Klaus Kohl

Zusatzhandbuch

zum

KKF_FG

Version 1.2/0

- Hinweise zum KKF_FG

- Beschreibung der Zusatzprogramme

- Informationen zum Prozessor RTX-2000/1A

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

Einleitung 5
1.1 Vorwort 5
1.2 Das KK-FORTH für RTX-2000/1A 5
1.3 Lieferumfang 6
1.4 Installation des KKF_FG 7
1.5 Das Arbeiten mit dem KKF_FG 7

Beschreibung 9
2.1 Allgemeines zum KKF_FG 9
2.2 Speicheraufteilung 9
2.2.1 Systemkonstanten 10
2.2.2 Systemvariablen 10
2.2.3 Der Taskbereich 13
2.2.4 Diskpuffer und andere Arbeitsspeicher 13
2.3 Aufbau der Befehle 14
2.4 Informationen zu KKF_FG-Befehle 18
2.4.1 32Bit-Adressen 18
2.4.2 Adreß-Align 19
2.4.3 Länge der Datentypen 19
2.4.4 Portbefehle 20
2.4.5 Filebefehle 21
2.4.6 Fehlermeldungen 21
2.4.7 Kontrollstrukturen 21
2.4.8 Kompilieren der Codefeldadresse 21
2.4.9 Daten- und Returnstackbefehle 22
2.5 Zusatzhinweise 23
2.5.1 Initialisierung des Systems 23
2.5.2 Ein-/Ausgabe und das Fileinterface 23
2.6 Informationen zum RTX-2000/1A 24

Zusatzprogramme 26
3.1 RTXDIS.SCR 26
3.2 UCODES.SCR 26
3.3 OPTICOMP.SCR 27
3.4 NECP6_T.SCR 28
3.5 RTXTEST.SCR 28
3.6 SIEVE.SCR 28
3.7 NECP6.SCR 29
3.8 MEM_EDIT.SCR 29
3.9 PC_TERM.SCR 30

Glossar-Zusatz 31

Fehlerliste 38

Terminalbefehle..... 39

Editor-Tastenbelegung 40

Programmlisting 41

Index 48

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 RTX-2000/1A

Das KKF_FG ist eine Anpassung des KK-FORTH an die FORTH-Prozessoren RTX-2000 und RTX-2001A. Obwohl es sich nicht um eine Weiterentwicklung des vom gleichen Autor erstellten FG-FORTH ist, läuft es auf den gleichen Boards (FG- und mc-Risc-Board). Es werden sowohl identische Speicheraufteilung (bei BOOT=1: Lesen aus ROM) als auch die gleiche serielle Schnittstelle (über G-Bus-Adresse $1F-Bit0/1 mit Handshake) verwendet.

Da es sich um eine FORTH-Version der KKF-Serie handelt, hat es folgende Vorteile:

- Stark erweiterten Befehlssatz
- Vorbereitet für Interrupt und Multitasking
- Über Terminalprogramm kann auf PC-Files zugegriffen werden
- Leichte Anpassung von KKF-Programmen

Das KKF_FG-Programm wird beim Einschalten von EPROM auf die immer vollständig zu bestückende RAM-Bank 0 übertragen und dann nur mit dem RAM weitergearbeitet. Die anderen RAM-Bänke können mit RAM oder EPROM bestückt werden und sind durch spezielle Befehle (**2@** , **2!** ...) zugreifbar. Diese Bänke können im Prinzip auch für Programme genutzt werden, jedoch ist der Interpreter/Kompiler dafür (noch) nicht vorgesehen. Da alle Interrupts des Prozessors auf Bank 0 gehen, ist am Speicherende eine Interrupt-Tabelle eingerichtet und so initialisiert, daß nach Abschaltung aller Interrupts eine Fehlermeldung ausgegeben wird.

Abweichend vom FG-FORTH ist das KKF_FG so realisiert, daß automatisch beim Start der vorhandene Prozessor (RTX-2000 oder RTX-2001A) erkannt wird. Es kann also ohne Änderung des KKF_FG-Kerns oder des Programmes an Stelle des teueren RTX-2000 auch der billigere RTX-2001A eingesetzt werden. Dabei wird auch die geringere Stacktiefe des Prozessors berücksichtigt. Die Multiplikation wird immer im Einzelschrittverfahren, also ohne die im RTX-2000 vorhandene Hardware durchgeführt.

Allen Befehle des KKF_FG retten die benötigten Register (DPR, MD und SR) und schreiben sie danach wieder zurück. Nur bei der vorzeichenbehafteten Multiplikation **M*** und bei **PICK** wird der Interrupt für einige Takte gesperrt. Man kann deshalb für Interruptprogramme nach Retten des Carry-Flag im Konfigurations-Register unter Beachtung der belegten Speicherbereiche alle FORTH-Befehle verwenden. Nur sollte man dabei auf eine serielle Datenübertragung verzichten, da häufig aufgerufene oder lange Interruptprogramme die Baudrate verfälschen.

Die Baudrate der Schnittstelle wird wie beim FG-FORTH durch Empfang des ersten Bytes (sollte RETURN sein) erkannt. Dabei können bei einem mit 6 MHz getakteten Prozessor sogar bis zu 115200 Baud gesendet oder empfangen werden. Für komfortables Arbeiten reichen die noch unkritischen 38400 Baud, auf die das Terminalprogramm eingestellt sind. Bei der Übertragung der Daten vom Terminal auf das KK-FORTH wird mit Hardware-Handshake gearbeitet.

1.3 Lieferumfang

Mit dieser Beschreibung erhalten Sie zwei Disketten (5.25" oder 3.5" - 360K). Die erste Diskette enthält das in zwei EPROM's zu übernehmende Binärfile KKF_FG.COM und allen Zusatzfiles zu dieser KKF-Version. Die zweiten 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_FG-Diskette

READ	ME	2011	29.08.91	16.49	
TERMINAL	COM	28828	29.08.91	16.48	Terminalprogramm (COM2-38400)
FED	COM	23999	29.08.91	16.40	FORTH-Screeneditor
--KKF_FG	---	0	21.12.90	9.34	
KKF_FG	COM	16338	29.08.91	15.19	Bitimage des KKF_FG-Kerns
--FILES-	---	0	21.12.90	9.34	
RTXDIS	SCR	34816	27.08.91	20.13	Disassembler für RTX-2000/1A
UCODES	SCR	14336	28.08.91	17.37	Einige UCode-Definitionen
OPTICOMP	SCR	11264	29.08.91	15.47	Optimierender Kompiler
RTXTEST	SCR	4096	28.08.91	17.41	Interrupt-Beispielprogramm
NECP6_T	SCR	21504	29.08.91	9.38	Druckertreiber für KKF_FG
---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_PC-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_FG

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 Disketten.

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

Vor dem Start muß zuerst noch das Binärfile aufgespalten und in zwei EPROM's gebrannt werden. Dabei werden die geraden Adressen (also Byte 0, 2, ...) in das Highbyte- und die ungeraden Adressen (also Byte 1, 3, ...) in das Lowbyte-EPROM übernommen. Dabei genügen wegen der Filelänge von $3FD2 Bytes schon zwei 8K-EPROM's. Meistens werden aber 32KByte-EPROM's eingesetzt, da sie im Preis fast gleich sind und genügend Platz für Autostart-Programme oder KKF-Versionen mit Zusatzbefehle bietet.

Obwohl der mitgelieferte Editor FED.COM und der Editor des Terminalprogramms direkt auf den Bildschirmspeicher greifen, dürfte es nur wenige Probleme mit der verwendeten Hardware geben. Es wird sowohl der Bildschirm-Modus 3 (80 Zeichen, mindestens 25 Zeilen) als auch der Hercules-Modus 7 unterstützt. Jedoch müssen vom BIOS die 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_FG

Nachdem man die RS232-Leitung (2/3 und 4/5 gekreuzt) an PC und FG- bzw. mc-RISC-Board angeschlossen hat, muß man das Programm TERMINAL.COM starten und falls nötig mit der Tastenkombination ALT+C an die entsprechende Schnittstelle anpassen. Nach dem Einschalten der Stromversorgung für das RTX-Board muß nach dem ersten Drücken der ENTER-Taste die Einschaltmeldung

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

erscheinen. Kommen verstümmelte Zeichen an, so sollte man zuerst die Masse-Verbindung der RS232 kontrollieren oder die Leitung verkürzen. Bei langsamen RS232-Treibern genügt oft das Heruntersetzen der Baudrate (ALT+B). In allen Fällen muß vor der nächsten Übertragung des Returnzeichens die Reset-Taste gedrückt werden.

Kommen überhaupt keine Zeichen an, so sollte man mit einem RS232-Tester kontrollieren, ob die Leitungen richtig gekreuzt sind und ob das Terminalprogramm überhaupt sendet. Falls sich der Status der Daten- und Handshakeleitung des FG-Boards beim Reset oder nach dem Einschalten nicht auf den richtigen Pegel einstellt, so kann auch ein falsch gebranntes oder falsch eingesetztes EPROM die Fehlerursache sein.

Wenn der KKF_FG-Kern läuft, so kann wie im Handbuch beschrieben gearbeitet werden. Das Terminalprogramm sendet alle von Tastatur eingegebenen Zeichen an das KKF_FG und stellt die empfangenen Zeichen am Bildschirm dar. Transparent für den Anwender werden dabei auch die

Befehle zur Filebehandlung übertragen. Man kann dies durch die Anzeige von @ oder ! in der unteren Statuszeile erkennen. Da aber die Eingabe während der Übertragung nicht blockiert ist, führt jeder Tastendruck beim Laden eines Files zu einer Fehlermeldung und zum Abbruch. Da man aber meistens weiß, wann etwas geladen werden soll, ist die Abbruchmöglichkeit eher ein Vorteil. In eigenen Programmen kann durch spezielle Terminalbefehle die Übertragung sowohl vom KKF_FG als auch von Tastatur aus gesperrt werden.

