Klaus Kohl

Zusatzhandbuch

zum

KKF_V20

für den

mc-PC-EMUF

Version 1.2/0

- Hinweise zum KKF_V20

- 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-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.

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 mc-PC-EMUF

Beim KKF_V20 handelt es sich um eine Anpassung des KK-FORTH an den mc-PC-EMUF mit V20-Prozessor. Es entstand eine FORTH-Version, die selbst in der kleinsten Speicherkonfiguration (32KByte EPROM und 32KByte RAM) voll funktionsfähig ist. Da aber dann ein Teil des KK-FORTH im ROM bleibt, kann der Debugger nicht genutzt werden. Um diesen Nachteil auszugleichen, sind auf den zwei mitgelieferten Disketten insgesamt vier verschiedene KKF-Versionen verfügbar:

KKF_V20S	EPROM-Version ohne Debuggermöglichkeit
KKF_V20D	EPROM-Version für Debugger (etwas langsamer)
KKF_V20M	RAM-Version (Ein-/Ausgabe und Fileinterface über Terminal)
KKF_PC	Normale PC-Version

Die Programme KKF_V20S und KKF_V20D benötigen ein 32KByte-EPROM am Speicherende und ersetzen damit das Monitor-EPROM des EMUF's. Am Anfang des zugreifbaren Speichers wird ein RAM mit mindestens 32KByte erwartet. Darin wird dann sowohl die Interrupt-Tabelle als auch veränderliche Daten und weitere Befehle untergebracht. Durch `SAVESYSTEM Filename` wird immer nur ein Zusatzimage gespeichert, daß ab Speicheradresse $4400 in das EPROM übernommen werden muß. Beim Start des KKF wird dieses Image erkannt, an die richtige Speicheradresse zurückkopiert und gestartet. Es können deshalb mit diesen Minimalversionen Autostartprogramme mit bis zu 15344 Bytes im Zusatzfile erstellt werden.

Die Programme KKF_V20M und KKF_PC setzen statt dessen ein Betriebssystem und einen freien Programmspeicher von 64KByte voraus. Die COM-Programme müssen dann, wie beim (MS)DOS üblich, ab der Adresse $0100 im gewünschten Segments abgelegt und dann gestartet werden.

Die Ein-/Ausgabe werden bei den ersten beiden KKF-Versionen direkt über die serielle Schnittstelle und bei den anderen beiden Programmen durch das Betriebssystem abgewickelt. Das Fileinterface wird nur in der letzten Version über das Betriebssystem direkt verwaltet. Die V20-Versionen verwenden statt dessen die Ein-/Ausgabebefehle und greift auf die Files des Terminalprogrammes zu.

Sowohl die Filestruktur als auch die Speicherverwaltung sind dem MSDOS entlehnt. Da aber bei den Versionen KKF_V20S und KKF_V20D das Betriebssystem nicht verwendet wird, ist eine Reservierung und Freigabe des externen Speichers nicht nötig.

1.3 Lieferumfang

Mit dieser Beschreibung erhalten Sie zwei Disketten (5.25" oder 3.5" - 360K). Die erste Diskette enthält die drei KKF_V20-Versionen mit allen Zusatzfiles. Die zweite Diskette wird zu allen Terminalversionen mitgeliefert und enthält neben dem KKF_PC mit Assembler und Druckertreiber 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_V20-Diskette

READ	ME	2888	10.09.91	11.44	
TERMINAL	COM	28828	29.08.91	16.44	Terminalprogramm (COM2-9600)
FED	COM	23999	29.08.91	16.40	FORTH-Screeneditor
-KKF_V20	S--	0	21.12.90	9.34	
KKF_V20S	COM	32768	10.09.91	11.33	Minimalversion
-KKF_V20	D--	0	21.12.90	9.34	
KKF_V20D	COM	32768	10.09.91	11:33	Minimalversion für Debugger
ASSDISDE	BIN	10114	10.09.91	11:46	Zusatzimage für KKF_V20D
-KKF_V20	M--	0	21.12.90	9.34	
KKF_V20M	COM	17100	25.08.91	23.44	Dos-Version für KKF-Terminal
--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
NECP6_T	SCR	21504	29.08.91	9.38	Druckertreiber für KKF_V20
V20TEST	SCR	8192	25.08.91	23.34	Beispielprogramme für PC-EMUF
RTCTEST	SCR	6144	25.08.91	17.49	Befehle für die Echtzeituhr
---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

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 die Programme oder dem Bootsektor der Diskette.

Besitzt der PC/AT eine Festplatte, so richtet man am besten ein eigenes Unterverzeichnis mit Namen KKF_V20 (bzw. KKF_TERM für zweite Diskette) ein. In dieses Unterverzeichnis kann man dann alle Files der ersten (zweiten) Diskette kopieren. Das Programm TERMINAL.COM bzw. FED.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 Terminalprogramm und der Editor sind 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).

Zum Arbeiten mit der Minimalversion muß das File KKF_V20S.COM bzw. KKF_V20D.COM in ein 32KByte-EPROM gebrannt und an Stelle des Monitor-EPROM's eingesetzt werden. Es verwendet die gleiche Schnittstelle und auch die gleiche Geschwindigkeit von 9600 Baud.

Bei den Versionen KKF_V20M.COM bzw. KKF_PC.COM der 2. Diskette ist die Installation vom Betriebssystem abhängig. Es muß dafür gesorgt werden, daß immer das vollständige File ab der Speicheradresse $0100 in ein eigenes 64KByte-Segment kopiert wird. Es verwendet für die Ein-/Ausgabe die DOS-Befehle für die Tastatur und kann deshalb auch umgeleitet werden. Nur bei der PC-Version wird auch das Fileinterface über das Betriebssystem realisiert. Das Programm KKF_V20M.COM erwartet weiterhin das KKF-Terminalprogramm als Empfänger der Kommandos.

Als Schnittstelle zwischen mc-PC-EMUF und dem PC-Terminal wird ein Interface-Modul (z.B. IF232) und eine serielles Kabel benötigt. Die Leitungen 2 und 3 müssen dabei gekreuzt werden. Da das Terminalprogramm den Handshake unterstützt, müssen die Leitungen 4 und 5 bzw. 6 und 20 am PC miteinander verbunden werden. Auf der EMUF-Seite können diese Pins offen bleiben.

Das Terminalprogramm ist auf die zweite Schnittstelle eingestellt. Falls andere COM-Ports verwendet werden sollen, so ist dies nach dem Start des Programmes TERMINAL.COM durch die Tastenkombination ALT+C einstellbar. Damit später die nicht verwendeten Schnittstellen unverändert bleiben, sollte die angepaßte Version mit ALT+T unter einem neuen Namen abgelegt und an Stelle von TERMINAL.COM verwendet werden.

Sollte sich das KKF_V20 nicht melden, so ist zuerst mit einem Schnittstellentester ermitteln, ob die Leitungen richtig gekreuzt sind und ob das Terminalprogramm bei Tastendruck auch sendet. Anschließend kann man mit dem ursprünglichen Monitor-EPROM ermitteln, ob die Schnittstelle des EMUF's funktioniert. Falls alles zutrifft, so kann nur ein fehlerhaftes EPROM die Ursache sein.

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 EGA-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), $0040:$004A/B (Spaltenanzahl: mindestens 80) und $0040:$0084 (Zeilenanzahl-1: mindestens 24) gesetzt sein.

1.5 Das Arbeiten mit dem KKF_V20

Bei der Erstellung oder beim Test neuer Programme für die Minimalversion sollte immer das KKF_V20D verwendet werden. Es hat den Vorteil, daß es auch für den Debugger geeignet ist. Falls man sich das ständige Nachladen von Assembler, Disassembler und Debugger ersparen will, kann man auch das mitgelieferte File ASSDISDE.BIN ab der EPROM-Adresse $4400 übernehmen. Beim Einschalten wird dann dieses Image erkannt und die darin enthaltenen Befehle wieder ins RAM geladen. Man sollte aber dann keine Applikationen damit erstellen, da der noch freie EPROM-Bereich mit 5230 Bytes doch schon sehr klein ist.

Bei `SAVESYSTEM Filename` wird bei den Versionen KKF_V20S und KKF_V20D nur das Zusatzimage gespeichert. Es kann dann ab der EPROM-Adresse $4400 in die Kopie des ursprünglichen EPROM's eingefügt werden. Es darf aber nicht länger als 15344 Bytes sein, da es sonst den Sprungvektor bei $7FF0 überschreiben würde.

Die RAM-Versionen KKF_V20M und KKF_PC speichern bei `SAVESYSTEM Filename` ein vollständiges Programm einschließlich KKF-Kern. Es kann deshalb an Stelle des ursprünglichen Programmes in ein EPROM übernommen oder direkt von Diskette gestartet werden.

