDP
ax rol,	AX um ein Bit verschieben
ax c* rol,	AX um CX Bits verschieben
tos) cl mov,	Lade CL mit dem Inhalt der Adresse TOS (BX)
tos w i) tos mov,	Lade TOS mit Inhalt der Adresse TOS+W
2 w d) tos mov,	Lade TOS mit Inhalt der Adresse W+2
2 tos w di) tos mov,	Lade TOS mit Inhalt der Adresse TOS+W+2
0= IF, ... THEN,	Führe Befehl aus, wenn Zero-Flag gesetzt ist

3.2 DIS8086.SCR

Der Disassembler ist ebenfalls dem volksFORTH entnommen und wurde von Charles Curley entwickelt. Beim Laden von DIS8086.SCR wird ein Vokabular mit Namen **DISAM** angelegt. Im FORTH-Vokabular stehen danach folgende Befehle zur Verfügung:

dis	(name ; --)	Code eines FORTH-Befehls disassemblieren
disasm	(addr --)	Ab addr disassemblieren
ldisasm	(seg:addr --)	Ab Pointer-Angabe disassemblieren

Der Disassembler generiert immer 10 Zeilen oder geht bis zum nächsten (NEXT-Code und wartet dann auf eine Bestätigung. Mit CTRL+X kann die Ausgabe abgebrochen werden. Im Vokabular **DISAM** ist noch die Variable **LINES** untergebracht. Falls eine bestimmte Anzahl von Zeilen ausgegeben werden soll, so ist der entsprechende Wert in diese Variable einzutragen und dann einer der oben genannten Befehle zu verwenden.

3.3 PC_DEBUG.SCR

Dieses File enthält sowohl einen Dekompiler als auch den Debugger. Obwohl es im Screen 1 schon vorbereitet ist, wurde auf das Anlegen eines eigenen Vokabulars verzichtet.

Der FORTH-Dekompilier

Ein Dekompilier ist die Umkehrung zu einem Kompiler und erzeugt aus dem FORTH-Code wieder ein Befehlslisting. Jedoch kann nur dann der entsprechende Befehlsname angegeben werden, wenn er nicht versteckt war. Es können natürlich nur :-Definitionen dekompiliert werden.

(decom	(min max --)	Speicherbereich dekompilieren
decom	(name ; --)	Befehl dekompilieren

Man kann entweder direkt den gewünschten Bereich angegeben oder der Bereich wird aus dem Befehlsname ermittelt. Da bei der Suche nach dem Befehlsende der nächste sichtbare Name herangezogen wird, kann es auch vorkommen, das **DECOM** mehrere Befehle dekompiliert. Bei den Befehlen wird immer der zuletzt definierte ALIAS-Name angegeben (z.B. **D>PTR** statt **SWAP**).

Bei beiden Befehlen wird die Ausgabe nach 10 Zeilen gestoppt und kann mit CTRL+X abgebrochen werden.

Der FORTH-Debugger

Der Debugger ist für die schrittweise Bearbeitung eines FORTH-Befehls gedacht. Dazu stehen folgende drei Befehle zur Verfügung:

debug	(name ; --)	Befehl <name> debuggen
debug	(min max --)	Bereich von min bis max debuggen
unbug	(--)	Debugger wieder abschalten

Wie der Debugger anzuwenden ist, wurde schon im Handbuch beschrieben. Wichtig ist in diesem Zusammenhang nur, daß auf keinen Fall die Befehle des Debuggers entfernt werden dürfen, bevor man nicht mit **UNBUG** die alten (NEXT-Routinen wieder aktiviert hat.

3.4 PC_EXTRA.SCR

Das Betriebssystem unterstützt entweder direkt über DOS-Aufrufe oder indirekt über den Befehlsinterpreter sowohl Datum-/Uhrzeitabfrage als auch viel Filebefehle wie DIR oder COPY . Einige dieser Befehle stehen nach dem Laden dieses Files auch dem Anwender zur Verfügung.

Datum- und Uhrzeit-Abfrage

Bei den Files wird immer der Zeitpunkt der letzten Änderung im sogenannten DOSDATE-Format gespeichert. In dieser komprimierten 32Bit-Angabe ist das Datum und die Uhrzeit mit einer Genauigkeit von 2 Sekunden gespeichert.

Bit 0-4	Sekunden/2
Bit 5-10	Minuten
Bit 11-15	Stunden
Bit 16-20	Tag (1...31)
Bit 21-24	Monat (1...12)
Bit 25-31	Jahr (ab 1980 gerechnet)

(date@	(-- wd t m j 0 | error) Datum holen	
(time@	(-- 1/100 s m h 0 | error) Uhrzeit holen	
dosdate@	(-- dosdate.)	DOSDATE holen
dosdate.	(dosdate. --)	DOSDATE ausgeben

Die ersten beiden Befehle dienen zur direkten Uhrabfrage und liefern entweder ein Fehlerflag oder unter dem OK-Flag 0 die entspechenden Angaben. **(DATE@** liefert aktuellen Wochentag (wd:0=Sonntag), Tag, Monat und Jahr. **(TIME@** dient zur Bereitstellung des aktuellen Wertes von 1/100stel Sekunde, Sekunde, Minute und Stunde.

Bei den DOSDATE-Befehlen wird immer ein 32Bit-Wert erwartet oder geliefert. Dabei enthält das höherwertige Wort die Tagesangabe und das niederwertige Wort die Uhrzeit. Bei der Ausgabe mit wird das Format "hh:mm:ss dd:mm:jj" erzeugt.

DOS-Befehlsaufruf

Sehr mächtig ist der ebenfalls im File enthaltene Befehl . Er wird entweder zum direkten Aufruf eines anderen Programmes oder über den Kommandointerpreter des Betriebssystems zur Ausführung der DOS-Befehle genutzt. Bei der Definition eines neuen DOS:-Befehls wird der Filename und ein String erwartet. Wenn dann der neue Befehl ausgeführt wird, kann ebenfalls noch ein Befehlsstring übergeben werden. Deshalb sollten diese Befehl nicht in :-Definitionen verwendet werden.

Bei der Definition von Befehlen des Kommandointerpreter muß wie bei dem nachfolgenden Beispiel am Anfang des Strings noch das "/C" stehen. Ebenso wichtig ist noch das Leerzeichen zwischen DIR und dem abschließenden Anführungszeichen.