Bei allen Übertragungen muß aber beachtet werden, daß im Terminalprogramm ein Spooler mit 16384 Zeichen verwendet wird. Bei WORDS ist die Liste schon längst übertragen, obwohl das Terminalprogramm weitere Zeichen anzeigt. Bei Überlauf des Spoolers werden einfach 16384 Zeichen vergessen.

Da das KKF_FG eine reine RAM-Version ist, erzeugt SAVESYSTEM Filename ein Bitimage, daß auch den KKF_FG-Kern beinhaltet. Dieses Image muß dann wie beim KKF_FG.COM in gerade und ungerade Adressen aufgespaltet und in zwei EPROM's gebrannt werden. Diese ersetzen dann den bis jetzt verwendeten KKF_FG-Kern.

Kapitel 2

Beschreibung

2.1 Allgemeines zum KKF_FG

Das KKF_FG ist ein vollständig im RAM der Bank 0 laufendes Programm. Dies bedeutet, daß sowohl Programm als auch die Daten im gleichen Segment untergebracht sind. Der RTX-2000/1A besitzt zwei Hardware-Stacks, die auch vom KKF_FG genutzt werden. Trotzdem wird in den entsprechenden Befehlen (**SP@** , **S0** , ...) so verfahren, als ob jeder Task seinen eigenen Stackspeicher im RAM besitzt. Dieser Speicher ist tatsächlich reserviert und kann in späteren Multitasking-Versionen zur Ablage der auf dem Hardwarestack befindlichen Werte genutzt werden.

Da das KKF_FG kein eigenes Betriebssystem hat und auch ohne das mitgelieferte Terminalprogramm bedient werden soll, wurde auf die Übergabe einer ersten Befehlszeile (wie z.B. beim KKF_PC) verzichtet.

Der Prozessor verfügt über einen externen Datenbus mit 16 Bit. Es müssen deshalb die Programme in Low- und Highbyte aufgespaltet und getrennt in ein EPROM gebrannt werden. Da bei 16Bit-Zugriffe auf ungerade Adressen die nächst tiefere Adresse verwendet und nur die Low-/Highbytes vertauscht werden, wurden alle 16Bit-Werte mit Anfang auf gerade Speicheradressen gesetzt. Es müssen deshalb die Align-Befehle immer verwendet werden. Da dies im KKF_FG-Kern automatisch gemacht wird, muß der Anwender nur bei eigenen Datenstrukturen darauf achten.

Der RTX-2000/1A verfügt über einen sogenannten G-Bus. Über diesen Bus können sowohl die internen Register als auch bis zu 8 externe Adressen erreicht werden. In Anlehnung an andere KKF-Versionen wird für das Anspechen des G-Bus statt den sonst üblichen **G@** und **G!** die Befehle **P@** und **P!** verwendet. Da dabei nur 16Bit-Zugriffe möglich sind, gibt es die Befehle **PC@** und **PC!** im KKF_FG nicht. Die externen Adressen beginnen mit dem Offset-Wert $18 und gehen bis zur seriellen Schnittstelle (Adresse $1F). Bei beiden Wörtern für den G-Bus muß die Adresse als Literal vorliegen, da es sich um spezielle +UCode-Befehle handelt.

Überhaupt weicht die interne Realisierung des KKF_FG erheblich von anderen KKF-Versionen ab, weil es sich um einen speziellen FORTH-Chip handelt:

- 16Bit-Zugriffe müssen bei geraden Adressen beginnen (Align notwendig)
- Viele Definitionstypen haben kein Codefeld
- Die Codefeldadressen werden vor Ablage im Dictionary durch 2 geteilt
- Viele Befehle werden als Inline-Code übernommen
- Kontrollbefehle beinhalten die Zieladresse des Sprunges im 16Bit-Code
- Literale werden als Inline-Code übernommen

Die Konzequenz dieser Struktur ist, daß es für das KKF_FG weder Assembler, Disassembler noch Debugger gibt. Statt dessen werden Files mit sogenannten UCode's, einen RTX-Dekompiler und ein optimierender Kompilier mitgeliefert.

2.2 Speicheraufteilung

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

$0000	SYSCON	Anfang des SYSCON-Bereiches
$0010	SYSVAR	Anfang des SYSVAR-Bereiches

$0060		Dictionary-Anfang
$3E90	HERE	Anfang des freien Programmbereiches
$EE2A	VDP @	Anfang des Variablenbereiches
$EE6C	HDP @	Anfang des Heap (beim Start leer)
$EE6C	TDP @	Anfang des Taskbereiches
$F1C6	TASK0	Anfang des USER-Bereiches im TASK0
$F51E	FIRST	Diskpuffer
$F920	SYSVAR $2C + @	Anfang des Puffers für 8 Befehlszeilen
$FBB0	SYSVAR $30 + @	Anfang des Anwender-Arbeitsbereich (leer)
$FBB0	(SYSVAR	Kopie des SYSVAR-Bereiches (für SAVE)
$FC00	LIMIT	Programmende+1
$FC00	INTBASE	Anfang der Interrupt-Tabelle

Am Ende des Speichers ist ein Bereich von 1KByte für die Interrupt-Tabelle reserviert. Diese Taballe ist mit Sprünge auf die entsprechende Fehlermeldung initialisiert.

2.2.1 Systemkonstanten

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

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

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

Die Anfangsadresse des Programmes ist durch die Startadresse des Prozessors nach dem Reset mit $0000 vorgegeben. Der im SYSCON-Bereich gespeicherte Wert wird deshalb ignoriert. Er dient, wie die Angabe der SYSVAR-Adresse, nur zur Information.

Die Endadresse+1 des RAM's ist bei der ausgelieferten Version mit $FC00 (entspricht vollem Speicher ohne Interrupt-Tabelle) angegeben. Sie kann jedoch verändert werden, um dann den restlichen Speicher für eigene Zwecke zu verwenden. Dabei ist aber darauf zu achten, daß noch genügend Speicher für das KK-FORTH und danach 1KByte für die Interrupt-Tabelle bleibt. Die Tabelle muß dabei immer an einer 1K-Grenze ($x000, $x400, $x800 oder $xC00) beginnen. Um mit der geänderten Speicheraufteilung zu Arbeiten, ist das Programm erst mit SAVESYSTEM Filename zu speichern und nach Erstellung eines EPROM's neu zu starten.

2.2.2 Systemvariablen

Beim Start des KKF_FG werden die Sytemvariablen (ab Adresse $0010) zuerst nach **(SYSVAR** kopiert und erst dann verwendet.

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

SYSVAR + 4 **Prozessorkennung**
$0023 System läuft auf RTX-2000 und RTX-2001A

SYSVAR + 6 **Hardwarekennung/Betriebssystem**
$0131 FG-Board (Boot=1:EPROM); RS232 über G-Bus ($1F-Bit0/1)

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

SYSVAR + 10 Attribut-Wort

```
%0001000010100111
 ^^                 RAM-Version
   ^^               Segmentierter Speicherverwaltung
     ^^             Ohne Floatingpoint-Unterstützung
       ^            Ein-/Ausgabe über Software-SIO
        ^           Fileinterface über Software-SIO
         ^^         Returnstack in Hardware
           ^^       Datenstack in Hardware
             ^      Kein Align beim 32Bit-Zugriff notwendig
              ^     Align beim 16Bit-Zugriff notwendig
               ^^   Reihenfolge der Daten: d3 d2 d1 d0
```

SYSVAR + 12 Zeiger auf Kopie der Systemvariablen
$FBB0 Kopie der Systemvariablen liegen am Speicherende

SYSVAR + 14 VOC-LINK
$1E02 Zeigt auf das als einziges vorhandene FORTH-Vokabular

SYSVAR + 16 DP
$3E90 Der KKF_FG-Kern hat eine Länge von 16016 Bytes

SYSVAR + 18 VDP
$0000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 20 VLEN
$0042 Im KKF_FG-Kern werden 66 Bytes für Variablen verwendet

SYSVAR + 22 HDP
$0000 Adresse wird beim Kaltstart ermittelt

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

SYSVAR + 26 TDP
$0000 Adresse wird beim Kaltstart ermittelt

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

SYSVAR + 30 TASK
$0000 Adresse wird beim Kaltstart ermittelt

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

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

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

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

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

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

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

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

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

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

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

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

SYSVAR + 56 **Enthält Backup der USER-Variable INPUT**

SYSVAR + 58 **Enthält Backup der USER-Variable OUTPUT**

SYSVAR + 60 **Enthält Backup der USER-Variable DISC**

SYSVAR + 62 **Reserviert für Baudraten-Angaben**
$0000 Gespeicherte Wert wird für eine Warteschleife verwendet

SYSVAR + 64 **Reserviert für UCode- und +UCode-Definitionen**
$0000 UCode wird hier eingetragen und dann der Befehl aufgerufen

SYSVAR + 66 **Reserviert für UCode- und +UCode-Definitionen**
$A020 Enthält Befehlscode für EXIT

SYSVAR + 68 **frei**

SYSVAR + 70 **frei**

SYSVAR + 72 **frei**

SYSVAR + 74 **frei**

SYSVAR + 76 **SFLAG**
Bit 0=1: Keine Start-Befehlszeile verfügbar (ist auf 1 gesetzt)
Bit 1=1: Keine Warnung durch **CREATE** ausgeben
Bit 2=1: Keine Zeilenpuffer verwenden
Bit 3=1: Die Baudrate wurde schon ermittelt
Bit 4-15: Werden im KKF_FG nicht verwendet

SYSVAR + 78 **UFLAG**
Frei für Anwender

Die serielle Schnittstelle wird durch Software realisiert. Die Anzahl der zusätzlich zu wartenden Takte wird in SYSVAR+&62 abgelegt. Bei der Umstellung der Ein-/Ausgabe auf **(OUTPUT** bzw. **(INPUT** wird Bit 3 von **SFLAG** untersucht. Ist es noch auf 0, so wird solange gewartet, bis ein Byte empfangen wurde. Auf Grund der Dauer des Stopbits (erstes Bit muß deshalb 1 sein) kann dann die Baudrate eingestellt werden.