Kapitel 2

Beschreibung

Die folgenden Beschreibungen beziehen sich hauptsächlich auf das Programm KKF_V20D . Bis auf die etwas anderen Adressen und der statt dem (NEXT-Aufruf direkt eingebundene Routine ist diese Version mit KKF_V20S identisch.

2.1 Allgemeines zum KKF_V20

Die Versionen KKF_V20S/D und KKF_V20M/PC unterscheiden sich erheblich, weil die ersten beiden Programme eigenständige EPROM-Versionen und die letzten beiden RAM-Versionen für ein Betriebssystem sind.

2.1.1 KKF_V20S.COM und KKF_V20D.COM

Das KKF_V20S und KKF_V20D sind KKF-Versionen, bei dem der Kern in einem 32KByte-EPROM ab Adresse $F800:$0000 steht und ab Adresse $0000:$0000 ein 32KByte-RAM erwartet wird. Durch diese Konfiguration kann das KKF_V20 in einer einzigen 64K-Bank (Segmentadresse $F800:0000) laufen.

Als Schnittstelle verwenden beide Minimalversionen den RS232-Port 1 mit 9600 Baud und 8 Datenbit. Da diese Angaben im SYSVAR-Bereich gespeichert sind, können sie sowohl im EPROM als auch bei Erstellung des Zusatzimage geändert werden.

Das erste KByte des RAM's werden durch KKF_V20S/D nicht belegt. Sie sind für die Interrupt-Vektoren reserviert und werden auf eine Routine umgeleitet, die alle Interrupts verhindert und eine Fehlermeldung ausgibt. Eine andere Fehlermeldung wird nur bei dem durch Divisionsüberlauf ausgelösten Interrupt 0 ausgegeben.

Der einzige Unterschied zwischen den beiden Programmen ist die Realisierung der (NEXT-Routine, also die Ausführung des nächsten FORTH-Befehls. Im KKF_V20S ist diese Routine wie im KKF_PC eine 4-Byte-Sequenz. Da aber das EPROM nicht verändert werden kann, gibt es keine Zugriffmöglichkeit für den Debugger. Es wird deshalb auf eine Verkettung der (NEXT-Routinen verzichtet.

Für Tests mit Debugger gibt es die Version KKF_V20D. Sie hat statt der 4-Byte-Sequenz einen Sprung an das Ende des RAM's. Dort steht dann eine durch den Debugger veränderbare (NEXT-Routine. Sie wird beim Start des Systems zusammen mit dem SYSVAR-Bereich kopiert.

2.1.2 KKF_V20M.COM

Diese Version erwartet wie auch KKF_PC das Betriebssystem. Es läuft vollständig im RAM und hat alle Eigenschaften und Befehle des KKF_PC. Die einzige Ausnahme ist die Behandlung der Files. Sie wird wie bei den anderen KKF_V20-Versionen über das Terminal abgewickelt. Dieses Programmn ist deshalb nur dann zu nehmen, wenn für Ein-/Ausgabe weiterhin eine serielle Schnittstelle und auf der anderen Seite das Terminalprogramm verwendet wird.

2.1.3 KKF_PC.COM

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.

Bei Aufruf des COM-Programmes werden zusätzliche Eingaben vom Betriebssystem erkannt und dem Programm an mitgegeben. Das KK-FORTH erkennt dies und verwendet einen vorhandenen Text als erste Eingabezeile. Dadurch lassen sich KKF_PC-Programme aus BATCH-Prozeduren mit Parameter versorgen.

2.2 Speicheraufteilung

Die folgenden Adressen beziehen sich auf das KKF_V20D V1.2/0. Beim KKF_V20S fehlt die (NEXT-Routine und die Bereiche von $F220 bis $FFFF liegen deshalb um $10 Bytes höher. Programme sollten diese Adressangaben nie direkt verwenden, sondern immer aus Variablen oder Konstanten ermitteln.

$0000	SYSCON	Anfang des SYSCON-Bereiches
$0010		Grundparameter des SYSVAR-Bereiches
$0060		Dictionary-Anfang
$4400		Anfangsadresse des Zusatzimage
$7FFF		Ende des EPROM's
$8000		RAM-Anfang mit Interrupt-Vektoren
$8400	(SYSVAR	Kopie des SYSVAR-Bereiches
$8450	HERE	Anfang des freien Programmbereiches
$F220	VDP @	Anfang des Variablenbereiches
$F25C	HDP @	Anfang des Heap (beim Start leer)
$F25C	TDP @	Anfang des Taskbereiches
$F5B6	TASK0	Anfang des USER-Bereiches im TASK0
$F90E	FIRST	Diskpuffer
$FD10	SYSVAR $2C + @	Anfang des Puffers für 8 Befehlszeilen
$FFA0	SYSVAR $30 + @	Anfang des Anwender-Arbeitsbereich (leer)
$FFA0	SYSVAR	SYSVAR-Bereich
$FFF0		(NEXT-Routine bei KKF_V20D.COM
$0000	LIMIT	Programmende+1

2.2.1 Systemkonstanten

Die System-Konstanten enthalten Werte, die nur beim Systemstart berücksichtigt werden. Dabei ist beim KKF_V20S/D sowohl die Image als auch die RAM-Anfangsadresse veränderbar. Die Endadresse des RAM's kann nicht verändert werden, weil sowohl die (NEXT-Routine als auch der SYSVAR-Bereich eine feste Adresse benötigt.

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

Die Adresse des Zusatzimage ist mit $4400 vorgegeben. Sie könnte beim KKF_V20D sogar auf $4000 heruntergesetzt werden. Beim Kaltstart untersucht das KKF_V20D, ob an der angegebenen Stelle ein Zusatzimage vorliegt. Ist dies der Fall, so wird es wieder an die richtige RAM-Adresse kopiert und die ebenfalls im Image gespeicherten Variablen- und USER-Bereiche übernommen. Es können Autostart-Programme erstellt werden, weil die Zeiger der DEFER-Wörter ebenfalls im Variablenbereich untergebracht sind.

Die Anfangsadresse des RAM-Bereiches ist mit $8400 vorgegeben. Dadurch kann bis auf die Interrupt-Vektoren der gesamte RAM-Speicher für das Programm verwendet werden. Man kann diesen Wert durch direkte Veränderung der EPROM-Adressen $000C/D (Low-Byte zuerst) auf einen höheren Wert setzen (z.B. $C000) und hat dann einen vom KK-FORTH nicht veränderbaren Speicher zur Verfügung.

Bei den Versionen KKF_V20M und KKF_PC ist die Belegung es SYSCON-Bereiches schon im Handbuch beschrieben. Es kann als einziges die RAM-Endadresse heruntergesetzt werden.

2.2.2 Systemvariablen

Beim Start des KKF_V20S/D werden entweder die vorgegebenen Systemvariablen oder die Werte im Zusatzimage übernommen. Der SYSVAR-Bereich befindet sich am Ende des RAM's. Die Kopie des SYSVAR-Bereiches wird an den Anfang des RAM's übertragen und dient gleichzeitig als Header für das Zusatzimage. Um keinen Programmabsturz bei Verwendung eines falschen Image zu verursachen, sind im Header auch die Angaben der FORTH-Version (Typ, Betriebssystem, Versionsnummer) enthalten.

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

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

SYSVAR + 6 **Hardwarekennung/Betriebssystem**
$0122 Version KKF_V20D (V20S: $0121; V20M: $0123; PC: $0211)