Beispiel: Directory-Ausgabe

```
DOS: dir /C DIR "          ( Definition des Befehls DIR )
dir *.*                    ( Ausgabe des Directory )
```

Hier noch eine vollständige Aufstellung der DOS:-Befehle des Files:

(dosshell	(csa --)	DOS-Shell mit angegebenen String aufrufen
dosshell	(String ; --)	DOS-Shell aufrufen
dir	(String ; --)	Directory ausgeben
copy	(String ; --)	Files kopieren
cd	(String ; --)	Directory wechseln
md	(String ; --)	Neues Directory einrichten

rd	(String ; --)	Directory entfernen

3.5 NECP6.SCR

Natürlich möche man ein Programm auch in schriftlicher Form vorliegen haben. Dies ist aber bei FORTH-Programmen nicht durch andere Text-Editoren zu erreichen, da keine Steuerzeichen im File vorhanden sind.

Programmlisting

Das hier vorliegende File ist für den NEC-P6 vorbereitet, kann aber ohne oder mit geringfügigen Änderungen auch für andere Drucker verwendet werden. Es dient zum Ausdruck eines Programmes in einem von drei Formaten. Um die Listings mit eigenen Copyright-Meldungen zu versehen, können die Fußzeilen auch auf eigene Routinen umgeleitet werden.

	(Befehle mit Copyright-Meldung: (C) Klaus Kohl ...)	
KKLIST:	(File ; --)	6 Screens pro Seite (0-3; 1-4;2-5)
KKSLIST:	(File ; --)	3 Screens mit Shaddowscreens pro Seite
KKDLIST:	(File ; --)	Zwei A5-Seiten mit 4 Screens pro A4-Seite
	(Befehle ohne Copyright-Meldung)	
	(File ; --)	6 Screens pro Seite (0-3; 1-4;2-5)
SLIST:	(File ; --)	3 Screens mit Shaddowscreens pro Seite
DLIST:	(File ; --)	Zwei A5-Seiten mit 4 Screens pro A4-Seite

Besonders gut für das Handbuch ist das letzte Format geeignet. Auf einem A4-Blatt werden dabei zwei A5-Seiten mit je 4 Screens, Kopf- und Fußzeile gedruckt. Nach dem Auseinanderschneiden der beiden Teile können sie gelocht und in das Handbuch eingelegt werden.

Druckerbefehle

Die Grundlage des Programmlistings sind die nach dem Laden des Files ebenfalls verfügbaren Steuerbefehle für den Drucker. Da das Listing mit ausreichendem Kommentar versehen ist, wird auf die Beschreibung der einzelnen Steuerzeichen verzichtet.

Um auch die Ausgaben von **DUMP** oder Disassembler auf den Drucker umzuleiten, wurde ein eigener Ausgabevektor mit Namen **>PRINTER** angelegt. Dieser im Vokabular **PRINTER** untergebrachte Befehl stellt die Ausgabe auf den Drucker um, wobei die Befehle **EMIT?** , **EMIT** , **TYPE** , **CR** , **CLS** (=Blattvorschub) , **AT** und **AT?** unterstützt werden. Vor Aufruf von **>PRINTER** sollte man sich die den vorherigen Vektor in **OUTPUT** merken und nach der Ausgabe wieder zurückschreiben.

3.6 PC_EDIT.SCR

Der direkt zum KKF_PC ladbare FORTH-Screeneditor verwendet den Diskpuffer als einziger Screenspeicher und den RAM-Bereich zwischen Dictonary und Variablen als Zeilen-/Zeichenpuffer. Auf eine ausführliche Beschreibung der Bedienung wird verzichtet, da sie schon im Handbuch erfolgte. Hier nur noch eine Zusammenfassung der Befehle:

L	(n --)	Screen n editieren
V	(--)	Letzte Fehlerposition editieren
-V	(--)	Zur letzten Editor-Position
FROM	(name ; --)	Angabe des zweiten Files

Da durch den Editor direkt auf den Bildschirmspeicher des PC's gegriffen wird, muß man sich unbedingt im Textmodus mit 80 Zeichen pro Zeile (Mode 3) befinden. Falls der Herkules-Mode genutzt werden soll, so ist das Programm FED.COM zu nutzen oder man muß die entsprechenden Anpassungen aus MEM_EDIT.SCR (zweite Diskette) übernehmen.

3.7 SIEVE.SCR

Ein weiteres, auf allen Rechnern gleichermaßen lauffähiges Beispiel ist die Ermittlung der Primzahlen durch das Sieb des Eratosthenes. Entgegegen allen sonstigen Gewohnheiten wird hier einfach das Dictionary zur Speicherung eines 16387 Byte großen Datenfeldes mißbraucht. Der Befehl **PRIMES** ermittelt dann die Anzahl der Primzahlen in diesem Bereich. Durch Aufruf von **AUSGABE** kann man sich eine Tabelle aller gefundenen Primzahlen ausgeben lassen. Dabei wird auch der Wert 0 als Primzahl angezeigt.

Durch Veränderung der Konstante **SIZE** vor dem Laden des Programmes können auch ein anderer Bereich analysiert werden. Die Größe ist durch den freien Speicher und der größten vorzeichenbehafteten Zahl 32767 begrenzt.

Das Programm markiert zuerst das ganze Feld durch Einschreiben von 1 als lauter Primzahlen. Danach beginnt es bei 2 und setzt immer die Vielfachen einer gefundenen Primzahl auf 0. Die dann noch verbleibenden 1er-Felder stellen dann die Primzahlen dar.

3.8 G_BIOS.SCR

IBM-PC's und Kompatible verfügen über unterschiedliche Grafik-Hardware. Um aber trotzdem die meisten Programme ohne spezielle Anpassung an die Hardware nutzen zu können, verfügt jeder Rechner über die BIOS-Grafik-Routinen für Grafik.