Einige Befehle des RTX-2000/1A sind als sogenannte UCode- bzw. +UCode-Definition ausgelegt. Wenn sie nicht kompiliert, sondern direkt ausgeführt werden müssen, legt das KKF_FG den entsprechenden Code an Adresse $0050 und springt auf diese Adresse. Dies ist auch bei

Interrupt- oder Multitasking-Programmen möglich, da der vorher darin aufbewahrte Werte gespeichert wird.

Die 4 nicht verwendeten Wörter unterhalb von **SFLAG** können für eigene Zwecke verwendet werden. Sie haben den Vorteil, daß ihre Speicheradressen unveränderlich sind.

2.2.3 Der Taskbereich

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

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

$EE6C	WDP@	Anfang des Arbeitsbereich für diesen Task - Ablagebereich für WORD
	…	
		- Arbeitsbereich für <# bis #>
$EEC0	PAD	Anfang des Stringbereich - 256 Bytes für Strings
	…	
		- Platz für 256 Stackeinträge
$F1C0	S0 @	Ende des Datenstacks
	…	(danach 6 Bytes frei)
$F1C6	TASKADDR@	Anfangsadresse des USER-Bereiches - Programm/Daten für Taskwechsel
	…	
$F2C6	TASKADDR@ + $0100	Eingabepuffers für **QUERY** (= TIB - 2)
	…	
		- Platz für 256 Returnstackeinträge
$F518	R0 @	Ende des Returnstacks (danach 6 Bytes frei)

2.2.4 Diskpuffer und andere Arbeitsspeicher

Am Ende des verwendeten Speichers befindet sich der Diskpuffer, der Arbeitsspeicher für System und Anwender und die Interrupt-Tabelle.

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 Screennummer des nachfolgenden Blockes für 1024 Zeichen. Um bei Veränderung dieses Blockes 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 der Puffer leer ist. Solange kein File geöffnet ist, darf diese Screennummer für die Anforderung eines Arbeitsspeichers verwendet werden. Dabei muß man darauf achten, daß dieser Speicher nicht auf Diskette zurückgeschrieben werden kann und bei Filezugriff überschrieben wird.

Sytem-Arbeitsbereich

Wenn man das unveränderte KKF_FG 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 Position des Cursors ab der SWORK-Adresse abgelegt.

Anwender-Arbeitsbereich

Der vom KKF-Kern unterstützte aber bis jetzt noch nicht verwendete UWORK-Bereich 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 das Programm erst mit `SAVESYSTEM Filename` speichern und erneut starten. Der Inhalt dieses reservierten Speichers wird durch den KKF-Kern nicht mehr verändert.

Interrupt-Tabelle

Hinter das durch **LIMIT** gekennzeichnete Ende des KKF-Systems ist die Interrupt-Tabelle angelegt. Die Anfangsadresse kann deshalb mit **LIMIT** oder mit **INTBASE** ermittelt werden. Dabei handelt es sich um eine bei einer 1K-Grenze beginnenden Tabelle für 17 Einträge mit jeweils 32 Bytes. Alle Einträge werden beim Start des KKF_FG mit der Sequenz $1C48 $A020 $0000 ... initialisiert und das IBC-Register auf diese Tabelle gesetzt. Falls der Interrupt erlaubt wird, so erfolgt zuerst der Sprung in die Tabelle und dann zu einer Routine, die alle Interrupts abschaltet und eine Fehlermeldung ausgibt.

Wie das Beispielprogramm RTXTEST.SCR zeigt, kann durch wenige Befehle der gewünschte Interrupt aktiviert und auf eigene Programme umgeleitet werden. Eine besondere Bedeutung bekommt dabei der letzte Interrupt (Adresse $FE00). Dieser im Handbuch als "No Interrupt" gekennzeichnete Vektor wird manchmal bei aktiven Interrupts bei möglichen Stackunter-/überläufen ausgelöst. Es handelt sich anscheinend um Spikes auf internen Interrupt-Leitungen.

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. Eine Besonderheit des FORTH-Prozessors ist, daß viele Definitionen keine Codefeldadresse besitzen und das einige Definitionstypen die Daten als Inline-Werte behandeln. Falls der Befehlsname auf einer geraden Adresse endet, wird durch Einfügen einer Null für eine gerade CFA gesorgt.

Ein Befehl besteht aus:

LFA	Linkfeldadresse	Verkettung zur LFA des nächsten Befehls
NFA	Namensfeldadresse	Filename mit Längenangabe und 3 Flagbits
CFA	Codefeldadresse	Zeiger oder schon :-Definition
PFA	Parameterfeldadresse	Enthält Daten des Befehls

Die höchsten drei Bits in der Namensfeldadresse werden für die Eigenschaft des Befehls verwendet. Die unteren 5 Bits geben dann die Länge des folgenden Befehlsnamen an. Es können deshalb bis zu 31 Zeichen verwendet werden. Nur wenn das Bit 5 gesetzt ist, enthält die Codefeldadresse einen Zeiger. Ansonsten beginnt dort schon der Programmteil.

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

Bei den angegebenen Routinen (VAR bis (DIC handelt es sich um relativ kurze :-Definitionen im KKF_FG-Kern. Sie sind für die Tätigkeit des Befehls verantwortlich. Bei gelöschtem Bit 15 handelt es sich um einen Unterprogrammaufruf einer Adresse (Bit 0-14 ist dazu mit 2 zu Multiplizieren). Werten mit gesetzten Bit 15 sind Befehle des Prozessors. Oft werden die folgenden Sonderbefehle verwendet, wobei das ; anzeigt, daß nach Ausführung des Befehls das Wort beendet wird:

PC@;	Nachfolgenden Adresse zum Stack
LIT;	Inhalt der nachfolgenden Adresse zum Stack

:-Definitionen am Beispiel :

```
0000:25F8  25 DE 01 3A 02 1B
```

$25F8/9	$25DE	Verkettung zum vorherigen Befehl
$25FA	$01	Namensfeldadresse (Länge: 1 Zeichen)
$25FB	$3A	Befehlsname :
$25FC/D	$021B	Highlevel-Teil (beginnt mit CURRENT)

Konstanten am Beispiel #B/BLK

```
0000:0254  02 4A 06 23 42 2F 42 4C 4B 00 DE 20 04 00
```

$0254/5	$024A	Verkettung zum vorherigen Befehl
$0256	$06	Namensfeldadresse (Länge: 6 Zeichen)
$0257 ...	$23 ...	Befehlsname #B/BLK
$025D	$00	Frei wegen Align auf gerade Adressen
$025E/F	$DE20	Befehl: LIT;
$0260/1	$0400	Wert der Konstanten

32Bit-Konstanten am Beispiel PI

```
$3.1415 2Constant pi

0000:3E90  39 BA 02 50 49 00 3E 00 03 14 15
```

$3E90/1	$3DC1	Verkettung zum vorherigen Befehl
$3E92	$02	Namensfeldadresse (Länge: 2 Zeichen)
$3E93/4	$50 $49	Befehlsname PI
$3E95	$00	Frei wegen Align auf gerade Adressen
$3E96/7	$003E	Adresse/2 von (2CON
$3E98..B	$0003.1415	Wert der 32Bit-Konstanten

Variablen am Beispiel FILE-ID

```
0000:0516  05 0A 07 46 49 4C 45 2D 49 44 00 38 00 00 00
```

$0516/7	$050A	Verkettung zum vorherigen Befehl
$0518	$07	Namensfeldadresse (Länge: 7 Zeichen)
$0519 ...	$46 ...	Befehlsname FILE-ID
$0520/1	$0038	Adresse/2 von (VAR
$0522/3	$0000	Offset in den Variablenbereich

32Bit-Variablen am Beispiel COUNTER

```
2Variable counter

0000:3E9C  3E 90 07 43 4F 55 4E 54 45 52 00 38 00 42
```

$3E9C/D	$3E90	Verkettung zum vorherigen Befehl
$3E9E	$07	Namensfeldadresse (Länge: 7 Zeichen)
$3E9F ...	$43 ...	Befehlsname COUNTER
$3EA6/7	$0038	Adresse/2 von (2VAR (gleich mit (VAR)
$3EA8/9	$0042	Offset in den Variablenbereich

USER-Variablen am Beispiel >IN

```
0000:0458  04 4E 03 3E 49 4E 00 63 00 3A
```

$0458/9	$044E	Verkettung zum vorherigen Befehl
$045A	$03	Namensfeldadresse (Länge: 3 Zeichen)
$045B ...	$3E ...	Befehlsname >IN
$045E/F	$0063	Adresse/2 von (USER
$0460/1	$003A	Offset in den USER-Bereich

Vokabulare am Beispiel FORTH

```
0000:1DF8  1D DE 05 46 4F 52 54 48 00 69 00 00 00 22
```

$1DF8/9	$1DDE	Verkettung zum vorherigen Befehl
$1DFA	$05	Namensfeldadresse (Länge: 5 Zeichen)
$1DFB ...	$46 ...	Befehlsname FORTH
$1E00/1	$0069	Adresse/2 von (VOC
$1E02/3	$0000	Verkettung zum nächsten Vokabular (fehlt)
$1E04/5	$0022	Offset in den Variablenbereich
(VDP @ $22 + ergibt $EE4C)		
$EE4C/D	$39BA	Zeiger zur BOOT-Linkfeldadresse

DEFER-Definitionen am Beispiel ERRORTEXT@

```
0000:3648  36 36 0A 45 52 52 4F 52 54 45 58 54 40 00 00 42 00 36
```

$3648/9	$3636	Verkettung zum vorherigen Befehl
$364A/B	$0A	Namensfeldadresse (Länge: 10 Zeichen)
$364C ...	$45 ...	Befehlsname ERRORTEXT@
$3655	$00	Frei wegen Align auf gerade Adressen
$3656/7	$0042	Adresse/2 von (DEFER
$3658/9	$0036	Offset in den Variablenbereich
(VDP @ $32 + ergibt $EE60)		
$EE60/1	$1B2D	Zeigt auf die versteckte Routine ($365A/2)

Die versteckte Routine zu **ERRORTEXT@** beginnt unmittelbar hinter dem Befehlsheader. Die im Variablenbereich abgelegte Adresse ist eine kompilierte Codefeldadresse und deshalb durch 2 geteilt.