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

SYSVAR + 10 **Attribut-Wort**

```
%0101000000000010
 ^^              RAM/ROM-Version
  ^^             Segmentierter Speicherverwaltung
   ^^            Ohne Floatingpoint-Unterstützung
     ^           Ein-/Ausgabe über eigene Schnittstelle
      ^          Fileinterface über Terminal
       ^^        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**
$0010 Zeigt beim Start auf Orginal-Image, später auf RAM-Anfang

SYSVAR + 14 **VOC-LINK**
$1E2F Zeigt auf das als einziges vorhandene FORTH-Vokabular

SYSVAR + 16 **DP**
$3F54 Der KKF_V20D-Kern hat eine Länge von 16212 Bytes

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

SYSVAR + 20 **VLEN**
$003C Im KKF_V20-Kern werden 60 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 **SWLEN (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**
$000C Für 9600 Baud vorbereitet

SYSVAR + 64 **Reserviert für SIO-Portadresse**
$03F8 Portadresse der SIO1

SYSVAR + 66 **frei**

SYSVAR + 68 **frei**
SYSVAR + 70 **frei**
SYSVAR + 72 **frei**
SYSVAR + 74 **frei**

SYSVAR + 76 **SFLAG**
Bit 0=1: Keine Start-Befehlszeile verfügbar (immer gesetzt)
Bit 1=1: Keine Warnung durch **CREATE** ausgeben
Bit 2=1: Keine Zeilenpuffer verwenden
Bit 3=1: Schnittstelle ist schon initialisiert (KKF_V20S/D)
Bit 4-15: Werden beim KKF_V20/PC nicht genutzt

SYSVAR + 78 **UFLAG**
Frei für Anwender

Die 5 nicht belegten Wörter unterhalb von **SFLAG** können für eigene Zwecke verwendet werden. Sie haben den Vorteil, daß ihre Speicheradressen unveränderlich sind.

Für die serielle Schnittstelle wird normalerweise COM1 mit 9600 Baud verwendet. Um dies sehr einfach ändern zu können, werden sowohl die Portadresse als auch die Teilerangabe für die Baudrate im SYSVAR-Bereich aufbewahrt. Wie andere Baudraten und Portadressen einzustellen sind, wird später noch beschrieben.

2.2.3 Der Taskbereich

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

Da alle Speicheradressen beim Kaltstart aus den 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_V20D führen zu folgender Speicheraufteilung:

$F25C	WDP@	Anfang des Arbeitsbereich für diesen Task - Ablagebereich für WORD
	…	
		- Arbeitsbereich für <# bis #>
$F2B0	PAD	Anfang des Stringbereich - 256 Bytes für Strings - Platz für 256 Stackeinträge
$F5B0	S0 @	Ende des Datenstacks (danach 6 Bytes frei)
$F5B6	TASKADDR@	Anfangsadresse des USER-Bereiches - Programm/Daten für Taskwechsel - USER-Variablen
$F6B6	TASKADDR@ $100 +	Eingabepuffer für **QUERY** (= TIB - 2)
	…	
		- Platz für 256 Returnstackeinträge
$F908	R0 @	Ende des Returnstacks (danach 6 Bytes frei)

2.2.4 Diskpuffer, System- und Anwender-Arbeitsspeicher

Am Ende des verwendeten Speichers befindet sich der Diskpuffer und die 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 der Speicher bei Verwendung eines FORTH-Screens überschrieben wird.

Sytem-Arbeitsbereich

Wenn man das unveränderte KK-FORTH 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 dieser Puffer um 82 Bytes nach oben geschoben und die neue Eingabezeile zusammen mit der Längenangabe und der letzten 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 durch Interrupts und bei der Initialisierung einige Einträge benötigt.

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 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 in den CFA's angegebenen Adressen (NEST bis (DIC handelt es sich um kurze Assemblerroutinen des KKF-Kern. Sie sind für die Art des Befehls verantwortlich.

:-Definitionen am Beispiel :

```
F800:26AF  91 26 01 3A 8C 00 EA 03
```

$26AF/B0	$2691	Verkettung zum vorherigen Befehl
$26B1	$01	Namensfeldadresse (Länge: 1 Zeichen)
$26B2	$3A	Befehlsname :
$26B3/4	$008C	Adresse von (NEST
$26B5/6	$03EA	Highlevel-Teil (beginnt mit CURRENT)

Konstanten am Beispiel #B/BLK

```
F800:0227  1D 02 06 23 42 2F 42 4C 4B BE 00 00 04
```

$0227/8	$021D	Verkettung zum vorherigen Befehl
$0229	$06	Namensfeldadresse (Länge: 6 Zeichen)
$022A ...	$23 ...	Befehlsname #B/BLK
$0230/1	$00BE	Adresse von (CON
$0232/3	$0400	Wert der Konstanten

32Bit-Konstanten am Beispiel PI

```
$3.1415 2Constant pi

F800:92A0  60 92 02 50 49 B3 00 15 14 03 00
```

$92A0/1	$9260	Verkettung zum vorherigen Befehl
$92A2	$02	Namensfeldadresse (Länge: 2 Zeichen)
$92A3/4	$50 $49	Befehlsname PI
$92A5/6	$00B3	Adresse von (2CON
$92A7..A	$0003.1415	Wert der 32Bit-Konstanten

Variablen am Beispiel FILE-ID

```
F800:04C6  BA 04 07 46 49 4C 45 2D 49 44 A8 00 00 00
```

$04C6/7	$04BA	Verkettung zum vorherigen Befehl
$04C8	$07	Namensfeldadresse (Länge: 7 Zeichen)
$04C9 ...	$46 ...	Befehlsname FILE-ID
$04D0/1	$00A8	Adresse von (VAR
$04D2/3	$0000	Offset in den Variablenbereich

32Bit-Variablen am Beispiel COUNTER

```
2Variable counter

F800:92AB  A0 92 07 43 4F 55 4E 54 45 52 A8 00 4A 00
```

$92AB/C	$92A0	Verkettung zum vorherigen Befehl
$92AD	$07	Namensfeldadresse (Länge: 7 Zeichen)
$92AE ...	$43 ...	Befehlsname COUNTER
$92B5/6	$00A8	Adresse von (2VAR (identisch mit (VAR)
$92B7/8	$004A	Offset in den Variablenbereich

USER-Variablen am Beispiel >IN

```
F800:0410  06 04 03 3E 49 4E 01 01 3A 00
```

$0410/1	$0406	Verkettung zum vorherigen Befehl
$0412	$03	Namensfeldadresse (Länge: 3 Zeichen)
$0413 ...	$3E ...	Befehlsname >IN
$0416/7	$0101	Adresse von (USER
$0418/9	$003A	Offset in den USER-Bereich

Vokabulare am Beispiel FORTH

```
F800:1E25  08 1E 05 46 4F 52 54 48 0C 01 00 00 1E 00
```

$1E25/6	$1E08	Verkettung zum vorherigen Befehl
$1E27	$05	Namensfeldadresse (Länge: 5 Zeichen)
$1E28 ...	$46 ...	Befehlsname FORTH
$1E2D/E	$010C	Adresse von (VOC
$1E2F/30	$0000	Verkettung zum nächsten Vokabular (fehlt)
$1E31/2	$001E	Offset in den Variablenbereich
(VDP @ $1E + ergibt $F23E)		
$F23E/F	$3B46	Zeiger zur BOOT-Linkfeldadresse

DEFER-Definitionen am Beispiel ERRORTEXT@

```
F800:37C6  B5 37 0A 45 52 52 4F 52 54 45 58 54 40 C5 00 30 00
```

$37C6/7	$37B5	Verkettung zum vorherigen Befehl
$37C8	$0A	Namensfeldadresse (Länge: 10 Zeichen)
$37C9 ...	$45 ...	Befehlsname ERRORTEXT@
$37D3/4	$00C5	Adresse von (DEFER
$37D5/6	$0030	Offset in den Variablenbereich
(VDP @ $30 + ergibt $F250)		
$F250/1	$37D7	Zeigt auf die versteckte Routine

Die Routine zu **ERRORTEXT@** beginnt unmittelbar hinter dem Befehlsheader.

LABEL-Definitionen am Beispiel HEREADDR

```
label hereaddr

F800:F01A  AB 92 A8 48 45 52 45 41 44 44 52 16 F0
F800:F016  17 01 B9 92
```

$F01A/B	$92AB	Verkettung zum vorherigen Befehl
$F01C	$A8	Namensfeldadresse (Länge: 8 Zeichen) (Befehl ist Immediate und Indirect)
$F01D ...	$48	Befehlsname HEREADDR
$F025/6	$F016	Zeiger auf Codefeldadresse (im Heap)
$F016/7	$0117	Adresse von (LABEL
$F018/9	$92B9	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

```
F800:11A3  90 11 A7 41 4C 49 47 4E 45 44 77 00
```

$11A3/4	$1190	Verkettung zum vorherigen Befehl
$11A5	$A7	Namensfeldadresse (Länge: 7 Zeichen)

		(Befehl ist Immediate und Indirect)
$11A6 ...	$41	Befehlsname ALIGNED
$11AD/E	$0077	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

F800:8450  46 3B 0B 33 44 2D 43 4F 4E 53 54 41 4E 54 8C 00
F800:8460  67 25 6B 07 2E 11 5C 07 2E 11 2E 11 EE 26 E8 71
F800:8470  7C CD 07 66 0D 5C 07 DC 09 CD 07 66 0D 5C 07 DC
```

$8450/1	$3B46	Verkettung zum vorherigen Befehl
$8452	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8453 ...	$33	Befehlsname 3D-CONSTANT
$845E/F	$008C	Adresse von (NEST
$8460 ...	$2567	Create-Teil (endet mit (;code)
$846E ... $8470	$E8 $71 $7C	(does> call,
$8471 ...	$07CD	Does>-Teil (endet mit exit)

```
F800:8485  50 84 03 31 32 33 6E 84 01 00 02 00 03 00
```

$8485/6	$8450	Verkettung zum vorherigen Befehl
$8487	$03	Namensfeldadresse (Länge: 3 Zeichen)
$8488 ...	$31	Befehlsname 123
$848B/C	$846E	Zeiger auf Does>-Teil von 3D-CONSTANT
$848D/E	$0001	x-Wert
$848F/90	$0002	y-Wert
$8491/2	$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