Das File G_BIOS.SCR enthält die wichtigsten Grafikbefehle bis hin zum Setzen und Löschen einzelner Punkte im hochauflösenden Bildschirm. Da diese Funktionen aber sehr langsam sind, wird oft (wie auch im nächsten Programm oder beim Screeneditor) direkt auf den Bildschirmspeicher zugegriffen.

mode!	(mode --)	Grafikmodus setzen (3=Text;7=Herkules ...)
mode@	(-- mode scr)	Grafikmodus und aktueller Screen abfragen
clrscr	(--)	Screen 0 einstellen und löschen
viewpage!	(scr --)	Sichtbare Screennummer ändern
viewpage@	(-- scr)	Sichtbare Screennummer abfragen
activpage!	(scr --)	Ausgabescreen einstellen
activpage@	(-- scr)	Ausgabescreen abfragen
curdim!	(start end --)	Start-/Endezeile des Cursors einstellen
curdim@	(-- start end)	Start-/Endezeile des Cursors abfragen
curat!	(x y scr --)	Cursorposition setzen
curat@	(scr -- x y)	Cursorposition abfragen
curatmax	(-- xmax ymax)	Bildschirmgröße ermitteln
curchar@	(scr -- c attr)	Zeichen und Attribut ermitteln
curchars!	(c attr scr n --)	Zeichen wiederholt ausgeben
curstringat!	(addr len attr x y scr --)	String ausgeben (Zeilenende ignorieren)
dot!	(x y col --)	Punkt setzen
dot@	(x y -- col)	Farbe eines Punktes abfragen

3.9 G_VGA.SCR

Da die BIOS-Aufrufe sehr langsam sind, wird bei vielen Programmen und Programmiersprachen direkt auf den Bildschirmspeicher zugegriffen. Dieses Beispiel für die VGA-Karte EIZO MDB-10 verwendet den 256Farben-Modus. Dabei können bis zu einer Auflösung von 800*600 Bildpunkte einzelne Punkte gesetzt oder sogar Linien und Kreise gezogen werden. Bei geringster Auflösung kann z.B. für animierte Grafik mit mehreren Screens gearbeitet werden. Dazu wird auf dem versteckten Screen ein Bild gezeichnet und dann umgeschaltet.

Grundbefehle

mode!	(mode --)	Grafikmodus setzen
mode@	(-- mode)	Grafikmodus abfragen
viewpage!	(scr --)	Sichtbare Screennummer ändern
viewpage@	(-- scr)	Sichtbare Screennummer abfragen
activpage!	(scr --)	Ausgabescreen einstellen
activpage@	(-- scr)	Ausgabescreen abfragen

Grafikbefehle

320*200	(--)	Grafikmodus &19 (320*200; 4 Screens)
640*350	(--)	Grafikmodus &45 (640*350; 1 Screen)
640*480	(--)	Grafikmodus &46 (640*480; 1 Screen)
800*600	(--)	Grafikmodus &48 (800*600; 1 Screen)
grafik	(--)	Grafikmodus einstellen
text	(--)	Textmodus einstellen
maxx	(-- xmax)	Maximale Bildschrirmbreite
maxy	(-- ymax)	Maximale Bildschirmhöhe
maxcol	(-- cmax)	Maximale Farbanzahl
maxpage	(-- pmax)	Maximale Seitenanzahl
gmode@	(-- mode)	Verwendeter Grafikmodus
fillpage	(col --)	Bildschirm mit einer Farbe füllen
dot!	(x y col --)	Pixel setzen
circle	(x y r col --)	Kreis um Mittelpunkt x,y mit Radius r
line	(x1 y1 x2 y2 col --)	Linie von x1/y1 nach x2/y2 ziehen

Die Befehle **320*200** bis **800*600** stellen einige Variablen um. Die Werte in der verwendeten Tabelle ist für die MDB10-Grafikkarte vorbereitet. Oft muß für eine andere VGA-Karte nur die Modusnummer geändert werden.

3.10 GTEST.SCR

Die Befehle des Files G_VGA.SCR dienen als Grundlage für dieses Beispiel-Menü. Alle vier verfügbaren Grafikmodes können eingestellt und mit unterschiedlichen Muster gefüllt werden. Dabei unterscheidet man zwischen Hyperbel, Regenbogen, Kreise und Sterne.

Das Programm ist durch Drücken der einzelnen Tasten zu steuern. Dabei muß man bei langsameren Rechnern im Menüpunkt Stern etwas Geduld haben, da der Bildschirm zuerst mit der Hintergrundfarbe gefüllt wird. Der Menüpunkt "Animierte Kreise" ist die Tastenkombination CTRL+X zu beenden.

Selbst bei anderen VGA-Grafikkarten ist der Grafikmodus 320*200 verwendbar, da er eine von IBM vordefinierte Modusnummer besitzt. Andere Grafikkarten können durch Veränderung in G_VGA.SCR angepaßt werden.

3.11 PC_MOUSE.SCR

Wie einfach die Mausabfrage im FORTH sein kann, soll mit dem letzten Programm der ersten Diskette gezeigt werden. Mit wenigen Assemblerbefehlen werden die wichtigsten Funktionen zur Abfrage und Steuerung der Maus zur Verfügung gestellt.

mflag	(-- addr)	Variable mit Flag, ob Maus verfügbar
minit	(--)	Initialisieren und MFLAG setzen
mcuron	(--)	Mauscursor einschalten
mcuroff	(--)	Mauscursor ausschalten
mxset	(min max --)	Mausfenster in X-Richtung begrenzt
myset	(min max --)	Mausfenster in Y-Richtung begrenzt
m>coord	(x y -- x y)	Maus- in X/Y-Position umrechnen
coord>m	(x y -- x y)	X/Y- in Maus-Position umrechnen
mpos!	(x y --)	Maus an Position X/Y setzen
mpos@	(-- b x y)	Mausposition und Tasten abfragen

Im Beispielprogramm **GRAFIKMAUS** werden diese Befehle zusammen mit den VGA-Befehlen genutzt, um auf einem Grafikscreen zu zeichnen. Dabei wird immer bei Drücken der linken Maustaste eine Linie zwischen der vorherigen und der aktuellen Mausposition gezogen. Mit der rechten Maustaste wird dann das Programm beendet und der Textmodus wieder aktiviert.

3.12 MEM_EDIT.SCR

Auf der zweiten Diskette ist ein eigenständiges Editorprogramm untergebracht, daß das gesamte File in den externen Speicher des PC's ladet und dort editiert. Zusätzlich erkennt dieses Programm auch noch den Unterschied zwischen EGA- und Herkules-Karte und stellt sowohl die Speicheradresse als auch die Farbe der Ausgaben darauf ein.