LABEL-Definitionen am Beispiel HEREADDR

```
label hereaddr

0000:EE5E  39 BA A8 48 45 52 45 41 44 44 52 00 EE 5A
0000:EE5A  00 6D 3E 90
```

$EE5E/F	$39BA	Verkettung zum vorherigen Befehl
$EE60	$A8	Namensfeldadresse (Länge: 8 Zeichen) (Befehl ist Immediate und Indirect)
$EE61 ...	$48	Befehlsname HEREADDR
$EE69	$00	Frei wegen Align auf gerade Adressen
$EE6A/B	$EE5A	Zeiger auf Codefeldadresse (im Heap)

$EE5A/B	$006D	Adresse/2 von (LABEL
$EE5C/D	$3E90	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 WAHR

```
' true  Alias  wahr

0000:3E90  EE 5E 24 57 41 48 52 00 01 EE
```

$3E90/1	$EE5E	Verkettung zum vorherigen Befehl
$3E92	$24	Namensfeldadresse (Länge: 4 Zeichen) (Befehl ist Indirect)
$3E93 ...	$57	Befehlsname WAHR
$3E97	$00	Frei wegen Align auf gerade Adressen
$3E98/9	$01EE	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

0000:3E9A  3E 90 0B 33 44 2D 43 4F 4E 53 54 41 4E 54 12 63
0000:3EAA  04 6D 09 0E AE 80 09 0E 09 0E 13 1A BE 01 A0 C0
0000:3DBA  EE 00 AE 80 B8 C2 A0 C0 EE 00 AE 80 B8 C2 EE 00 ...
```

$3E9A/B	$3E90	Verkettung zum vorherigen Befehl
$3E9C	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$3E9D ...	$33	Befehlsname 3D-CONSTANT
$3EA8/9 ...	$1263	Create-Teil (endet mit (;code)
$3EB6/7	$BE01	UCODE-Befehl R>
$3EB8 ...	$A0C0	Does>-Teil (endet mit exit)

```
0000:3ECC  3D 9A 03 31 32 33 1F 5B 00 01 00 02 00 03
```

$3ECC/D	$3D9A	Verkettung zum vorherigen Befehl
$3ECE	$03	Namensfeldadresse (Länge: 3 Zeichen)
$3ECF ...	$31	Befehlsname 123
$3ED2/3	$1F5B	Aufruf des Does>-Teil von 3D-CONSTANT
$3ED4/5	$0001	x-Wert
$3ED6/7	$0002	y-Wert
$3ED8/9	$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

0000:3EDA  3E CC 0B 33 44 2D 56 41 52 49 41 42 4C 45 12 C4
0000:3EEA  BE 40 09 32 BE 40 09 32 BE 40 09 32 13 1A 00 5B
0000:3EFA  AE 80 A0 02 A8 40 A0 20
```

$3EDA/B	$3ECC	Verkettung zum vorherigen Befehl
$3EDC	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$3EDD ...	$33	Befehlsname 3D-VARIABLE
$3EE8/9 ...	$12C4	VCreate-Teil (endet mit (;code)
$3EF8/9	$005B	(vdoes> aufrufen
$3EFA ...	$AE80	VDoes>-Teil (endet mit exit)

```
0000:3F02  3E DA 07 56 45 43 54 4F 52 31 1F 7C 00 42
```

$3F02/3	$3EDA	Verkettung zum vorherigen Befehl
$3F04	$07	Namensfeldadresse (Länge: 7 Zeichen)
$3F05 ...	$56	Befehlsname VECTOR1
$3F0C/D	$1F7C	Aufruf des VDoes>-Teil von 3D-VARIABLE
$3F0E/F	$0042	Offset in den Variablenbereich

UCODE-Definition am Beispiel DROP

```
0000:09B0  09 9E 84 44 52 4F 50 00 00 74 AE 40
```

$09B0/1	$099E	Verkettung zum vorherigen Befehl
$09B2	$84	Namensfeldadresse (Länge: 4 Zeichen) (Befehl ist Immediate)
$09B3 ...	$44 ...	Befehlsname DROP
$09B7	$00	Frei wegen Align auf gerade Adresse
$09B8/9	$0074	Adresse/2 von (UCODE
$09BA/B	$AE40	Befehlscode für DROP

+UCODE-Definition am Beispiel P@

```
0000:116E  11 5A 82 50 40 00 00 85 BE 00
```

$116E/F	$115A	Verkettung zum vorherigen Befehl
$1170	$84	Namensfeldadresse (Länge: 4 Zeichen) (Befehl ist Immediate)
$1171/2	$50 $40	Befehlsname P@
$1173	$00	Frei wegen Align auf gerade Adresse
$1174/5	$0085	Adresse/2 von (+UCODE
$1176/7	$BE00	Befehlscode für P@

2.4 Informationen zu KKF_FG-Befehle

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

2.4.1 32Bit-Adressen

Der RTX-2000/1A hat 16Bit-Register und kann gleichzeitig mit einer Programm- und einer Datenbank zu je 64KByte arbeiten. Im KKF_FG wird nur die mit 64KByte bestückte Bank 0 sowohl für Programm als auch für Daten genutzt. Darüber hinaus können aber noch 15 weitere 64K-Bänke angesprochen werden. Solange nur mit den 64KBytes des FORTH-Segmentes gearbeitet wird, ist dies nicht ersichtlich. Nur bei dem Befehl **DUMP** wird auch die Banknummer angezeigt.

In der Befehlsliste sind viele Befehle zu finden, die als Parameter eine 32Bit-Adresse (genannt ptr) erwarten. Beim KKF_FG ist dabei immer die kombinierte Angabe von Banknummer und Adresse

gemeint. Dabei wird zuerst die Banknummer und dann die Adresse angegeben. Dies hat den Vorteil, daß die Adresse schnell ändern werden kann.

(ptr --) entspricht (bank addr --)

Befehle wie **L2@** oder **L2!** können zur direkten Adressierung eines bestimmten Speichers verwendet werden. Dabei ist aber zu beachten, daß wie bei **2@** und **2!** zuerst der oberste Stackeintrag (also der höherwertige Teil) und dann erst der niederwertige 16Bit-Wert abgelegt wird.

Man kann die gleichen Befehle aber auch zu einer echten 32Bit-Adressierung heranziehen. Dabei müssen aber immer die Befehle **D>PTR** und **PTR>D** zur Umrechnung zwischen 32Bit-Adresse und Pointerangabe herangezogen werden. Beim KKF_FG handelt es sich bei beiden Befehle um ALIAS-Definitionen von **SWAP** .

Bei allen Befehlen mit ptr-Angabe ist darauf zu achten, daß die Banknummer nicht geändert wird. Nach der Adresse bank:$FFFF wird die entsprechende Aktion (kopieren, lesen ...) bei bank:$0000 fortgesetzt. Bei Verschiebungen müssen deshalb die Bank-Grenzen beachtet werden.

2.4.2 Adreß-Align

Um das KKF_FG nicht unnötig langsam zu machen, wurde auf eine Sonderbehandlung bei Zugriff auf 16Bit-Werte an ungeraden Adressen verzichtet. Die vom Prozessor als Befehl zur Verfügung gestellten **@** und **!** verwenden bei ungeraden Anfangsadressen statt der nächsten Speicherzelle die vorhergehende Adresse und vertauschen zusätzlich noch High- und Lowbyte. Da dies zu falschen Ergebnisse führt, wird mit Hilfe der Align-Befehle für eine gerade Adresse bei allen 16Bit-Werten sowohl im Dictionary als auch in allen anderen Speichern wie Variablen-, Heap- und Taskbereich gesorgt.

Solange keine gemischten Datenfelder erzeugt oder Inline-Werte in eine Definition übernommen werden, muß sich der Anwender nicht darum kümmern. Das KKF_FG sorgt automatisch bei neuen Befehlen und bei Variablen für eine gerade Anfangsadresse. Bei Inline-Strings werden, falls nötig, eine zusätzliche 0 angehängt.

2.4.3 Länge der Datentypen

Der RTX-2000/1A 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. Bei Inline-Strings oder in Tabellen muß dafür gesorgt werden, daß der nächste String an geraden Adressen beginnt. Dies wird auch durch die Befehle **$,** und **/$** unterstützt.

16Bit-Worte haben eine Länge von zwei Adressen. Dabei dürfen nur gerade Zieladressen als Anfangswert angegeben werden. Es wird anders als beim KKF_PC immer das höherwertige 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 24-31 (höchstwertigste Byte)
addr+1	Bit 16-23
addr+2	Bit 8-15
addr+3	Bit 0-7 (niederwertigste Byte)

2.4.4 Portbefehle

Die RTX-Prozessoren verfügen über einen sogenannten G-Bus. Diese im KKF_FG wie Ports zu behandelnden Adresse dienen zum Zugriff auf interne Register (Adressen $00 bis $17) oder zum Ansprechen des externen G-Bus (Adresse $18 bis $1F). An Stelle der bei anderen RTX-FORTH-Versionen definierten **G@** und **G!** werden die im KKF üblichen Befehle **P@** und **P!** verwendet. Dabei handelt es sich immer um 16Bit-Zugriffe.