F800:8493  85 84 0B 33 44 2D 56 41 52 49 41 42 4C 45 8C 00
F800:84A3  3B 26 4B 01 00 00 6C 11 4B 01 00 00 6C 11 4B 01
F800:84B3  00 00 6C 11 EE 26 E8 34 7C 5C 07 DD 08 7C 09 97
```

$8493/4	$8485	Verkettung zum vorherigen Befehl
$8495	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8496 ...	$33	Befehlsname 3D-VARIABLE
$84A1/2	$008C	Adresse von (NEST
$84A3 ...	$263B	Create-Teil (endet mit (;code)
$84B9 ... $84BB	$E8 $34 $7C	(vdoes> call,
$84BC ...	$075C	VDoes>-Teil (endet mit exit)

```
F800:84C4  93 84 07 56 45 43 54 4F 52 31 B9 84 3C 00
```

$84C4/5	$8493	Verkettung zum vorherigen Befehl
$84C6	$07	Namensfeldadresse (Länge: 7 Zeichen)
$84C7 ...	$56	Befehlsname VECTOR1
$84CE/F	$84B9	Zeiger auf VDoes>-Teil von 3D-VARIABLE
$84D0/1	$003C	Offset in den Variablenbereich

CODE-Definition am Beispiel DROP

```
F800:0853  3B 08 04 44 52 4F 50 5C 08 5B E9 90 F7
```

$0853/4	$083B	Verkettung zum vorherigen Befehl
$0855	$04	Namensfeldadresse (Länge: 4 Zeichen)
$0856 ...	$44 ...	Befehlsname DROP
$085A/B	$085C	Zeigt auf nachfolgenden Code
$085C	$5B	tos push,
$085D	$E9 $90 $F7	(next jmp,

Bei allen anderen KKF_V20/PC-Versionen wird die (NEXT-Routine direkt eingefügt.

PROC-Definition am Beispiel DOSCALL

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

F800:84D2  C4 84 07 44 4F 53 43 41 4C 4C A1 00 CD 21 C3
```

$84D2/3	$8D58	Verkettung zum vorherigen Befehl
$84D4	$07	Namensfeldadresse (Länge: 7 Zeichen)
$84D5 ...	$44 ...	Befehlsname DOSCALL
$84DC/D	$00A1	Adresse von (DIC
$84DE/F	$CD $21	$21 int,
$84E0	$C3	ret,

2.4 Informationen zu KKF_V20-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.

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_V20/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 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. Es 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 schon beim 8086-Prozessor eine höhere Geschwindigkeit bei der Programmausführung erreicht werden könnte, wird beim allen KKF_V20- und der KKF_PC-Version 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

Der Prozessor besitzt 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 Werte 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 Speicherbefehle 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_V20 sind Aufrufe der entsprechenden (MS)DOS-Routine. 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

Beim KKF_V20S.COM und KKF_V20D.COM sind die Fehlermeldungen im EPROM untergebracht. Falls dort neue Texte übernommen werden sollen, so müssen mit einem geeigneten Monitorprogramm (oft beim EPROM-Programmierer vorhanden) folgende Schritte durchgeführt werden:

- Erstellen und Speicherung einer neuen Fehlermeldungs-Tabelle
  * Alte Dictionaryende merken (`$20 @ hex u.`)
  * Restlänge des Image merken (`$24 @ $100 + hex u.`)
  * Anfang der alten Fehlertabelle ermitteln (`>errortext @ hex u.`)
  * Mit CREATE ERRTEXT den Anfang einer Tabelle definieren
  * Fehlernummer und Texte mit **,** und **$,** kompilieren
  * Mit einem `0 ,` die Tabelle abschließen
  * Tabelle abspeichern (Länge merken : `here errtext - hex u.`)

```
" NEWTEXT.BIN" 1+ #rw (file-create error
dup ds@ drop errtext rot over here swap - swap (file-write error
(file-close error
```

- EPROM verändern
  * Rest des Image (Länge siehe oben) an neue Adresse bringen
    Quellenadresse: Dictionaryende
    Zieladresse: Alte Fehlertabelle-Adresse + neue Tabellenlänge
    (Zieladresse wird später für **DP** gebraucht)
  * Neue Fehlertabelle ab alter Fehlertabelle-Adresse laden
  * **DP** (Adresse $20) auf Zieladresse des Imagerest setzen (Low-Byte zuerst)

Da das Zusatzimage erst bei $4400 beginnt, bleibt auch für eine größere Tabelle noch genügend Platz. Nötigenfalls kann auch noch die Image-Adresse heraufgesetzt werden.

Beim KKF_V20M.COM und dem KKF_PC sind die Texte zu den Fehlermeldungen am Ende des KKF-Kerns untergebracht. Man kann deshalb, wie im Handbuch beschrieben, nach Freigabe des belegten Speichers mit `here >errortext @ - allot` einfach die neuen Fehlermeldungen kompilieren.

2.5 Zusatzhinweise

Ein Blick hinter die "Kulissen" des KKF-Systems soll in dem letzten Kapitel der Beschreibung des KKF_V20-Kerns die Initialisierungssequenz und die Ein-/Ausgabe verdeutlichen. Die Realisierung der Schnittstellenbefehle in den einzelnen KKF_V20-Versionen ist als Listing im Anhang E angegeben. Da es für den Targetkompiler konzipiert ist, besitzt es Vorwärtsreferenzen und kann deshalb nicht direkt geladen werden.

2.5.1 KKF_V20S.COM und KKF_V20D.COM

Der KKF-Kern bleibt bei beiden Versionen im EPROM. Nur die Systemvariablen, der USER-Bereich und ein evtl. vorhandenes Zusatzimage wird wieder an die richtige Adresse kopiert.

Initialisierung

Der genaue Ablauf der Initialisierung geschieht wie folgt:

- Initialisierungen: DI und CLD (keine Interrupts, Richtung aufsteigend)
- Alle Segmente auf Wert des Codesegments setzen
- Stack vor SYSVAR-Bereich setzen
- Suche ob ein Zusatzimage vorhanden ist
  * Ins RAM übertragen, wenn Zieladresse von Orginaladresse abweicht
  * wenn Image fehlt, dann RAM-Anfang verwenden
- Endadresse des Image ermitteln
- Adressen von UWORK, SWORK und FIRST ermitteln
- Alle Tasks kopieren und dabei die Adressen korrigieren
- Heap- und Variablenbereich kopieren
- SYSVAR-Bereich noch in den RAM-Anfang kopieren
- FORTH mit Task0 starten
- Interrupt 0 auf eigene Fehlermeldung umleiten
- Alle anderen Interruptvektoren initialisieren
- Filevariablen und Zeilenpuffer löschen, **SINGLETASK**
- Schnittstelle initialisieren
- '**BOOT** , '**COLD** und dann **ABORT** aufrufen

Speicherbelegung

Für das Minimalsystem werden nur 32KByte EPROM am Speicherende und ebenfalls 32KByte RAM am Speicheranfang verwendet. Alle anderen Speicherbereiche des Prozessors werden durch den KKF-Kern nicht verändert, können aber durch die Pointerbefehle angesprochen werden. Da keine Bereiche reserviert werden müssen, entfallen die Befehle **(MALLOC** , **(MRELOC** und **(MFREE** .

Da ein nicht initialisierter Interrupt zu einem Programmabsturz führen würde, wird die Interrupttabelle am Speicheranfang (Größe: $400 Bytes) vom KKF_V20S/D freigehalten und mit einer Routine belegt, die nach Abschaltung aller Interrupts eine entsprechende Fehlermeldung ausgibt.

Ein-/Ausgabe

Da kein Betriebssystem vorhanden ist, muß die Ein-/Ausgabe direkt vom KK-FORTH durchgeführt werden. Dazu wird bei Systemstart oder bei jeder Umstellung auf die alten Routinen **(OUTPUT** bzw. **(INPUT** untersucht, ob die Schnittstelle schon initialisiert wurde. Dies ist durch das gesetzte Bit 3 in **SFLAG** zu erkennen. Ist dieses Bit noch auf 0, so wird die ebenfalls im SYSVAR-Bereich angegebene Schnittstelle initialisiert. Bei den ausgelieferten Programmen werden dabei die RS232-Schnittstelle COM1 verwendet und auf 9600 Baud eingestellt.

Durch Veränderung der beiden Variablen im SYSVAR-Bereich können auch andere RS232-Ports (2:$02F8; 3:$03E8; 4:$02E8) oder andere Baudraten (Teiler = 115200/Baudrate) eingestellt werden. Bei Tests mit einem IF232-Modul und einem TurboLite110-Laptop (9.6MHz-V20) gab es bei kurzen Leitungen auch mit 115200 Baud keine Probleme. Man sollte sich aber mit unkritischen 38400 Baud (Teilerwert 3) begnügen.