Die Sourcen zu FED.COM sind im File MEM_EDIT.SCR untergebracht. Zur Kompilierung des geänderten FED-Programmes ist folgende Eingabe notwendig:

```
KKF_PC  include mem_edit.scr
```

Vom Sourcefile werden dann automatisch die benötigten Zusatzsourcen wie Assembler und die Zusatzbefehle zum DOS (in PC_EXTRA.SCR) nachgeladen. Nach dem vollständigen Laden der Files wird das neue FED.COM gespeichert und das Programm beendet. Die Tastenbelegung des Editors entspricht dabei dem des File PC_EDIT.SCR und wurde schon im Handbuch beschrieben. Der Anhang dieser Zusatzbeschreibung enthält ebenfalls die vorgegebene Belegung. Durch Veränderung der Tabelle können aber auch andere Tastenkombinationen eingestellt werden.

3.13 PC_TERM.SCR

Ebenfalls auf der zweiten Diskette ist das für alle EMUF-Versionen des KK-FORTH genutzte Terminalprogramm mit allen Sourcen untergebracht. Im File PC_TERM.SCR sind alle Anweisungen zur Erstellung des Programmes und die eigentlichen Steuerbefehle des Terminals enthalten. Durch die Anweisung

KKF_PC include pc_term.scr

kann ein verändertes Terminalprogramm erstellt und gespeichert werden. Die Funktionen der einzelnen Tasten ist sowohl im Handbuch als auch im Anhang dieser Beschreibung zu finden.

Vom Terminalprogramm werden auch die zusätzlich vorhandenen Files T_EXTRA.SCR, T_EDIT.SCR und FRAME.SCR geladen. Die ersten beiden Files sind etwas veränderte Versionen von PC_EXTRA.SCR und PC_EDIT.SCR . Das letzte File enthält einige auch für eigene Zwecke gut verwendbare Befehle zur Begrenzung der Bildschirmausgabe auf einen bestimmten Bereich des Bildschirms.

Anhang A

Glossar-Zusatz

Der folgende Glossarzusatz enthält nur Befehle, die nicht im Handbuch aufgeführt worden sind oder zu denen es weitere Hinweise gibt.

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	Systemabhängige 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 (seg:addr) für externen Speicherzugriff

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 versteckter Runtime-Routine
83	Befehl des FORTH83-Standards
E83	Zusatzbefehl zum FORTH83-Standard
ANSI	Befehl des gepanten ANSI-Standards

#BRK (-- $18) Con
Als Abbruchtaste wird CTRL+X verwendet.

#DELIN (-- $08) Con
Bei KKF_PC liefert **#DELIN** den Wert $08 und entspricht damit dem Code der Backspace-Taste oder der Tastenkombination CTRL+H.

#DELOUT (-- $08) Con
Bei der Ausgabe dient der von **#DELOUT** gelieferte Wert $08 zur Verschiebung des Cursors um eine Position nach Links. Es werden aber keine Zeichen gelöscht.

#RO (-- $00) Con
Attribut für **(FILE-OPEN** , daß nur gelesen werden soll.

#RW (-- $02) Con
Attribut für **(FILE-OPEN** , daß sowohl gelesen als auch geschrieben werden soll..

#TIB-MAX (-- 79) Con
Der Wert wird von **QUERY** verwendet und ist beim KKF_PC auf 79 Zeichen gesetzt, um kein Zeilenumbruch bei der Befehlseingabe zu verursachen.

#WO (-- $01) Con
Attribut für **(FILE-OPEN** , daß nur geschrieben werden soll.

(FILE-CLOSE (handle -- error) UV
Ist das File verändert worden, so wird noch das Datum und die Uhrzeit des Fileeintrages auf den aktuellen Wert gesetzt.

(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. Dabei wird eine DOS-Funktion verwendet (INT$21-$48), die unterhalb der zurückgelieferten Adresse einen MCB mit 16 Bytes zur Verwaltung anlegt. Bei der zurückgelieferten ptr-Angabe ist die Adresse immer auf 0 gesetzt.

(MFREE (ptr -- error)
Freigeben eines mit **(MALLOC** reservierten Speichers. Ein Fehler wird ausgegeben, falls ptr nicht auf den Anfang eines mit **(MALLOC** belegten Speichers zeigt.

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

,A (cfa --) DX
Die angegebene Codefeldadresse wird unverändert ins Dictionary übernommen.

,C (w --) DX
Dieser Befehl ist im KKF_PC V1.2/0 nicht verfügbar.

?BRANCH (f --) 83 R
?BRANCH benötigt noch einen Inline-Offset, den dann zum aktuellen Programmzeiger addiert wird, wenn das Flag f Null ist.

AT (x y --) UV
Im KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden.

AT? (-- x y) UV
Im KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden.

BRANCH (--) E83 R
Dieser Befehl erwartet ein nachfolgenden Inline-Offset, der zum aktuellen Wert des Programmzeigers addiert wird.

BOOT (--)
Beim Start des Systems werden im KKF_PC noch die Interruptvektoren für Division durch 0 (INT $00) und für CTRL+C-Programmabbruch (INT $23) auf eigene Routinen umgeleitet und das CTRL+C-Flag deaktiviert.

BYE (--) 83
Nach '**BYE** werden die von **BOOT** veränderten Interrupt-Vektoren wieder auf ihre alten Werte zurückgesetzt und das Programm verlassen. Da dabei noch der aktuelle Wert von **UFLAG** an das System zurückgeliefert wird, können BATCH-Prozeduren entsprechende Flags auswerten.

CELL+ (addr1 -- addr2) ANSI
Beim KKF_PC ist **CELL+** eine ALIAS-Definition von **2+** .

CELLS (n -- addr) ANSI
Beim KKF_PC ist **CELLS** eine ALIAS-Definition von **2*** .

CHAR+ (addr1 -- addr2) ANSI
Beim KKF_PC ist **CHAR+** eine ALIAS-Definition von **1+** .

CHARS (n -- len) ANSI I
Beim KKF_PC ist **CHARS** eine ALIAS-Definition von **NOOP** und mit **IMMEDIATE** gekennzeichnet.

CLS (--) UV
Im KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden.

(command -- error)
Dieser Befehl ist im KKF_PC V1.2/0 nicht verfügbar.