Interne G-Bus-Adressen

Adresse	Lesen	Schreiben
$00	R@	RDROP >R
$01	R>	>R
$02	R@ 2*	OF(
$03	CR-Register (Konfiguration)	
$04	MD-Register (für Multiplikation/Division)	
$05	Pseudoregister	
$06	SR-Register (für Quadratwurzel)	
$07	PC-Register (Programmzeiger)	
$08	IM-Register (Interrupt-Maske: 1=deaktiviert)	
$09	SP-Register (beide Stackpointer)	
$0A	SU-Register (nur bei RTX-2001A: Stack-Untergrenze)	
$0B	IV-Register (Stack-Obergrenze; Interruptvektor)	
$0C	IP-Register (Bit 16-19 des Returnstacks)	
$0D	DP-Register (Banknummer für Datenzugriff)	
$0E	UP-Register (Banknummer für USER-Zugriff)	
$0F	CP-Register (Banknummer für Code-Zugriff)	
$10	IBC-Register (zusätzliche Konfigurationen)	
$11	UB-Register (Anfangsadresse des USER-Bereiches)	
$12	reserviert	
$13	Counter/Timer 0	
$14	Counter/Timer 1	
$15	Counter/Timer 2	
$16	ML- (RTX-2000) oder RX-Register (RTX-2001A)	
$17	MH- (RTX-2000) oder RH-Register (RTX-2001A)	

Beim KKF_FG werden alle Zugriffe auf den Codebereich des Prozessors durchgeführt. Das DP-Register wird nur im Zusammenhang mit den externen Speicherzugriffen verwendet.

Die hier mit UBR und UPR bezeichneten Register zeigen nicht auf den USER-Bereich des KK-FORTH, sondern auf einen für eigene Zwecke oder sogar für Hardware verwendbare Speicheradresse. Da diese Register im KKF_FG nicht belegt sind, können sie zusammen mit den speziellen USER-Codes für eigene Zwecke eingesetzt werden.

Der RTX-2000 verfügt über einen Hardware-Multiplikator. Dieser wird über spezielle Befehle gestartet, die nichts anderes als Zugriffe auf ML- und MH-Register sind. Beim RTX-2001A stehen statt dessen zwei Register zur Verfügung, wobei RX als automatisches Zählregister verwendet werden kann. Da weder der Hardware-Multiplikator noch die Register RX bzw. RH im KKF_FG genutzt werden, stehen sie zur freien Verfügung.

Externe G-Bus-Adressen

$18 ... $1E	frei für eigene Hardware
$1F	Serielle Schnittstelle Bit 0: Handshake lesen oder Bit schreiben Bit 1: Bit lesen oder Handshake schreiben

Die G-Bus-Adresse $1F ist im FG- und dem mc-Risc-Board für die serielle Schnittstelle vorbereitet. Dabei werden die Bits 0 und 1 für Daten und für Handshake-Leitungen verwendet.

2.4.5 Filebefehle

Alle Filebefehle des KKF_FG sind direkte Aufrufe der entsprechenden (MS)DOS-Befehle über das mitgelieferte Terminalprogramm. 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 Banknummer nicht verändert wird.

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

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

2.4.6 Fehlermeldungen

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

2.4.7 Kontrollstrukturen

Anders als bei anderen KKF-Versionen stehen die vom FORTH benötigten Grundbefehle für Kontrollstrukten als Inline-Codes zur Vefügung. Da im Code auch die Zieladresse des Sprungs untergebracht sind, werden diese Befehle in einem Takt ausgeführt. Leider ist aber dazu eine andere Befehlsstruktur notwendig, die dazu führt, daß **BRANCH** , **?BRANCH** und **NEXTBRANCH** als Immediate-Befehle definiert wurden.

Wenn sich der Programmierer aber an die schon im Handbuch erwähnten Hinweise hält und die drei Befehle immer mit **POSTPONE** kompiliert und zur Adreßberechnung **>MARK** , **>RESOLVE** , **<MARK** und **<RESOLVE** verwendet, dann sind eigene Kontrollstrukturen auch im KKF_FG lauffähig.

2.4.8 Kompilieren der Codefeldadresse

Der RTX-2000/1A markiert durch das gelöschte Bit 15 in einem auszuführenden Befehl, daß es sich nicht um einen "Opcode", sondern um eine auszuführende Routine handelt. Es wird dann zur Ausführung des Wortes die Rücksprungadresse auf den Returnstack gebracht und das gewünschte Wort aufgerufen. Die Adresse des Wortes muß dazu vorher mit 2 multipliziert werden.

Alle Befehle, die eine Codefeldadresse kompilieren müssen, teilen vor der Übernahme der Adresse in das Dictionary die Adresse durch zwei. Im KKF_FG sorgt das dafür vorgesehene für die Anpassung. Bei direkter Manipulation von Codefeldadressen im Dictionary oder in DEFER-Definitionen muß ebenfalls die Halbierung der Adresse beachtet werden (z.B. **PERFORM**).

Innerhalb einer :-Definition stehen aber nicht nur Aufrufe anderer FORTH-Worte, sondern auch die sogenannten UCode's. Dabei handelt es sich um 16Bit-Werte, die direkte Anweisungen an den Prozessor darstellen. Da es dabei auch Befehle gibt, die nach dem eigentlichen Code noch einen

weiteren 16Bit-Wert im Programm ablegen, muß man dem optimierenden Kompiler durch **,C** angeben, an welcher Adresse der letzte Befehl abgelegt wurde. **,C** speichert dabei wie **,A** die aktuelle Dictionaryadresse in **CODE?** , legt aber den angegebenen Wert unverändert ab.

2.4.9 Daten- und Returnstackbefehle

Obwohl beide Stacks direkt in Hardware realisiert sind, wird bei der Verwaltung immer der vorgesehene Speicher im Taskbereich angegeben.

Unterschiede zwischen RTX-2000 und RTX-2001A

Anscheinend gab es bei der Implementation der Stacks im RTX-2000 große Probleme. Deshalb wurde die Größe der Stacks von vorher 256 auf 64 Einträge reduziert und zusätzlich der Stackkontroller vollständig überarbeitet.

Die Funktionen der beiden Kontroller sind dabei so unterschiedlich, daß das KKF_FG beim Start den entsprechenden Prozessortyp erkennt und im Flag **RTX2001A?** speichern muß. Danach wird gemäß dieses Flags unterschiedliche Routinen in **PICK** , **SP@** , **SP!** , **RP@** und **RP!** ausgeführt.

Obwohl es dem RTX-2000/1A egal ist, wo die Stackzeiger stehen, ist das KKF_FG so eingerichtet, daß der Stackanfang mit dem physikalischen Anfang zusammenfallen. Dies kann man bei Aktivierung des Stackunterlauf-Interrupts auch erkennen. Da aber im KKF_FG der Interrupt deaktivert ist, wird einfach die Stacktiefe als negativ angegeben, wenn der Zeiger auf einen der letzten 16 Einträge steht. Dies führt dazu, daß für den RTX-2000 noch 240 und für den RTX-2001A noch 48 Werte auf dem Daten- und Returnstack erlaubt sind.

Datenstackbefehle

Die Endadresse des Datenstacks wird in **S0** aufbewahrt. **SP@** liefert wie beim KKF_PC eine Adresse, die unterhalb oder auf diesem Wert liegt. Da für jeden 16Bit-Wert zwei Adressen belegt werden, ermittelt sich die Datenstacktiefe nach folgender Formel:

```
Stacktiefe:     sp@  s0 @  swap -  2/
```

Bei der Übergabe einer neuen Stacktiefe mit **SP!** ist ebenfalls eine Speicheradresse anzugeben. Sie sollte möglichst vorher durch **SP@** geholt worden sein oder in **S0** stehen.

Returnstackbefehle

Die Endadresse des Returnstacks wird in **R0** aufbewahrt. **RP@** liefert wie beim KKF_PC eine Adresse, die unterhalb oder auf diesem Wert liegt. Da für jeden 16Bit-Wert zwei Adressen belegt werden, ermittelt sich die Datenstacktiefe nach folgender Formel:

```
Returntiefe:    rp@  r0 @  swap -  2/
```

Bei der Übergabe einer neuen Returnstacktiefe mit **RP!** ist ebenfalls eine Speicheradresse anzugeben. Sie sollte möglichst vorher durch **RP@** geholt worden sein oder in **R0** stehen.

Da wie in anderen KKF-Versionen ein Returnstack-Unterlauf von bis zu 3 Werten abgefangen werden soll, sind die ersten drei Einträge mit einer Fehlerroutine belegt. Zusammen mit der Rückkehradresse zu **INTERPRET** und **QUIT** sind beim Aufruf eines Wortes fünf Returnstackeinträge belegt.

2.5 Zusatzhinweise

Ein Blick hinter die "Kulissen" des KKF-Systems soll in dem vorletzten Kapitel der Beschreibung des KKF_FG-Kerns die Initialisierungssequenz und die Ein-/Ausgabe verdeutlichen. Die Realisierung der Schnittstellenbefehle des KKF_FG ist als Listing (Files P7_TERM.SCR) im Anhang E angegeben.

2.5.1 Initialisierung des Systems

Beim Start des KKF_FG wird zuerst der Interrupt deaktiviert und das EPROM aktiviert. Anschließend wird der Dictionarybereich von Adresse 0 bis **HERE** unverändert ins RAM kopiert. Die danach folgenden Daten des EPROM's wie der Variablen- oder USER-Bereich werden dann an die richtigen RAM-Adressen gebracht und die entsprechenden Zeiger korrigiert. Der Speicher zwischen HERE und VDP @ wird also beim Start des KKF_FG nicht verändert. Nach Umschalten auf das RAM wird dann noch die Interrupt-Tabelle und die die Ein-/Ausgabeschnittstellen initialisiert und danach der FORTH-Interpreter/Kompiler aufgerufen.

2.5.2 Ein-/Ausgabe und das Fileinterface

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

Bei der hier verwendeten, rein softwaremäßig realisierten RS232-Schnittstelle wird die Baudrate am ersten emfangenen Byte erkannt. Es werden immer ein Startbit, 8 Datenbits und ein Stopbit erwartet und beim Senden noch ein zweites Stopbit ausgegeben.

Initialisierung der Schnittstelle

Beim jedem Aufruf von **(OUTPUT** oder **(INPUT** wird an Hand von Bit 3 in **SFLAG** untersucht, ob die Schnittstelle schon initialisiert wurde. Ist dies nicht der Fall, so werden die Anzahl der Wartetakte pro Halbbit am ersten empfangenen Stopbit ermittelt. Dieser Wert wird dann in den SYSVAR-Bereich geschrieben und das SFLAG-Bit gesetzt.