Man kann auch nach dem Start des FORTH noch die verändern und dann in Applikationen beibehalten. Dazu ist nur die Änderung der vorgegebenen Werte und nach dem Löschen des Initialisierungsbit in **SFLAG** ein erneuter Aufruf des Ausgabevektors notwendig. Man muß dann natürlich auch das Terminalprogramm auf die neue Baudrate einstellen.
Beispiel: Umstellen der Geschwindigkeit auf 38400 Baud

```
$03  sflag $0e -  !            ( Teilerangabe )
sflag @  $fff7 and  sflag !    ( Flag löschen )
(output                              ( SIO initialisieren )
```

2.5.2 KKF_V20M.COM und KKF_PC.COM

Beim Start des KKF_V20M bzw. KKF_PC werden durch die BOOT-Routine ein Flag des Betriebssystems und zwei Interruptvektoren verändert. Erst beim Verlassen des KK-FORTH 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 dann vergrößert werden, wenn keine weiteren Datenfelder belegt worden sind. Der Grund dazu ist, daß jeder angeforderte Speicher einen Header mit 16 Bytes bekommt, in dem Informationen zum Speichersegment 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 und der Veränderung eines Interrupt-Vektors (INT $23) und des CTRL+C-Flags kann dies verhindert werden. Eine Sicherung des CTRL+C-Interrupt ist nicht notwendig, da er bei Programmstart automatisch gerettet wird.

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_V20M und das KKF_PC einen Zeichenpuffer. Sowohl **KEY** als auch **KEY?** fragen deshalb zuerst den Puffer ab und rufen erst dann die DOS-Funktion auf, wenn kein Zeichen vorhanden ist.

Das Betriebssystem kann die Tastaturabfrage auch auf Files umleiten. Da aber diese Umleitung für das KK-FORTH transparent ist und vom Betriebssystem nicht automatisch am Fileende wieder rückgängig gemacht wird, muß im File auch der Befehl für das Beenden des Programmes untergebracht sein.

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_V20 kann durch die Angabe `>Filename` oder `>Device` die Ausgabe auf ein File oder ein anderes Gerät umgeleitet werden. Da dies für das KK-FORTH transparent ist, können weiterhin nur die Werte $00 bis $FE ausgegeben werden.

Filebefehle

Wie auch in Kapitel 2.4.5 angesprochen, sind alle Filebefehle des KKF_PC 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.

Im KKF_V20M wird statt dessen ein Terminalprogramm vorausgesetzt. Da die normale Ein-/Ausgabe als Schnittstelle verwendet wird, kann diese FORTH-Version auch auf einem PC laufen, würde aber bei Filebefehle nach dem zweiten Tastendruck eine Fehlermeldung ausgeben.

Kapitel 3

Zusatzprogramme

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

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 nachfolgender 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 aufgeführt. 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 einige Register auch mit FORTH-Namen versehen:

```
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. Bei Zahlenwerte oder Adreßangaben muß deshalb immer ein # oder ein #) folgen.

AH ... DI	8Bit- und 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 möglich. 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 Sprünge oder bei Unterprogrammaufrufe ist eine 16Bit-Adresse anzugeben. Sollte einmal innerhalb eines Befehls ein bedingter oder unbedingter Sprung eingebunden werden, so sind folgende Kontrollstrukturen und Bedingungen zu verwenden:

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

0= 0<> 0< 0>=	(Vergleich mit 0)
< >= <= >	(Vorzeichenbehaftete Vergleiche)
u< u>= u<= u>	(Vorzeichenlose Vergleiche)
ov nov	(Überlauf)
<>c0= c0= ?c0= c0<>	(Bedingungen für Repeat-Anweisungen)

Der (NEXT,-Makrobefehl

Durch **NEXT,** wird eine 4-Byte-Sequenz kompiliert, die den nächsten FORTH-Befehl ausführt. Dies ist bei fast allen Code-Definitionen notwendig, die nicht auf andere FORTH-Worte des Kerns springen.

In den RAM-Version KKF_V20M und KKF_PC wird hinter der Sequenz noch ein Verkettungszeiger abgelegt, der bei **NEXT-LINK** beginnt und für den Debugger benötigt wird.

Im KKF_V20D wird durch **NEXT,** ein Sprung auf die (NEXT-Routine am RAM-Ende erzeugt. Deshalb kann in dieser Version auch ohne Verkettung durch **NEXT-LINK** der Debugger genutzt werden.

Die NEXT,-Sequenz besteht aus folgenden Assembleranweisungen:

lods,	(Nächste CFA in AX bringen)
ax w xchg,	(AX mit DI tauschen)
w) jmp,	(Sprung auf die gewünschte Assemblerroutine)

Verwendung von (USER-)Variablen

Bei Verwendung von Variablen in Assembler-Routinen ist zu beachten, daß sich die Adresse ändert. Es muß deshalb immer der Offset in den Variablenbereich gespeichert und zum aktuellen Inhalt der Systemvariable **VDP** addiert werden.

Beispiel: Zähler erhöhen