CS@ (-- ptr)

CS@ liefert im KKF_PC einen Pointer an den Anfang des aktuellen Codesegments mit Adreßoffset 0. Das KK-FORTH selbst beginnt wie alle COM-Programme bei Adresse $0100.

D>PTR (d -- ptr)

Die Umrechnung des 32Bit-Wertes in eine Pointer-Adresse (seg:addr) geschieht nach folgender Formel:

seg = d / 16
addr = d - seg * 16

DS@ (-- ptr)

Im KKF_PC sind Programm, Daten und Stack im gleichen Segment und liefern deshalb den gleichen Pointer (ptr=cs:$0000).

EMIT (char --) UV,83

Im KKF_PC wird **EMIT** durch DOS-Funktion $06 abgewickelt, um keinen Abbruch durch CTRL+C zu ermöglichen. Da diese Funktion aber den ASCII-Wert $FF nicht verwenden kann, wird statt dessen $FE ausgegeben.

KEY (-- char) UV,83

Das KKF_PC verwendet zur Verhinderung eines CTRL+C-Programmabbruches die direkte Zeicheneingabe und einen eigenen Zeichenpuffer für ein Zeichen. Es können deshalb alle Tastencodes empfangen werden. Bei Sondertasten wird zuerst eine ASCII-Null und dann ein entsprechender Tastencode gelesen. **KEY** ist vom Betriebssystem umleitbar. Es muß aber auch das Programmende über diese Umleitung angegeben werden, weil nicht automatisch wieder auf die Konsole zurückgeschaltet wird.

L! (w ptr --)

Der 16Bit-Wert w wird ab der Speicheradresse ptr (seg:addr) abgelegt. Dabei wird das höherwertige Wort zuerst, aber innerhalb der Wörter zuerst das niederwertige Byte abgelegt. Die Segmentadresse wird bei der Adreßerhöhung nicht verändert.

L2! (dw ptr --)

Der 32Bit-Wert dw wird ab der Speicheradresse ptr (seg:addr) abgelegt (Reihenfolge siehe **L!**).

L2@ (ptr -- dw)

Die Umkehrung des Befehls **L2!** holt den abgelegten Wert wieder zum Datenstack.

L@ (ptr -- w)

Die Umkehrung des Befehls **L!** bringt den an der 32Bit-Adresse ptr abgelegten 16Bit-Wert w wieder zum Datenstack.

LC! (char ptr --)

Der 8Bit-Wert char wird ab der 32Bit-Adresse ptr (seg:addr) abgelegt.

LC@ (ptr -- char)

Die Umkehrung des Befehls **LC!** bringt den an der 32Bit-Adresse ptr (seg:addr) abgelegten 8Bit-Wert char wieder zum Datenstack.

LCMOVE (ptr1 ptr2 len --)
Beim KKF_PC bleiben die Segmentnummer in ptr1 und ptr2 konstant. Deshalb wird nach Erreichen der Adresse 65535 wieder bei 0 begonnen.

LCMOVE> (ptr1 ptr2 len --)
Siehe **LCMOVE** .

LDUMP (ptr len --)
Bei **LDUMP** kann direkt die Segmentadresse und der Offset angegeben werden.

LFILL (ptr len char --)
Siehe **LCMOVE** .

LIMIT (-- sys)
LIMIT ist im KKF_PC auf $0000 (gesamter Speicher) gesetzt. Er kann aber auch herabgesetzt werden. Die Adressen über **LIMIT** werden dann nicht mehr verändert.

LMOVE (ptr1 ptr2 len --)
Siehe **LCMOVE** .

LWFILL (ptr count w --)
Siehe **LCMOVE** und **L!** .

M/ (d n -- q)
Bei Division durch 0 oder bei Überlauf wird über den Interrupt-Vektor 0 eine Fehlerbehandlung eingeleitet.

M/MOD (d n -- r q)
siehe **M/** .

MAXAT (-- x y) UV
Die maximale Bildschirmgröße wird im (OUTPUT-Vektor durch direktes Auslesen der BIOS-Variablen (Speicheradresse $0040:$004A und $0040:$0084) ermittelt. Bei einigen alten Bildschirmkarten werden diese Variablen nicht gesetzt.

MOD (n1 n2 -- r) 83
Siehe **M/** .

NEXT-LINK (-- addr) SV
(NEXT ist beim KKF_PC eine 4Byte-Routine, die an jeden Assemblerbefehl direkt angehängt wurde. Um trotzdem den Debugger verwenden zu können, wurde hinter den **(NEXT**-Routinen ein Verkettungszeiger mit Anfang in **NEXT-LINK** abgelegt. Beim Aktivieren des Debuggers werden dann alle (NEXT-Routinen modifiziert.

NEXTBRANCH (-- ; R: n -- n-1 |) R
Der von **NEXT** kompilierte **NEXTBRANCH** erwartet noch ein Inline-Offset, der beim Rücksprung zum aktuellen Programmzeiger addiert wird.

P! (w addr --)
Im KKF_PC wird zuerst das niederwertige Byte in Portadresse addr und dann das höherwertige Byte in Portadresse addr+1 geschrieben.

P@ (addr -- w)
Im KKF_PC wird zuerst das niederwertige Byte aus Portadresse addr und dann das höherwertige Byte aus Portadresse addr+1 gelesen.

PAUSE (--) D
Momentan ist nur ein aktiver Task verfügbar. Trotzdem ist die Taskumschaltung schon implementiert. Die Vorbereitung der einzelnen Tasks muß aber durch ein Zusatzprogramm abgewickelt werden.

PC! (char addr --)
Es wird nur das niederwertige Byte von char in Portadresse addr geschrieben.

PC@ (addr -- char)
Es wird nur die Portadresse addr ausgelesen.

PTR>D (ptr -- d)
Die Pointer-Angabe seg:addr wird durch folgende Formel in einen 32Bit-Wert umgewandelt:
d = seg * 16 + addr

QUERY (--)
Im KKF_PC wird beim Start überprüft, ob noch zusätzliche Zeichen hinter dem Filenamen angegeben wurden. Ist dies der Fall, so wird der Rest dieser Zeile als erste Befehlszeile interpretiert.
Es ist aber darauf zu achten, daß schon das Betriebssystem die Zeichen ">", ">>", "<" und "|" als -/Ausgabeumleitung interpretiert und nur die davor abgelegten Zeichen zum Programm gelangen.