Wenn vor der Abspeicherung des Programm mit `SAVESYSTEM Filename` das Bit 3 in dem SFLAG-Wert des (SYSVAR-Bereiches gesetzt wird (Befehlsfolge: `sflag sysvar - (sysvar + dup @ $08 or swap !`) entfällt beim Programmstart die Initialisierung und das KKF_FG meldet sich mit der zuletzt verwendeten Baudrate.

Zeichen von Tastatur holen

Wenn noch kein Wert im internen Puffer vorhanden ist, wird zum Empfang eines Zeichens durch **KEY** die Handshake-Leitung deaktiviert und so lange gewartet, bis ein Zeichen kommt. Beim Empfang des Startbits wird sofort die Handshakeleitung wieder aktiviert und damit sichergestellt, daß keine weiteren Bytes übertragen werden.

Für **KEY?** müßte ein Status abfragbar sein. Dies ist aber bei anderen Terminalprogrammen nicht möglich. Statt dessen verwendet das KKF_FG den internen Puffer für ein Zeichen. Ist dort ein Zeichen gespeichert, so liefert **KEY?** sofort das Flag -1 zurück. Ansonsten wird ähnlich wie bei **KEY** die Handshakeleitung für ca. 1 ms deaktiviert. Während und bis ca. 1 ms nach der Handshake-Abschaltung wird ein Zeichen erwartet und in den Puffer übertragen. Wenn innerhalb dieser 2 Millisekunden immer noch kein Zeichen ankommt, so liefert **KEY?** das Flag 0 zurück. Weil durch diese Methode die Tastaturabfrage sehr langsam wird und deshalb kaum in schnellen Schleifen verwendet werden kann, sollte bei Bedarf einer externen Abbruchmöglichkeit eher auf die Interrupts zurückgegriffen werden.

Zeichen ausgeben

Die Ausgabe ist ebenfalls in Software realisiert. Dabei kann mit **KEY?** der Status der Handshake-Leitung direkt abgefragt werden. Erst wenn der Handshake vom Terminal freigegeben ist, wird zuerst das Startbit, dann die 8 Datenbits und zum Schluß noch zwei Stopbits übertragen.

Filebefehle

Wie schon in Kapitel 2.4.5 angesprochen, sind alle Filewörter im KKF_FG durch COMMAND!-Befehle und im Terminalprogramm 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.

Bei der Übertragung wird das ESC-Zeichen als Anfang einer Befehlsübergabe verwendet und dabei auch Fehlernummern vom Terminalprogramm erwartet. Jeder dabei übergebene Tastencode führt zu einem Fehler und sollte deshalb vermieden werden. Falls ein Menüprogramm realisiert werden soll, so kann man die Möglichkeit zum Sperren der Tastatur nutzen. Dies ist im Handbuch beim Kapitel über das Terminalprogramm beschrieben.

2.6 Informationen zum RTX-2000/1A

Beim RTX-2000 bzw. RTX-2001A handelt es sich um Weiterentwicklungen des vom Charles Moore konzipierten FORTH-Prozessors NC4000.

Die Prozessoren kommen ohne Microcodes aus und können deshalb bis auf externe Speicherzugriffe oder 16Bit-Literals alle Befehle in einem Takt ausführen. Durch seine dem FORTH angepaßte Struktur ist es sogar möglich, bis zu 5 FORTH-Befehle in einem 16Bit-Befehlscode unterzubringen.

Um was für einen Befehlscode es sich handelt, erkennt der RTX aus dem Bitmuster. Dabei sind Werte mit gelöschten Bit 15 immer Unterprogrammaufrufe. Die Speicheradresse des Befehls wird dann aus Bit 0-14 durch Multiplikation mit 2 ermittelt. Bei gesetzten Bit 15 muß die Art des Befehls aus den anderen Bits ermittelt werden. Bis auf die Spungbefehle kann dabei im Bit 6 des Codes durch das Setzen des sogenannten EXIT-Bits ein Rücksprung zum aufrufenden Wort erzwungen werden.

Bit 15=0 Unterprogrammaufruf (Adresse: 2 * Bit0..14)
Bit 15=1 Befehl

```
%1000 0bba aaaa aaaa        ?DUP BRANCH
%1000 1bba aaaa aaaa        ?BRANCH
%1001 0bba aaaa aaaa        BRANCH
%1001 1bba aaaa aaaa        NEXTBRANCH

%1010 xxxx xx;x xxxx        Arithmetik-Befehle

%1011 xxxx x0;x xxxx        G-Bus-Befehle
%1011 xxxx x1;x xxxx        Kurzliterals

%1100 xxxx x0;x xxxx        Befehle für den USER-Bereich
%1100 xxxx x1;x xxxx        reserviert

%1101 xxxx x0;x xxxx        16Bit-Literals
%1101 xxxx x1;x xxxx        reserviert

%1110 xxxx xx;x xxxx        16Bit-Speicherzugriffe
%1111 xxxx xx;x xxxx        8Bit-Speicherzugriffe
```

Ausführliche Beschreibungen zu den einzelnen Befehlsgruppen sind im "RTX Programmer's Referenz"-Handbuch zu finden. Die wichtigsten Angaben sind auch schon in den Datenblättern zum RTX-2000 bzw. RTX-2001A erwähnt. Besonders gelungen finde ich das leider nicht einzeln erhältliche Handbuch, daß mit der RTX2001A-Platine für den RTX-Wettbewerb ausgeliefert wurde.

Kapitel 3

Zusatzprogramme

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

3.1 RTXDIS.SCR

Der Disassembler oder richtiger gesagt der Dekompiler ist eine Anpassung des RTX-Dekompilers aus dem FG-FORTH und stammt vom Dirk Brühl. Im FORTH-Vokabular stehen nach dem Laden und **HCLEAR** folgende Befehle zur Verfügung:

disassemblieren	(addr --)	Ein paar Zeilen oder bis ; disassemblieren
(disass	(addr --)	Wie DISASSEMBLIEREN
disass	(name ; --)	Befehl Name disassemblieren
dis	(name ; --)	Bis Tastenabbruch disassemblieren
ndis	(addr n --)	n EXIT's ab addr disassemblieren

Der Disassembler generiert immer 10 Zeilen oder geht bis zum nächsten EXIT-Code und wartet dann auf eine Bestätigung. Mit CTRL+X kann die Ausgabe abgebrochen werden.

3.2 UCODES.SCR

Der RTX kennt viele FORTH-Befehl als 16Bit-Code. Das File UCODES.SCR ist die Grundlage des KKF_FG und enthält deshalb alle im Kern verwendeten Codes. Um die nicht als eigene Befehle zur Verfügung stehenden Wörter in Applikationen zur Verfügung zu haben, kann dieses File nachgeladen werden. Die Befehle teilen sich in mehrere Befehlsgruppen auf.

Kombination mehrerer FORTH-Befehle

Um auch ohne optimierenden Kompiler mehrere FORTH-Befehle in einem Takt auszuführen, können die speziellen Befehle **RDROP>R** bis **2DUP!2+** verwendet werden.

Spezielle Befehle

Im Prozessor gibt es interne Register und Zusatzbits wie z.B. das Carry-Flag. Darüber hinaus werden oft durch spezielle G-Bus-Zugriffe bestimmte Aktionen ausgelöst.

Die Befehle **CLC** , **+C** und **-C** dienen zur Manipulation des Carry-Bits bei 32Bit-Befehle. Es muß aber dabei beachtet werden, daß jede Addition und Subtraktion das Carry-Flag verändert.

Nach einem Arithmetik-Befehl kann das Carry-Flag auch als Bit 16 bei Schiebeoperationen verwendet werden. Dies wird in den Befehlen **&U2/** bis **-&U2/** durch das Zeichen & verdeutlicht.

Mit sogenannten Math-Step's können 16Bit-Multiplikationen, 32/16Bit-Divisionen, Quadratwurzelberechnungen, CRC-Prüfsummenermittlungen und Bitinversionen in wesentlich wenigeren Taktzyklen als bei diskreter Berechnung ermittelt werden.

Registerdefinitionen

Wie schon oben erwähnt, haben die internen Register auch Namen. Sowohl für das Lesen als auch für das Schreiben wurde bei jedem Register ein eigener Befehl definiert.

Speicherzugriffe

Gleichzeitig mit dem externen Speicherzugriff kann die Adresse erhöht oder erniedrigt werden. Die Befehle **C@+1** bis **2DUP!2+** verwenden diese Möglichkeit, um die Adresse zu erhalten und zu erhöhen.

Im KKF_FG wird bis auf die Pointerbefehle immer nur mit der Codepage gearbeitet. Wenn in eigenen Programmen auf eine andere Datenbank zugegriffen werden soll, so kann dies durch Veränderung der Zielbank in DPR und Umschaltung des verwendeten Registers durch **SELDPR** bzw. **SELCPR** angegeben werden.

Der Befehl OF(

Eine weitere Besonderheit des RTX ist die Möglichkeit der Befehlswiederholung. Dazu wird die Anzahl-1 in ein spezielles Register geschrieben und dadurch der nächste Befehl bis zu 65536 mal wiederholt. Da dabei aber nur die sogenannten UCODE-Definitionen verwendet werden können und Befehlsoptimierungen eher zu Fehler führen, sorgt der Befehl **(OF** für eine Überprüfung des nachfolgenden Befehls und für die Abschaltung des optimierenden Kompiliers. **OF(** ist besonders bei Speicherbefehle sinnvoll, weil nach dem Holen des Befehlscodes immer nur ein Takt pro Speicherzugriff erforderlich ist.

Der Hardware-Multiplikator

Ein nur auf dem RTX-2000 lauffähiges Beispiel ist die Verwendung des Hardware-Multiplikators. Wird statt der normalweise verwendeten Befehle diese neuen Definitionen aufgerufen, kann eine Beschleunigung der Berechnung um Faktor 4-5 erreicht werden. Hier wird auch zum ersten Mal das Flag **RTX2001A?** außerhalb der Stackbefehle herangezogen.