```
Variable co
```

```
Code co+  ( Zähler erhöhen )
      vdp #) w mov,            ( Variablenanfang )
      co vdp @ - w d) inc,     ( Wert erhöhen )
      next,                    ( zurück zum FORTH )
 End-Code  ( Ende der Definition )
```

Wie man am Beispiel sieht, haben Systemvariablen eine feste Adresse. Bei USER-Variablen ist die Anfangsadresse in der Systemvariable **TASKADDR** gespeichert und muß wie bei Variablen zum Offset addiert werden.

Falls Interruptprogramme mit Variablen arbeiten, müssen sie während der Veränderung des Variablen- und Heap-Bereiches gestoppt werden. Oft genügen für Interruptparameter die an festen Adressen gebundenen Speicherplätze unterhalb von **SFLAG** .

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 nur eingeschränkt für KKF_V20S verwendbare 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.

Da das KKF_V20S im EPROM läuft und keine (NEXT-Routine im RAM hat, kann der Debugger nicht eingreifen. Sollte ein Programm für die Minimalversion getestet werden, so ist das etwas langsamere KKF_V20D zu verwenden.

3.4 NECP6_T.SCR

Bei diesem File handelt es sich um die Anpassung des auf der zweiten Diskette befindlichen Druckertreibers NECP6.SCR . Es nützt dabei die Möglichkeit des Terminalprogrammes, über Handlenummer 4 direkt zum Drucker zu gelangen.

Ich habe dieses Programm als Beispiel für Ausgabeumleitung mitgegeben. Darüber hinaus zeigt es, daß man aufpassen muß, wenn die Filebefehle über die gewöhnliche Ein-/Ausgabeschnittstelle abgewickelt werden. Im Beispiel wird durch Neudefinition alter Befehle auf den alten OUTPUT-Vektor zurückgeschaltet.

3.5 V20TEST.SCR

Dieses File enthält einige kleine Bespiele für den mc-PC-EMUF. Es wurde dabei auf die Verwendung des Assemblers verzichtet. Jeder der nachfolgend beschriebenen Screens verfügt auch noch über zusätzliche Komentare.

Baudrate umstellen

Im Screen 2 ist die schon oben beschriebene Sequenz als Befehl **BAUD** verfügbar. Das Wort erwartet noch die Teilerangabe, verändert aber nicht die Portnummer der Schnittstelle.

baud	(teiler --)	Baudrate umstellen (3=38400)

Watchdog zurücksetzen

Der mc-PC-EMUF verfügt über einen Watchdog-Timer, der durch Schließen des Jumpers J1 aktiviert werden kann. Es ist dann aber ein ständiger Reset der Watchdog notwendig. Der im Screen 3 definierte Befehl **WD-RESET** ist deshalb in Programme mindestens einmal pro Sekunde aufzurufen.

wd-reset	(--)	Watchdog zurücksetzen

Verwendung der COM2-Schnittstelle

Die zweite serielle Schnittstelle ist ebenfalls schon in der Grundversion des mc-PC-EMUF's verfügbar. Mit den in Screen 4 untergebrachten Befehle kann diese Schnittstelle initialisiert, der Ein-/Ausgabestatus ermittelt und Zeichen gelesen oder geschrieben werden.

sio2-init	(teiler --)	Schnittstelle initialisieren (3=38400)
sio2-emit?	(-- f)	Liefert Ausgabestatus
sio2-emit	(char --)	Ausgabe eines Zeichens
sio2-key?	(-- f)	Liefert Flag, ob Zeichen verfügbar ist
sio2-key	(-- char)	Holt ein Zeichen

Verwendung des Timer 0

Im Chip M1101 sind auch 3 Timer verfügbar. Die im Screen 5 definierten Befehle verwenden den auch sonst für Timerzwecke genutzten Timer 0 zur Bestimmung der Zeitdifferenz. Dazu wird der Timer auf 0 gesetzt und nach einer beliebigen Zeit wieder ausgelesen.

#ti-ctrl	(-- $43)	Kontrollportadresse
#ti0	(-- $40)	Portadresse des Timer 0
ti0-start	(--)	Timer 0 starten
ti0@	(-- n)	Timer 0 auslesen

Tonausgabe

Sowohl der PC-EMUF als auch der normale PC nutzen den Timer 2 und einen Port des Bausteins 8255 (im M1101 enthalten) zur Ansteuerung des Lautsprechers. Durch die im Screen 6 untergebrachten Befehle kann der Lautsprecher ein-/ausgeschaltet und in seiner Tonhöhe verändert werden. Die Lautstärke ist leider nicht veränderbar.

beep-on	(--)	Lautsprecher einschalten
beep-off	(--)	Lautsprecher ausschalten
ti2!	(n --)	Tonhöhe f einstellen (f proportional 1/n)

3.6 RTCTEST.SCR

Es gehört auch eine Echtzeituhr zur Hardware des mc-PC-EMUF's. Die durch das File RTCTEST.SCR zur Verfügung gestellten Befehle erlauben das Auslesen und Setzen dieser Uhr. Dabei wird aber ein Datenformat verwendet, daß beim PC in der Filebehandlung verwendet wird. Dieses Format gibt Datum und Uhrzeit in einem 32Bit-Wert mit einer Genauigkeit von zwei Sekunden an.

rtcinit	(--)	Initialisieren der Uhr
rtcset	(String ; --)	Uhr setzen (String: wjjmmddhhmmss)
time@	(-- dosdate.)	Datum/Uhrzeit holen
time!	(dosdate. --)	Datum/Uhrzeit setzen
dosdate.	(dosdate. --)	Datum/Uhrzeit ausgeben

Beim ersten Mal muß die Echtzeituhr Initialisiert werden. Beim Setzen kann entweder durch **RTCSET String** ein String mit Datumsangabe oder mit **TIME!** ein DOSDATE-Format übergeben werden. Die Zeit 11:32:46 des 10.09.91 (Dienstag=2) ist als String "2910910113246" anzugeben. Bei der Ausgabe des mit **TIME@** geholten Wertes durch **DOSDATE.** wird das Format "10.09.91 11:32:46" erzeugt.

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 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.

3.9 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.

3.10 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
#DELIN liefert 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 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)
Dieser nur in den Versionen KKF_V20M und KKF_PC verfügbare 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. Dieser Befehl ist nur in den Versionen KKF_V20M und KKF_PC verfügbar.

(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. Dieser Befehl ist nur in den Versionen KKF_V20M und KKF_PC verfügbar.

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

,C (w --) DX
Dieser Befehl ist im KKF_V20/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_V20M und KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden. Im KKF_V20S und KKF_V20D benötigt dieser Befehl ein KKF-Terminal zur Befehlsinterpretation.

AT? (-- x y) UV
Im KKF_V20M und KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden. Im KKF_V20S und KKF_V20D benötigt dieser Befehl ein KKF-Terminal zur Befehlsinterpretation.

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_V20M und KKF_PC noch die Interruptvektoren für Divisionüberlauf (INT $00) und für CTRL+C-Programmabbruch (INT $23) auf eigene Routinen umgeleitet und das CTRL+C-Flag deaktiviert.

BYE (--) 83
Nach '**BYE** werden im KKF_V20M/PC die von **BOOT** veränderten Interrupt-Vektoren und das CTRL+C-Flag 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
Bei KKF_V20S/D wird das FORTH durch Rücksprung auf Adresse 0 neu gestartet. Alle nicht vom KK-FORTH benötigten Register und die Hardware bleiben unverändert.

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

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

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

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

CLS (--) UV
Im KKF_V20M und KKF_PC wird diese Funktion durch direkten Aufruf einer BIOS-

Grafikfunktion realisiert und kann deshalb im Betriebssystem nicht umgeleitet werden. Im KKF_V20S und KKF_V20D benötigt dieser Befehl ein KKF-Terminal zur Befehlsinterpretation.

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

CS@ (-- ptr)
CS@ liefert einen Pointer an den Anfang des aktuellen Codesegments mit Adreßoffset 0. KKF_V20M und KKF_PC selbst beginnt wie alle COM-Programme bei Adresse $0100. Die Versionen KKF_V20S und KKF_V20D beginnen bei Adresse 0 im Segment $F800.

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 und in den KKF_V20-Versionen sind Programm, Daten und Stack im gleichen Segment und liefern deshalb den gleichen Pointer (ptr=cs:$0000).

EMIT (char --) UV,83
Im KKF_V20M und 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 und das KKF_V20M 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_V20 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 die Segmentadresse und der Offset angegeben werden.

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

LIMIT (-- sys)
LIMIT ist auf $0000 (gesamter Speicher) gesetzt. Er kann aber in den Versionen KKF_PC und KKF_V20M 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
Beim KKF_PC und beim KKF_V20M wird die maximale Bildschirmgröße durch direktes Auslesen der BIOS-Variablen (Speicheradresse $0040:$004A und $0040:$0084) ermittelt. Bei einigen alten Bildschirmkarten werden diese Variablen nicht gesetzt. Im KKF_V20S und KKF_V20D benötigt dieser Befehl ein KKF-Terminal zur Befehlsinterpretation.

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

NEXT-LINK (-- addr) SV

(NEXT ist 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.
Die Version KKF_V20S läuft im EPROM und kann nicht verändert werden. Es wurde deshalb auf die Verkettung mit **NEXT-LINK** verzichtet. Statt dessen kann aber die etwas langsamere Version KKF_V20D verwendet werden, die eine einzelne (NEXT-Routine am Ende des RAM's besitzt.

NEXTBRANCH (-- ; R: n -- n-1 |) R

Der von **NEXT** kompilierte **NEXTBRANCH** erwartet noch einen Inline-Offset, der für den Rücksprung zum aktuellen Programmzeiger addiert wird.

P! (w addr --)

Es wird zuerst das niederwertige Byte in Portadresse addr und dann das höherwertige Byte in Portadresse addr+1 geschrieben.

P@ (addr -- w)

Es 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_V20M und beim KKF_PC wird vom Betriebssystem 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 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=1:	Schnittstelle wurde schon initialisiert
Bit 4..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)
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 (-- sys)
Der SYSCON-Bereich beginnt immer am Programmanfang ab Adresse $0100 (KKF_V20M und KKF_PC) bzw. Adresse 0 (KKF_V20S und KKF_V20D).

SYSVAR (-- sys)
Gleich hinter dem SYSCON-Bereich an Adresse $0110 bzw. $0010 folgen die System-Variablen.

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

UFLAG (-- sys) SV
UFLAG wird von KKF_V20M und von KKF_PC 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

Programmlistings

IO_SIO.SCR

```
         Screen # 0
 0 \\ Interface-Routinen für die PC-SIO           ( 03.06.91/KK )
 1
 2 System:         KKF_V20S/D V1.2/0
 3 Änderung:       03.06.91  KK: File an COM1 angepaßt
 4
 5                      Hinweise:
 6  - Portadresse und Baudrate extern vorgeben:
 7     portaddr  :  $03f8 für COM1    $02f8 für COM2
 8     baud      *  Baudrate  =  115200
 9
10  - Fileinterface ebenfalls über SIO
11
12
13
14
15

         Screen # 1
 0 \ Loadscreen                                  ( 03.06.91/KK )
 1
 2   &02 &05 thru      \ Kernroutinen
 3   &06 &12 thru      \ Ein-/Ausgabe
 4   &13 &16 thru      \ Fileinterface
 5   &17 &18 thru      \ Vektorisierung und Initialisierung
 6
 7
 8
 9
10
11
12
13
14
15

         Screen # 2
 0 \ sio-init                                    ( 03.08.91/KK )
 1 \ Nur einmal initialisieren
 2 | Code sio-init     ( -- ) ( COM1 initialisieren )
 3     sflag #) ax mov,  $08 # ax test,
 4     0= IF, $08 # ax or,  ax sflag #) mov,         \ Flag setzen
 5            p_portaddr #) dx mov,  3 # dx add,
 6            $80 # al mov,  dx byte out,         \ BRR inschalten
 7            2 # dx sub,   p_baud #) ax mov,           \ Baudrate
 8                     al ah xchg,  dx byte out,        \ Highbyte
 9            dx dec,  al ah xchg,  dx byte out,         \ Lowbyte
10            3 # dx add,
11            $0a07 # ax mov,  dx out,      \ 8bit, noP, +RTS +OUT
12        THEN,
13     next,
14    End-Code
15
```