R0 (-- addr) U
R0 enthält die Endadresse+1 des Returnstacks. Beim Ablegen eines Wertes wird zuerst der Returnstackzeiger um zwei erniedrigt und dann der Wert gespeichert (Low-Byte zuerst).

RP! (addr --) R
Bei **RP!** ist direkt die Speicheradresse anzugeben, auf den der Returnstackpointer gesetzt werden soll.Man sollte immer nur den mit **RP@** geholten oder den in **R0** stehenden Wert übergeben.

RP@ (-- addr)
RP@ liefert die aktuelle Wert des Returnstackzeigers.

S0 (-- addr) U
S0 enthält die Endadresse+1 des Datenstacks. Beim Ablegen eines Wertes wird zuerst der Datenstackzeiger um zwei erniedrigt und dann der Wert gespeichert (Low-Byte zuerst).

SFLAG (-- sys) SV
Folgende Bits von **SFLAG** werden im KKF_PC V1.2/0 verwendet:
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..15: Werden noch nicht verwendet

SP! (addr --)
Bei **SP!** ist direkt die Speicheradresse anzugeben, auf den der Returnstackpointer gesetzt werden soll.Man sollte immer nur den mit **SP@** geholten oder den in **S0** stehenden Wert übergeben.

SP@ (-- w)
SP@ liefert die aktuelle Inhalt des Datenstackzeigers.

SS@ (-- ptr) X
Da beide Stacks im gleichen Segment wie Programm und Daten abgelegt sind, liefern **SS@** , **CS@** und **DS@** den gleichen Wert. Der Adreß-Offset hat dabei immer den Wert 0.

STOP? (-- f)
Alle Befehle, die **STOP?** verwenden (z.B. **DUMP** oder **WORDS**) haben Probleme bei der Eingabeumleitung. Da nach jeder Ausgabe die Tastatur abgefragt wird, werden meistens zwei Zeichen gelesen.

SYSCON (-- $0100)
Der SYSCON-Bereich beginnt am Programmanfang ab Adresse $0100.

SYSVAR (-- $0110)
Gleich hinter dem SYSCON-Bereich folgen die System-Variablen.

SYSVARLEN@ (-- $50)
Der SYSVAR-Bereich hat eine Länge von 80 Bytes oder 40 Zellen.

UFLAG (-- sys) SV
UFLAG wird beim Verlassen des Programmes an das Betriebssystem zurückgeliefert und kann in Batch-Prozeduren als Fehlernummer verwendet werden.

Anhang B

Fehlerliste

Bit 15=1 : Kompilermodus verlassen und Datenstack löschen

Fehlernummer	Art
$0000	Kein Fehler (ERROR entfernt nur den Wert)
$7fff	Fehlermeldung als CSA liegt auf dem Stack
$0001 ...	Fehler des Betriebssystem
$0001	Unbekannte Funktionsnummer
$0002	File nicht gefunden
$0003	Pfad nicht gefunden
$0004	Zu viele Files offen
$0005	Zugriff verweigert
$0006	Unbekannte Handlenummer
$0007	MCB 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 (z.B. Division durch 0)
$7f08	Nicht initialisierter Interrupt
$7f09	Fehlerhafte Adresse
$7f0a	DEFER-Definition nicht initialisiert
$7f0b	Dictionary voll
$7f0c	USER-Bereich voll
$7f0d	CONTEXT-Bereich 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 nach IS kein DEFER-Befehl
$7f16	File nicht geöffnet
$7f17	Blocknummer zu groß
$7f18	Blocknummer nicht erlaubt (z.B. 0 bei LOAD)
$7f19	Fehler bei Terminal-Befehlsübertragung
$7f1a	Fehler bei UVECTOR-Befehl (Tabelle zu kurz)
$7f1b	Befehl wird nicht unterstützt

Anhang C

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 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. Davor sollten alle vom KK-FORTH verwendeten Files geschlossen werden. Zur Sicherheit wird noch 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 vorgegeben:

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 Namen 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 D

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 bringen

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
ESC	File speichern, Editor verlassen

Zusätze bei EDITOR.COM:

F9	Eingabe der ID-Kennung
^K S	File speichern, danach weiter editieren
^K D	Editor ohne Speicherung des Files verlassen

Anhang E

Programmlisting

IO_MSDOS.SCR

```
         Screen # 0
 0 \\ Ein-/Ausgabe über MSDOS                        ( 03.06.91/KK )
 1
 2 System:         KKF_PC V1.2/0 mit MSDOS ab V2.11
 3 Änderung:       03.06.91  KK: Fileaufspaltung
 4
 5                      Hinweise:
 6  - Alle Ein-/Ausgaben laufen über Konsole und sind umleitbar
 7  - Da CTRL+C bei einigen DOS-Aufrufen zum Programmabbruch führt
 8    mußte eine Funktion verwendet werden, die sofort die Taste
 9    liefert und kein $ff ausgeben kann. Deshalb enthält LASTKEY
10    Flag und ASCII-Wert und $FF wird als $FE ausgegeben.
11  - Bei CLS, AT und AT? wird das BIOS aufgerufen
12    (Achtung: Bei Umleitung liefern sie verkehrte Werte)
13  - Direktes Fileinterface verwendet aktuelles Verzeichnis
14  - FCB-Adresse bleibt auf $80 (für (FILE-FIRST ...)
15

         Screen # 1
 0 \ Loadscreen                                      ( 03.06.91/KK )
 1
 2   &02 &03 thru      \ Variable und BDOS-Aufruf
 3   &04 &10 thru      \ Ein-/Ausgabe
 4   &11 &14 thru      \ Fileinterface
 5   &15 &16 thru      \ Vektorisierung und Initialisierung
 6
 7
 8
 9
10
11
12
13
14
15

         Screen # 2
 0 \ Variablen                                       ( 03.06.91/KK )
 1 \ Arbeitsvariable für Tastenabfrage (für MSDOS notwendig)
 2 | Variable lastkey      \ Flag und Zeichencode der letzen Taste
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