3.3 OPTICOMP.SCR

Hinter diesem File versteckt sich der oOptimierende Kompiler, welcher nach dem Laden folgende Befehle zur Verfügung stellt:

OPTIMIZE?	(-- addr)	Flag, ob Optimizer aktiv ist
OPTIMIZE	(--)	Aktivieren des optimierenden Kompilers
NONOPTIMIZE	(--)	Abschalten des optimierenden Kompilers

Durch **OPTIMIZE** wird der Befehl **,C** auf eine eigene Routine umgeleitet. Diese Routine untersucht dann, ob der angegebene Code zusammen mit den letzten Befehlen ausgeführt werden kann. Ist dies nicht der Fall, so wird der Code unverändert abgelegt, aber die Adresse in **CODE?** gemerkt. Im anderen Fall wird der geeignete Code ermittelt und dazu oft die vorher abgelegten Werte übernommen.

Da es manchmal notwendig ist, mehrere Befehle zu kombinieren, besitzt dieser Kompiler noch eine Variable namens **CODE2?** , die bei Ablage eines Wertes im Dictionary den alten Wert von **CODE?** übernimmt. Falls es einmal nötig ist, kann die Optimierung auch für die Dauer einiger Befehle durch Löschen von **OPTIMIZE?** deaktiviert werden.

Auf eine ausführliche Beschreibung des optimierenden Kompilier möchte ich hier verzichten. Das Listing ist gut strukturiert und mit Kommentar versehen. Wer mehr daraus machen will, kann es als Grundlage verwenden. Ansonsten muß man sagen, daß dieses noch nicht sehr intensiv

getestete, aber schon zur Erstellung des KKF_FG herangezogene Programm mit Vorsicht zu genießen ist und noch bei weitem nicht alle Möglichkeiten des Prozessors ausnützt.

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 RTXTEST.SCR

Dieses File enthält ein kleines Bespiel für Interrupt-Programmierung mit dem RTX-2000/1A. Bei `Include RTXTEST.SCR` wird auch das File UCODE.SCR nachgeladen, falls es nicht schon vorhanden ist. Es stehen danach folgende Befehle zur Verfügung:

EI	(--)	Interrupt erlauben
DI	(--)	Interrupt abschalten
WAIT	(--)	100000 Takte (15ms bei 6.67MHz) warten
CO	(-- addr)	Variable mit Zählerstand
+CO	(--)	Interrupt-Programm (erhöht **CO**)
INTTEST	(--)	Beispielprogramm

Durch Aufruf von **INTTEST** wird das Interrupt-Programm gestartet. Alle 15 Millisekunden wird dann bis zum Tastendruck der aktuelle Wert der Variable **CO** ausgegeben. Da die Interruptroutine sehr kurz ist, wird die Datenübertragung zwischen Terminalprogramm und KKF_FG bei 38400 Baud nicht gestört.

Die Vorbereitung für ein Timer0-Interruptprogramm ist dabei sehr einfach:

- Merken des alten Interruptvektors (hier Inhalt von $FD00 zum Returnstack)
- Neuer Interruptvektor setzen (Codefeldadresse von +CO durch 2 teilen)
- Zähler zurücksetzen
- Timer0 auf 10000 Takte setzen
- In dem Interrupt-Maskenregister nur Timer0-Interrupt aktivieren
- Interrupt mit **EI** allgemein erlauben

Am Ende des Programms wird der Interrrupt mit **DI** gesperrt und der alte Interrupt-Vektor zurückgeschrieben.

3.6 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 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 kann 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.

3.7 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 muß mit dem KKF_PC kompiliert werden und benötigt dazu auch den 8086-Assembler. All diese Files sind auf der Terminaldiskette vorhanden. 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.

3.8 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.

3.9 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 wie beim KKF_PC das CTRL+X verwendet.

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

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

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

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

#TIB-MAX (-- 79) Con
Der Wert wird von **QUERY** verwendet und ist beim KKF_FG 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 Befehl ist im KKF_FG V1.2/0 nicht verfügbar.

(MFREE (ptr -- error)
Dieser Befehl ist im KKF_FG V1.2/0 nicht verfügbar.

(MRELOC (ptr len. -- error)
Dieser Befehl ist im KKF_FG V1.2/0 nicht verfügbar.

+UCODE (name ; code --)
Der angegebene Code wird unter dem nachfolgenden Name als +UCODE-Definition gespeichert. Bei Verwendung des Namens in :-Definitionen wird dann der Code zusammen mit dem vorher kompilierten Kurzliteral (Wertebereich: 0...31) im Dictionary abgelegt. Im Interpretermodus erwartet es noch eine Adreßangabe (ebenfalls 0...31), legt diese zusammen mit dem Code in den SYSVAR-Bereich und ruft ihn dort auf.
Beispiel:
```
$BE00 UCode p@
3 p@                                    ( CR auslesen )
```

,A (cfa --) DX
Die angegebene Codefeldadresse wird ins Dictionary übernommen und dazu durch 2 geteilt.

,C (w --) DX

Der angegebene 16Bit-Wert wird unverändert ins Dictionary übernommen. **,C** wird zum Beispiel in OPTICOMP.SCR auf einen optimierenden Kompilier umgeleitet.

?BRANCH (f --) 83 I,R

?BRANCH legt den Code $8800 im Dictionary ab. Die Rücksprungadresse muß von **<RESOLVE** oder **>RESOLVE** direkt in den Code eingefügt werden. Dabei kann das Sprungziel nur in der gleichen oder in einer benachbarten 512Byte-Bank sein.

AT (x y --) UV

Dieser Befehl wird unter Verwendung von **COMMAND!** realisiert und kann deshalb nur mit einem KKF-Terminal verwendet werden. Die Statuszeile des Terminalprogrammes können nicht erreicht werden.

AT? (-- x y) UV

Dieser Befehl wird unter Verwendung von **COMMAND!** realisiert und kann deshalb nur mit einem KKF-Terminal verwendet werden.

BRANCH (--) E83 I,R

BRANCH legt den Code $9000 im Dictionary ab. Die Rücksprungadresse muß von **<RESOLVE** oder **>RESOLVE** direkt in den Code eingefügt werden. Dabei kann das Sprungziel nur in der gleichen oder in einer benachbarten 512Byte-Bank sein.

BOOT (--)

Beim Start des Systems wird das EPROM in die RAM-Bank 0 übertragen und die Interruptvektoren auf eine Fehlerroutine im KKF_FG umgeleitet.

BYE (--) 83

Nach dem Schließen des noch offenen Files wird auf Adresse 0 der ROM-Bank 0 gesprungen. Dort steht normalerweise ein Sprung auf die Boot-Routine, die dann für die eigenliche Initialisierung durchführt. Bis auf die schon veränderten Prozessorregister reagiert das KKF_FG wie nach dem Einschalten.

CELL+ (addr1 -- addr2) ANSI

Beim KKF_FG ist **CELL+** eine ALIAS-Definition von **2+** .

CELLS (n -- addr) ANSI

Beim KKF_FG ist **CELLS** eine ALIAS-Definition von **2*** .

CHAR+ (addr1 -- addr2) ANSI

Beim KKF_FG ist **CHAR+** eine ALIAS-Definition von **1+** .

CHARS (n -- len) ANSI I

Beim KKF_FG ist **CHARS** eine ALIAS-Definition von **NOOP** und mit **IMMEDIATE** gekennzeichnet.

CLS (--) UV

Dieser Befehl wird unter Verwendung von **COMMAND!** realisiert und kann deshalb nur mit einem KKF-Terminal verwendet werden. Es wird nur der Bildschirm zwischen den Statuszeilen gelöscht.

CS@ (-- ptr)

CS@ liefert im KKF_FG einen Pointer an den Anfang des aktuellen Codesegments mit Adreßoffset 0. Das KK-FORTH selbst startet in Bank 0, wo auch die Interrupt-Tabelle immer stehen muß.

D>PTR (d -- ptr)

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

bank = High-Wort von d
addr = Low-Wort von d

Daraus ergibt sich, daß **D>PTR** wie auch **PTR>D** eine ALIAS-Definition von **SWAP** ist.

DS@ (-- ptr)

Im KKF_FG sind Programm, Daten und Stack im gleichen Segment und liefern deshalb den gleichen Pointer (ptr=CS-Bank:$0000).

L (scr --)

Mit **L** wird der Editor des Terminalprogrammes aufgerufen. Dieser Befehl ist unter Verwendung von **COMMAND!** realisiert und kann nur mit einem KKF-Terminal verwendet werden.

L! (w ptr --)

Der 16Bit-Wert w wird ab der Speicheradresse ptr (bank:addr) abgelegt. Dabei wird das höherwertige Wort zuerst und innerhalb der Wörter zuerst das höherwertige Byte abgelegt. Die Bankadresse wird bei der Adreßerhöhung nicht verändert. Der Wortzugriff muß immer bei einer geraden Speicheradresse beginnen, weil sonst die nächst niedrigere Adresse verwendet und das Low-/High-Byte vertauscht wird.

L2! (dw ptr --)

Der 32Bit-Wert dw wird ab der Speicheradresse ptr (bank:addr) abgelegt (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 an der 32Bit-Adresse ptr (bank:addr) abgelegt.

LC@ (ptr -- char)

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

LCMOVE (ptr1 ptr2 len --)

Beim KKF_FG bleiben die Segmentnummer in ptr1 und ptr2 konstant. Deshalb wird nach Erreichen der Adresse 65535 wieder bei 0 begonnen.

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

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

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

LIMIT (-- sys)
LIMIT ist im KKF_FG auf $FC00 (gesamter Speicher ohne Interrupt-Tabelle) gesetzt. Er kann aber auch durch Veränderung der LIMIT-Angabe im SYSCON-Bereich herabgesetzt werden.

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 der Wert -1 zurückgeliefert.

M/MOD (d n -- r q)
Bei Überlauf während der Division wird als Quotient -1 zurückgeliefert. Der Rest entspricht aber dem Teiler oder 1 bei Division durch 0.