```
         Screen # 3
 0 \ Prozeduren (((emit?  (((key?                  ( 03.08.91/KK )
 1  Assembler
 2   L: (((emit?   ( verändert DX, AX, Flags )
 3     p_portaddr #) dx mov,  5 # dx add,  dx in,
 4 \   $1020 # ax and,  $1020 # ax cmp,  ret,      \ mit Handshake
 5     $0020 # ax and,  $0020 # ax cmp,  ret,     \ ohne Handshake
 6
 7   L: (((key?    ( verändert DX, AX, Flags )
 8     p_portaddr #) dx mov,  5 # dx add,  dx in,
 9     $0001 # ax and,  $0001 # ax cmp,  ret,
10  Forth
11
12
13
14
15


         Screen # 4
 0 \ ((emit? ((emit ((type                         ( 03.08.91/KK )
 1 | Code ((emit?   ( -- f ) ( nur Abfrage )
 2     tos push,  tos tos xor,  (((emit? # call,
 3     0= IF,  tos dec,  THEN,  next,                    End-Code
 4
 5 | Code ((emit   ( char -- ) ( wartet, bis Bereit )
 6     BEGIN,  (((emit? # call,  0= UNTIL,
 7     tosl al xchg,  5 # dx sub,  dx byte out,
 8     tos pop,  next,                                   End-Code
 9
10 | Code ((type   ( addr n -- )
11     di pop,  tos inc,
12    BEGIN, tos dec,  0= IF,  tos pop,  next,  THEN,
13     BEGIN,  (((emit? # call,  0= UNTIL,
14     di ) al mov,  5 # dx sub,  dx byte out,
15     di inc,  AGAIN,                                   End-Code


         Screen # 5
 0 \ ((cr ((del ((bell ((cls ((maxat ((at ((at?    ( 09.08.91/KK )
 1 | : ((cr        ( -- )
 2     #cr emit   $0a emit ;
 3 | : ((del       ( -- )
 4     #delout emit  #bl emit  #delout emit ;
 5 | : ((bell      ( -- )
 6     7   emit ;
 7
 8 | : ((cls       ( -- ) ( Bildschirm durch rollen löschen )
 9     $0101 command! error ;
10 | : ((maxat     ( -- xmax ymax ) ( Default: 40*8 )
11     $0104 command! error  com>w  com>w ;
12 | : ((at        ( x y -- ) ( nächste Zeile einrücken )
13     $0103 command!  ?dup IF nip nip error THEN  w>com  w>com ;
14 | : ((at?       ( -- x y ) ( Videoposition abfragen )
15     $0102 command! error  com>w  com>w ;
```

```
         Screen # 6
 0 \ ((key?  ((key  ((string                          ( 03.06.91/KK )
 1 | Code ((key?   ( -- f )
 2     tos push,  tos tos xor,  (((key? # call,
 3     0= IF,  tos dec,  THEN,  next,                      End-Code
 4
 5 | Code ((key    ( char -- )
 6     tos push,  BEGIN,  (((key? # call,  0= UNTIL,
 7     5 # dx sub,  dx byte in,  al tosl mov,
 8     tosh tosh xor,  next,                               End-Code
 9
10 | Code ((string   ( addr n -- )
11     di pop,  tos inc,
12    BEGIN, tos dec,  0= IF,  tos pop,  next,  THEN,
13     BEGIN,  (((key? # call,  0= UNTIL,
14     5 # dx sub,  dx byte in,  al di ) mov,
15     di inc,  AGAIN,                                     End-Code


         Screen # 7
 0 \ ((editstring-Zusätze                             ( 09.08.91/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                          ( 25.08.91/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                                        ( 01.06.91/KK )
 1 | : ((query     ( -- )
 2 \ Zusatz für Interpretation der Kommandozeile
 3     sflag @  dup 1 or  sflag !   1 and 0=
 4     IF   $80 count  sflag @ 4 and 0=
 5          IF 2dup + over dup saveline drop 2drop THEN
 6          span !  >tib !
 7     ELSE taskaddr@  maxtlen@ +  2+   >tib  !
 8          tib  #tib-max  expect  space
 9     THEN span @  #tib !  >in off  blk off ;
10
11
12
13
14
15


         Screen # 11
 0 \ w>com ... com>$                                ( 03.08.91/KK )
 1 | : w>com       ( w -- ) ( Wort an Terminal )
 2     dup flip emit  emit ;
 3 | : d>com       ( d -- ) ( 32Bit-Wert an Terminal )
 4     w>com  w>com ;
 5 | : $>com       ( addr -- ) ( String zum Terminal )
 6     BEGIN count dup emit 0= UNTIL drop ;
 7
 8 | : com>w       ( -- w ) ( Wort vom Terminal )
 9     key flip  key or ;
10 | : com>d       ( -- d ) ( 32Bit-Wert vom Termial )
11     com>w  com>w  swap ;
12 | : com>$       ( addr -- ) ( String vom Termial )
13     1- BEGIN 1+  key  2dup swap c!  0= UNTIL drop ;
14
15
```

```
         Screen # 12
 0 \ command!                                    ( 09.08.91/KK )
 1 | : key_err?    ( com -- com | error mit EXIT )
 2     key?  IF  drop err_command!  rdrop  exit  THEN ;
 3
 4   : command!    ( com -- error ) ( Befehl senden )
 5     key_err?  #esc emit  key_err?       ( ESC-Befehl schicken )
 6     w>com  com>w ;
 7
 8 \ Bei COMMAND!-Error ein oder zwei Parameter verwerfen
 9 | : 1com!       command!  ?dup IF nip rdrop exit THEN ;
10 | : 2com!       command!  ?dup IF nip nip rdrop exit THEN ;
11
12
13
14
15


         Screen # 13
 0 \ Diskverwaltung: ((file? ... ((file-close      ( 01.08.91/KK )
 1 | : ((file?            ( string/0 -- error )
 2     $0201 1com!  $>com  com>w ;
 3
 4 | : ((file-create       ( string/0 -- id 0 | error )
 5     $0202 1com!   $>com   com>w  dup ?exit
 6     com>w  swap ;
 7 | : ((file-delete       ( string/0 -- error )
 8     $0203 1com!   $>com   com>w ;
 9
10 | : ((file-open         ( string/0 attr -- id 0 | error )
11     $0204 1com!   w>com  $>com   com>w  dup ?exit
12     com>w  swap ;
13 | : ((file-close        ( id -- error )
14     $0205 1com!   w>com   com>w ;
15


         Screen # 14
 0 \ Diskverwaltung: ((file-size ... ((file-pos@   ( 01.06.91/KK )
 1 | : ((file-size         ( id -- d 0 | error )
 2     $0206 1com!  w>com   com>w  dup ?exit  com>d  rot ;
 3
 4 | : ((file-pos!         ( d dir id -- error )
 5     $0207 command!  ?dup IF >r  drop drop 2drop  r>  exit THEN
 6     w>com  w>com  d>com  com>w ;
 7
 8 | : ((file-pos@         ( id -- d 0 | error )
 9     $0208 1com!  w>com   com>w  dup ?exit  com>d  rot ;
10
11
12
13
14
15
```

```
         Screen # 15
 0 \ Diskverwaltung: ((file-read ... ((file-free   ( 25.08.91/KK )
 1 | : ((file-read         ( seg addr len id -- error )
 2     $0209 command!  ?dup IF  nip nip nip nip  exit  THEN
 3     w>com  dup w>com   com>w ?dup  IF  nip nip nip  exit  THEN
 4     0 ?DO 2dup key -rot lc! 1+ LOOP 2drop 0 ;
 5
 6 | : ((file-write        ( seg addr len id -- error )
 7     $020a command!  ?dup IF  nip nip nip nip  exit  THEN
 8     w>com  dup w>com  0 ?DO 2dup lc@ emit 1+ LOOP 2drop
 9     com>w ;
10
11 | : ((file-free         ( dev -- free. max. 0 | error )
12     $020d 1com!   w>com   com>w ?dup ?exit
13     com>d  com>d  0 ;
14
15


         Screen # 16
 0 \ ((file-first  ((file-next                      ( 01.08.91/KK )
 1 | : ((file-first        ( string/0 attr -- dta 0 | error )
 2     $020b 2com!   w>com  $>com   com>w dup ?exit
 3     pad  dup key string  swap ;
 4
 5 | : ((file-next         ( -- dta 0 | error )
 6     $020c command!  dup ?exit
 7     pad  dup key string  swap ;
 8
 9
10
11
12
13
14
15