```
         Screen # 3
 0 \ Diskverwaltung: INT$10 und INT$21-Aufrufe     ( 03.06.91/KK )
 1  Assembler
 2   L: >int$10
 3     fip push,  frp push,  $10 int,
 4     frp pop,  fip pop,  ret,
 5
 6   L: >int$21f?  ( Aus Wort bei Fehler herausspringen )
 7     $21 int,
 8     u< IF, ax tos mov,  sp inc,  sp inc,  next, THEN,
 9     ret,
10   L: int$21_f   ( nur Flag zurückliefern )
11     >int$21f? # call,  tos tos xor,  next,
12   L: int$21_hf  ( wenn ok, auch AX zurückliefern )
13     >int$21f? # call,  ax push,  tos tos xor,  next,
14  Forth
15

         Screen # 4
 0 \ ((emit? ((emit ((type ((cr ((del ((bell       ( 09.08.91/KK )
 1 | ' true        Alias ((emit?   ( -- f ) ( immer möglich )
 2   Assembler   L: (((emit   ( dl=char )
 3     6 # ah mov,  $ff # dl cmp,  0= IF, dl dec, THEN,
 4                  $21 int,  ret,
 5   Forth
 6 | Code ((emit   ( char -- ) ( nur an Konsole ausgeben )
 7     tosl dl mov,  (((emit # call,  tos pop,  next,     End-Code
 8 | Code ((type   ( addr n -- ) ( nur an Konsole )
 9     tos cx mov,  w pop,
10     c0<> IF, BEGIN, w ) dl mov,  w inc,
11                      (((emit # call,     c0= UNTIL,
12     THEN, tos pop,  next,     End-Code
13 | : ((cr        ( -- )  #cr emit   $0a emit ;
14 | : ((del       ( -- )  #delout emit  #bl emit  #delout emit ;
15 | : ((bell      ( -- )  7   emit ;

         Screen # 5
 0 \ ((at?  ((at  ((maxat  scrollup  ((cls         ( 30.07.91/KK )
 1 | Code ((scrollup       ( yyxx1 yyxx2 n -- )
 2     tosl al mov,  $06 # ah mov,  dx pop,  cx pop,
 3     >int$10 # call,  tos pop,  next,                  End-Code
 4 | : ((cls       ( -- ) ( Bildschirm durch rollen löschen )
 5     0  ((maxat flip or  $0700  ((scrollup  0 0 ((at ;
 6 | : ((maxat     ( -- xmax ymax )
 7     0 $44a l@  0 $484 lc@ 1+ ;
 8 | Code ((at     ( x y -- ) ( Video-Position setzen )
 9     dx pop,  tosl dh mov,  bh bh xor,  2 # ah mov,
10     >int$10 # call,  tos pop,  next,                  End-Code
11 | Code ((at?    ( -- x y ) ( Videoposition abfragen )
12     tos push,  bh bh xor,  3 # ah mov,  >int$10 # call,
13     dh tosl mov,  dh dh xor,  dh tosh mov,
14     dx push,  next,                                   End-Code
15
```

```
         Screen # 6
 0 \ ((key?  ((key  ((string                          ( 03.06.91/KK )
 1 | Code ((key?   ( -- f ) ( Ausgabestatus holen )
 2     tos push,  p_vdp #) tos mov,  ' lastkey >body @ # tos add,
 3     1 tos d) ah mov,  ah ah or,
 4     0= IF, $ff # dl mov,  6 # ah mov,  $21 int,  0 # ah mov,
 5           0<> IF, dl ah mov,  ax tos ) mov, THEN,
 6        THEN, ah tosl mov,  ah tosh mov,  next,         End-Code
 7 | Code ((key    ( -- char ) ( Zeichen von Konsole )
 8     tos push,  p_vdp #) tos mov,
 9     ' lastkey >body @ # tos add,  tos ) ax mov,
10     ah ah or,  0= IF, 7 # ah mov,  $21 int, THEN,
11     ah ah xor,  ax tos ) mov,                ( Flag löschen )
12     ax tos mov,  next,                              End-Code
13 | : ((string    ( addr len -- )
14     0  ?DO  key over c!  1+  LOOP  drop ;
15

         Screen # 7
 0 \ ((editstring-Zusätze                          ( 29.08.91/KK )
 1 | : del's       0 ?DO #delout emit LOOP ;
 2 | : cur-left    1 del's  >r  $1.0001 d-  r> ;
 3 | : cur-right   >r >r  dup 1 -type  1+  r> 1+  r> ;
 4 | : del-line    over del's  dup spaces  del's  -  0. ;
 5 | : del-char    ( addr pos max -- addr pos max-1 )
 6     1-  dup >r  over >r  swap -  >r  dup 1+ over r@ cmove
 7     dup r@ -type space  r> 1+ del's  r> r> ;
 8 | : ins-char    ( addr pos max char -- addr+1 pos+1 max+1 )
 9     -rot  2dup 2>r  swap -  >r  over dup 1+ r@ cmove> over c!
10     dup r@ 1+ -type  r> del's  1+ 2r> $1.0001 d+ ;
11 | : saveline    ( addr pos max -- addr pos max )
12     swork  dup p_llen @ +  p_llines @ 1- p_llen @ *  cmove>
13     dup swork 1+ c! >r  dup swork c!
14     2dup -  swork 2+  r@ p_llen @ umin  move  r> ;
15

         Screen # 8
 0 \ ((editstring                                  ( 09.08.91/KK )
 1 | : getkey      ( -- char ) ( Zeichen holen )
 2     key  0 case? IF key flip THEN
 3     $4b00 case? IF $13 THEN  $4d00 case? IF $04  THEN
 4     $5300 case? IF $07 THEN  $4800 case? IF #esc THEN $ff and ;
 5
 6 | : ((editstring   ( addr maxlen pos len -- pos2 len2 )
 7     >r  over umin  swap span !  r> over umin
 8     >r  2dup -type   dup r@ - del's  r>
 9     swap >r  tuck + swap  r>                  ( addr pos max )
10     BEGIN  getkey  dup #esc =  sflag @ 4 and 0=  and
11        IF  -1 swap           ( ??? -- addr pos max -1 char )
12           BEGIN  drop  1+ p_llines @ mod  >r  del-line 2drop
13                  r@ p_llen @ * swork +  count  span @ umin  >r
14                  count  span @ umin  >r  over r@ cmove
15
```

```
         Screen # 9
 0 \ ((editstring  2. Teil                          ( 09.08.91/KK )
 1                  dup r@ -type  r> swap  r@ +  r> rot
 2                  2dup swap -  del's   r>  getkey  dup #esc <>
 3           UNTIL nip
 4        THEN
 5        CASE #cr     OF  sflag @ 4 and 0= IF saveline THEN
 6                         rot drop exit                ENDOF
 7             $13     OF over    IF cur-left THEN      ENDOF
 8             $04     OF 2dup u< IF cur-right THEN     ENDOF
 9             #brk    OF del-line                      ENDOF
10             #delin  OF over IF cur-left del-char THEN ENDOF
11             $07     OF 2dup <> IF del-char THEN      ENDOF
12                     over span @ u<  over $1f u>  and
13                     IF dup >r ins-char r> THEN
14        ENDCASE
15     REPEAT ; -2 allot

         Screen # 10
 0 \ ((query                                        ( 05.06.91/KK )
 1 | : ((query     ( -- )
 2 \ Zusatz für Interpretation der Kommandozeile
 3     taskaddr@  maxtlen@ +  2+  >tib  !
 4     sflag @  dup 1 or  sflag !   1 and 0=
 5     IF   $80  count #tib-max umin
 6          sflag @ 4 and 0=
 7          IF 2dup + over dup saveline drop 2drop THEN
 8          dup span !  tib swap  cmove
 9     ELSE tib  #tib-max  expect  space
10     THEN span @  #tib !  >in off  blk off ;
11
12
13
14
15

         Screen # 11
 0 ( Diskverwaltung: ((file? ... ((file-close      ( 30.07.91/KK )
 1 | : ((file?             ( string/0 -- error )
 2     #ro (file-open  ?dup ?exit  (file-close ;
 3
 4 | Code ((file-create    ( string/0 -- id 0 | error )
 5     tos dx mov,  cx cx xor,  $3c # ah mov,
 6     int$21_hf # jmp,                                End-Code
 7 | Code ((file-delete    ( string/0 -- error )
 8     tos dx mov,  $41 # ah mov,  int$21_f # jmp,      End-Code
 9
10 | Code ((file-open      ( string/0 attr -- id 0 | error )
11     tosl al mov,   dx pop,   $3d # ah mov,
12     int$21_hf # jmp,                                End-Code
13 | Code ((file-close     ( id -- error )
14     $3e # ah mov,  int$21_f # jmp,                  End-Code
15
```

```
         Screen # 12
 0 ( Diskverwaltung: ((file-size ... ((file-pos@   ( 03.06.91/KK )
 1 | : ((file-size         ( id -- d 0 | error )
 2     dup >r  (file-pos@       ?dup IF rdrop exit THEN
 3     0.  2  r@  (file-pos!    ?dup IF rdrop nip nip exit THEN
 4          r@ (file-pos@       ?dup IF rdrop nip nip exit THEN
 5     2swap  0  r>  (file-pos!  dup IF nip nip THEN ;
 6
 7 | Code ((file-pos!      ( d dir id -- error )
 8     ax pop,   $42 # ah mov,   cx pop,   dx pop,
 9     int$21_f # jmp,                                  End-Code
10 | Code ((file-pos@      ( id -- d 0 | error )
11     $42 # ah mov,  1 # al mov,  cx cx xor,  dx dx xor,
12     >int$21f? # call,  ax push,  dx push,
13     tos tos xor,  next,                              End-Code
14
15


         Screen # 13
 0 ( Diskverwaltung: ((file-read ... ((file-free   ( 03.06.91/KK )
 1 | Code ((file-read      ( seg addr len id -- error )
 2     cx pop,  dx pop,  ds di mov,  ds pop,
 3     $3f # ah mov,  $21 int,  u>= IF, ax ax xor, THEN,
 4     di ds mov,  ax tos mov,  next,                   End-Code
 5 | Code ((file-write     ( seg addr len id -- error )
 6     cx pop,  dx pop,  ds di mov,  ds pop,
 7     $40 # ah mov,  $21 int,  u>= IF, ax ax xor, THEN,
 8     di ds mov,  ax tos mov,  next,                   End-Code
 9 | Code ((file-free      ( dev -- free. max. 0 | error )
10     tosl dl mov,  $36 # ah mov,  >int$21f? # call,
11     cx dx xchg,  dx mul,
12     ax w mov,  bx mul,  ax push,  dx push,
13     w ax mov,  cx mul,  ax push,  dx push,
14     tos tos xor,  next,                              End-Code
15

         Screen # 14
 0 \ ((file-first  ((file-next                    ( 01.08.91/KK )
 1 | Code ((file-first   ( string/0 attr -- dta 0 | error )
 2     tos cx mov,  dx pop,  $4e # ah mov,              End-Code
 3   Assembler L: ((FF1
 4     >int$21f? # call,  $0080 # ax mov,  ax push,
 5     tos tos xor,  next,
 6   Forth
 7 | Code ((file-next   ( -- dta 0 | error )
 8     tos push,  $4f # ah mov,  ((ff1 # jmp,
 9    End-Code
10
11
12
13
14
15
```

```
         Screen # 15
 0 \ standard-i/o's                                ( 28.07.91/KK )
 1   &10 output UTable: (output
 2     noop  ((emit?  ((emit  ((type   ((cr  ((del
 3           ((bell   ((cls   ((maxat  ((at  ((at? ;
 4
 5   &05 input  UTable: (input
 6     noop  ((key?  ((key  ((string  ((editstring  ((query ;
 7
 8   &13 disc   UTable: (disc
 9     noop  ((file?       ((file-create  ((file-delete
10           ((file-open   ((file-close
11           ((file-size   ((file-pos!    ((file-pos@
12           ((file-read   ((file-write   ((file-free
13           ((file-first  ((file-next ;
14
15


         Screen # 16
 0 \ standard-io                                   ( 03.06.91/KK )
 1   : standard-io     ( -- )
 2     p_cinput  @ 6 - execute
 3     p_coutput @ 6 - execute
 4     p_cdisc   @ 6 - execute ;
 5
 6
 7
 8
 9
10
11
12
13
14
15


         Screen # 17
 0 \\ Logbuch                                      ( 03.06.91/KK )
 1
 2  03.06.91  KK: Ausgliederung des Files (ab Version 1.2/0)
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Index