         Screen # 17
 0 \ standard-i/o's                                 ( 03.08.91/KK )
 1   &10 output UTable: (output
 2     sio-init  ((emit?  ((emit  ((type   ((cr  ((del
 3               ((bell   ((cls   ((maxat  ((at  ((at? ;
 4
 5   &05 input  UTable: (input
 6     sio-init  ((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 # 18
 0 \ standard-io  v  l                              ( 01.08.91/KK )
 1   : standard-io     ( -- )
 2     p_cinput  @ 6 - execute
 3     p_coutput @ 6 - execute
 4     p_cdisc   @ 6 - execute ;
 5
 6 ( Editor des Terminalprogrammes aufrufen )
 7 | : ((ed        ( scr offset -- )
 8    file-fcb @  >r  close   $0001 command!
 9    IF   2drop
10    ELSE swap w>com w>com  r@ 1+ $>com  com>w drop
11    THEN r> (open ;
12   : v           ( -- )          scr @  r# @   ((ed ;
13   : l           ( scr -- )              0      ((ed ;
14
15
```

IO_V20M.SCR

```
         Screen # 0
 0 \\ Ein-/Ausgabe über MSDOS -> Terminal          ( 18.07.91/KK )
 1
 2 System:         KKF_V20M V1.2/0 mit MSDOS ab V2.11
 3 Änderung:       18.07.91  KK: Fileinterface über Terminal
 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  - CLS, AT, AT? und das Fileinterface gehen zum Terminal
12
13
14
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 \ ((emit? ((emit ((type                         ( 18.07.91/KK )
 1 \ Arbeitsvariable für Tastenabfrage (für MSDOS notwendig)
 2 | Variable lastkey      \ Flag und Zeichencode der letzen Taste
 3
 4 | ' true        Alias ((emit?   ( -- f ) ( immer möglich )
 5   Assembler   L: (((emit   ( dl=char )
 6     6 # ah mov,  $ff # dl cmp,  0= IF, dl dec, THEN,
 7                  $21 int,  ret,
 8   Forth
 9 | Code ((emit   ( char -- ) ( nur an Konsole ausgeben )
10     tosl dl mov,  (((emit # call,  tos pop,  next,     End-Code
11 | Code ((type   ( addr n -- ) ( nur an Konsole )
12     tos cx mov,  w pop,
13     c0<> IF, BEGIN, w ) dl mov,  w inc,
14                     (((emit # call,     c0= UNTIL,
15     THEN, tos pop,  next,     End-Code
```

```
         Screen # 3
 0 \ ((cr ((del ((bell ((cls ((maxat ((at ((at?    ( 03.08.91/KK )
 1 | : ((cr        ( -- )
 2     #cr emit   $0a emit ;
 3 | : ((del       ( -- )
 4     #delout emit  #bl emit  #delout emit ;
 5 | : ((bell      ( -- )
 6     7   emit ;
 7
 8 | : ((cls       ( -- ) ( Bildschirm durch rollen löschen )
 9     $0101 command! error ;
10 | : ((maxat     ( -- xmax ymax ) ( Default: 40*8 )
11     $0104 command! error  com>w  com>w ;
12 | : ((at        ( x y -- ) ( nächste Zeile einrücken )
13     $0103 command!  ?dup IF nip nip error THEN  w>com  w>com ;
14 | : ((at?       ( -- x y ) ( Videoposition abfragen )
15     $0102 command! error  com>w  com>w ;


         Screen # 4
 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 # 5
 0 \ ((editstring-Zusätze                           ( 03.06.91/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 # 6
 0 \ ((editstring                                  ( 03.06.91/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 # 7
 0 \ ((editstring  2. Teil                         ( 03.06.91/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 # 8
 0 \ ((query                                       ( 18.07.91/KK )
 1 | : ((query     ( -- )
 2 \ Zusatz für Interpretation der Kommandozeile
 3     sflag @  dup 1 or  sflag !   1 and 0=
 4     IF   $80 count  sflag @ 4 and 0=
 5          IF 2dup + over dup saveline drop 2drop THEN
 6          span !  >tib !
 7     ELSE taskaddr@  maxtlen@ +  2+   >tib  !
 8          tib  #tib-max  expect  space
 9     THEN span @  #tib !  >in off  blk off ;
10
11
12
13
14
15
```

```
         Screen # 9
 0 \ w>com ... com>$                                ( 03.08.91/KK )
 1 | : w>com       ( w -- ) ( Wort an Terminal )
 2     dup flip emit  emit ;
 3 | : d>com       ( d -- ) ( 32Bit-Wert an Terminal )
 4     w>com  w>com ;
 5 | : $>com       ( addr -- ) ( String zum Terminal )
 6     BEGIN count dup emit 0= UNTIL drop ;
 7
 8 | : com>w       ( -- w ) ( Wort vom Terminal )
 9     key flip  key or ;
10 | : com>d       ( -- d ) ( 32Bit-Wert vom Termial )
11     com>w  com>w  swap ;
12 | : com>$       ( addr -- ) ( String vom Termial )
13     1- BEGIN 1+  key  2dup swap c!  0= UNTIL drop ;
14
15


         Screen # 10
 0 \ command!                                       ( 03.08.91/KK )
 1 | : key_err?    ( com -- com | error mit EXIT )
 2     key?  IF  drop err_command!  rdrop  exit  THEN ;
 3
 4   : command!    ( com -- error ) ( Befehl senden )
 5     key_err?  #esc emit  key_err?       ( ESC-Befehl schicken )
 6     w>com  com>w ;
 7
 8 \ Bei COMMAND!-Error ein oder zwei Parameter verwerfen
 9 | : 1com!       command!  ?dup IF nip rdrop exit THEN ;
10 | : 2com!       command!  ?dup IF nip nip rdrop exit THEN ;
11
12
13
14
15


         Screen # 11
 0 \ Diskverwaltung: ((file? ... ((file-close      ( 01.08.91/KK )
 1 | : ((file?             ( string/0 -- error )
 2     $0201 1com!   $>com  com>w ;
 3
 4 | : ((file-create       ( string/0 -- id 0 | error )
 5     $0202 1com!   $>com   com>w  dup ?exit
 6     com>w  swap ;
 7 | : ((file-delete       ( string/0 -- error )
 8     $0203 1com!   $>com   com>w ;
 9
10 | : ((file-open         ( string/0 attr -- id 0 | error )
11     $0204 1com!   w>com   $>com   com>w  dup ?exit
12     com>w  swap ;
13 | : ((file-close        ( id -- error )
14     $0205 1com!   w>com   com>w ;
15
```

```
         Screen # 12
 0 \ Diskverwaltung: ((file-size ... ((file-pos@   ( 18.07.91/KK )
 1 | : ((file-size         ( id -- d 0 | error )
 2     $0206 1com!  w>com   com>w  dup ?exit  com>d  rot ;
 3
 4 | : ((file-pos!         ( d dir id -- error )
 5     $0207 command!  ?dup IF >r  drop drop 2drop  r>  exit THEN
 6     w>com  w>com  d>com  com>w ;
 7
 8 | : ((file-pos@         ( id -- d 0 | error )
 9     $0208 1com!  w>com   com>w  dup ?exit  com>d  rot ;
10
11
12
13
14
15


         Screen # 13
 0 \ Diskverwaltung: ((file-read ... ((file-free   ( 25.08.91/KK )
 1 | : ((file-read         ( seg addr len id -- error )
 2     $0209 command!  ?dup IF  nip nip nip nip  exit  THEN
 3     w>com  dup w>com   com>w ?dup  IF  nip nip nip  exit  THEN
 4     0 ?DO 2dup key -rot lc! 1+ LOOP 2drop 0 ;
 5
 6 | : ((file-write        ( seg addr len id -- error )
 7     $020a command!  ?dup IF  nip nip nip nip  exit  THEN
 8     w>com  dup w>com  0 ?DO 2dup lc@ emit 1+ LOOP 2drop
 9     com>w ;
10
11 | : ((file-free         ( dev -- free. max. 0 | error )
12     $020d 1com!   w>com   com>w ?dup ?exit
13     com>d  com>d  0 ;
14
15


         Screen # 14
 0 \ ((file-first  ((file-next                     ( 01.08.91/KK )
 1 | : ((file-first        ( string/0 attr -- dta 0 | error )
 2     $020b 2com!   w>com  $>com   com>w dup ?exit
 3     pad  dup key string  swap ;
 4
 5 | : ((file-next         ( -- dta 0 | error )
 6     $020c command!  dup ?exit
 7     pad  dup key string  swap ;
 8
 9
10
11
12
13
14
15
```

```
         Screen # 15
 0 \ standard-i/o's                                ( 29.07.91/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  v  l                             ( 01.08.91/KK )
 1   : standard-io     ( -- )
 2     p_cinput  @ 6 - execute
 3     p_coutput @ 6 - execute
 4     p_cdisc   @ 6 - execute ;
 5
 6 ( Editor des Terminalprogrammes aufrufen )
 7 | : ((ed        ( scr offset -- )
 8    file-fcb @  >r  close   $0001 command!
 9    IF   2drop
10    ELSE swap w>com w>com r@ $>com com>w drop
11    THEN r> (open ;
12   : v           ( -- )          scr @  r# @   ((ed ;
13   : l           ( scr -- )             0      ((ed ;
14
15
```

Index