MAXAT (-- x y) UV
Dieser Befehl wird unter Verwendung von **COMMAND!** realisiert und kann deshalb nur mit einem KKF-Terminal verwendet werden. Das Terminalprogramm liefert die verfügbare Bildschirmgröße ohne die beiden Statuszeilen zurück.

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

NEXT-LINK (-- addr) SV
Diese Variable wird im KKF_FG nicht verwendet.

NEXTBRANCH (-- ; R: n -- n-1 |) R
NEXTBRANCH legt den Code $9800 im Dictionary ab. Die Rücksprungadresse muß von **<RESOLVE** oder **>RESOLVE** direkt in den Code eingefügt werden. Dabei kann das Sprungziel nur in der gleichen oder in einer benachbarten 512Byte-Bank sein.

P! (w addr --)
Im KKF_FG wird ein 16Bit-Schreibzugriff auf die angegebene G-Bus-Adresse durchgeführt. Da es sich um einen +UCODE-Befehl handelt, muß die Zieladresse in :-Definitionen als Kurzliteral kompiliert sein. Es sind deshalb keine Zählerschleife mit unterschiedlichen Adressen möglich.

P@ (addr -- w)

Siehe **P!** .

PAUSE (--) D

Momentan ist nur ein aktiver Task verfügbar. Taskumschaltung ist nicht implementiert, da dazu eine Auslagerung des Daten- und Returnstacks erforderlich währe.

PC! (char addr --)

Dieser Befehl ist im KKF_FG V1.2/0 nicht verfügbar.

PC@ (addr -- char)

Dieser Befehl ist im KKF_FG V1.2/0 nicht verfügbar.

PTR>D (ptr -- d)

Die Pointer-Angabe seg:addr wird durch folgende Formel in einen 32Bit-Wert umgewandelt:

d = Bank * 65536 + addr

Daraus ergibt sich, daß **PTR>D** wie auch **D>PTR** eine ALIAS-Definition von **SWAP** ist.

R0 (-- addr) U

R0 enthält die Endadresse+1 des Returnstacks. Beim Ablegen eines Wertes wird der Returnstackzeiger um zwei erniedrigt. Da der Returnstack in Hardware implementiert ist, bleiben aber die Speicherzellen des angegebenen Bereiches unverändert.

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.

RTX2001A? (-- addr) V

In der Variablen **RTX2001A?** wird ein Flag aufbewahrt, ob es sich beim Prozessor um einen RTX-2001A handelt. Dieses Flag darf nicht verändert werden, weil sonst die Befehle **PICK** , **SP@** , **SP!** , **RP@** und **RP!** falsche Werte liefern oder einen Absturz verursachen.

S0 (-- addr) U

S0 enthält die Endadresse+1 des Datenstacks. Beim Ablegen eines Wertes wird der Datenstackzeiger um zwei erniedrigt. Da der Datenstack in Hardware implementiert ist, bleiben aber die Speicherzellen des angegebenen Bereiches unverändert.

SFLAG (-- sys) SV

Folgende Bits von **SFLAG** werden im KKF_FG V1.2/0 verwendet:

Bit 0=1:	Keine Befehlszeile vom Betriebssystem (immer auf 1 gesetzt)
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@ (-- addr)

SP@ liefert die aktuelle Inhalt des Datenstackzeigers.

SS@ (-- ptr)

Da die beiden Stacks in Hardware realisiert sind, ist dieser Befehl im KKF_FG nicht verfügbar. Falls einmal ein Zugriff auf den vorgesehenen Stackbereich nötig ist, so kann **DS@** verwendet werden.

SYSCON (-- $0000)

Der SYSCON-Bereich beginnt am EPROM-/RAM-Anfang ab Adresse $0000.

SYSVAR (-- $0010)

Gleich hinter dem SYSCON-Bereich folgen die System-Variablen.

SYSVARLEN@ (-- $50)

Der SYSVAR-Bereich hat eine Länge von 80 Bytes oder 40 Zellen.

UCODE (name ; code --)

Der angegebene Code wird unter dem nachfolgenden Name als UCODE-Definition gespeichert. Bei Verwendung des Namens in :-Definitionen wird dann der Code im Dictionary abgelegt. Im Interpretermodus kopiert die Runtime-Routine den Code in den SYSVAR-Bereich und ruft ihn dort auf.
Beispiel: $A0C0 UCode dup

UFLAG (-- sys) SV

UFLAG kann für eigene Zwecke verwendet werden.

V (--)

Mit **V** wird der Editor des Terminalprogrammes mit Cursor nach dem letzten Fehler aufgerufen. Dieser Befehl ist unter Verwendung von **COMMAND!** realisiert und kann nur mit einem KKF-Terminal 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
$7f08	Nicht initialisierter Interrupt
$7f09	Fehlerhafte Adresse
$7f0a	DEFER-Definition nicht initialisiert
$7f0b	Dictionary voll
$7f0c	USER-Bereich voll
$7f0d	CONTEXT-Bereich voll
$7f0e	DP liegt im HEAP
$7f0f	Befehl ist geschützt
$7f10	Name erwartet
$7f11	Name nicht gefunden
$7f12	Es wurde kein Befehl definiert
$7f13	Befehl ist nur in :-Definitionen zulässig
$7f14	Fehlerhafte Kontrollstruktur
$7f15	Es folgte nach IS kein DEFER-Befehl
$7f16	File nicht geöffnet
$7f17	Blocknummer zu groß
$7f18	Blocknummer nicht erlaubt (z.B. 0 bei LOAD)
$7f19	Fehler bei Terminal-Befehlsübertragung
$7f1a	Fehler bei UVECTOR-Befehl (Tabelle zu kurz)
$7f1b	Befehl wird nicht unterstützt

Anhang C

Terminalbefehle

F1
ALT+H Anzeige der Hilfsinformation zur Tastenbelegung

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

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

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

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

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

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

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

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

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

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

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

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

Anhang D

Editor-Tastenbelegung

Cursorsteuerung:

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

Zwischenspeicherung von Zeichen und Zeilen:

F1	Zeichen speichern und löschen
F2	Zeichen speichern, Cursor rechts
F3	Zeichen einfügen
F5	Zeile speichern und löschen
F6	Zeile speichern, Cursor in die nächste Zeile
F7	Zeile einfügen

Steuerung der Zeicheneingabe und des Löschens:

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

Suchen und Ersetzen:

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

Speicherung:

F10	Änderungen in diesem Screen rückgängig machen
ESC	File speichern, Editor verlassen

Zusätze bei EDITOR.COM:

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

Anhang E

Programmlisting

P7_TERM.SCR

```
         Screen # 0
 0 \\ Interface-Routinen für das FG-Board          ( 09.07.91/KK )
 1
 2 System:         KKF_FG V1.2/0
 3 Änderung:       07.08.91  KK: Ein-/Ausgabe über Port 7
 4
 5                     Hinweise:
 6  - Teiler für Baudrate wird im SYSVAR-Bereich untergebracht
 7  - Zusätzliche Variable: lastkey
 8
 9
10
11
12
13
14
15
         Screen # 1
 0 \ Loadscreen                                    ( 27.06.91/KK )
 1
 2   { debug? @   debug? off }
 3   &02 load           \ Baudraten-Ermittlung
 4   { debug? ! }
 5   &03 &10 thru       \ Ein-/Ausgabe
 6   &11 &16 thru       \ Fileinterface
 7   &17 &18 thru       \ Vektorisierung und Initialisierung
 8
 9
10
11
12
13
14
15
         Screen # 2
 0 \ (b/2_del  msbaud                              ( 29.08.91/KK )
 1 | Variable lastkey            ( Zeichencode der letzten Taste )
 2 | : (b/2_del    ( -- ) ( Einen halben Takt warten )
 3     p_baud @ FOR NEXT ;      ( benötigt bei p_baud=0  8 Takte )
 4
 5 | : msbaud      ( Ausgang initialisieren, Baudrate? )
 6     sflag @  dup 8 or  sflag !   8 and  ?exit      ( schon da )
 7     -1 lastkey !   1 $1f p!            ( Ausgang auf Ruhelage )
 8     0                                   ( Zähler zurücksetzen )
 9     BEGIN  $1f p@  2 and  2 xor  UNTIL  ( Warten auf Startbit )
10     BEGIN  1+  $1f p@  2 and  UNTIL           ( Bis wieder 1 )
11     3 $1f p!                                ( RTS blockieren )
12     4 *  &23 -  u2/  dup p_baud !         ( Takte pro 1/2 Bit )
13     p_baud  sysvar -  (sysvar +  !  ( auch im (SYSVAR-Bereich )
14     $0a FOR  (b/2_del (b/2_del  NEXT ;      ( Bis Zeichenende )
15
```

```
         Screen # 3
 0 \ ((emit?  ((emit  ((type                          ( 05.08.91/KK )
 1 | : ((emit?     ( -- f )
 2 \   -1 ;
 3     $1f p@  1 and  0= ;
 4
 5 | : ((emit      ( char -- ) ( Hauptschleife: 26 Takte )
 6     BEGIN $1f p@ 1 and 1- UNTIL                    ( Enable )
 7     2 $1f p!  (b/2_del (b/2_del                 ( Startbit )
 8     7  FOR  dup 1 and 2 or  $1f p!  2/ noop, noop, ( Ausgeben )
 9             (b/2_del  (b/2_del     NEXT  drop      ( Warten )
10     3 $1f p!  5 FOR (b/2_del NEXT ;             ( 3 Stopbits )
11
12 | : ((type      ( addr n -- ) ( nur an Konsole )
13     ?DUPIF  1- FOR count emit NEXT  THEN  drop ;
14
15


         Screen # 4
 0 \ ((cls  ((maxat  ((at  ((at?                    ( 02.06.91/KK )
 1 | : ((cr        ( -- ) ( CR/LF ausgeben )
 2     #cr emit   $0a emit ;
 3 | : ((del       ( -- ) ( BACKSPACE )
 4     #del emit  #bl emit  #del emit ;
 5 | : ((bell      ( -- ) ( Ton )
 6     7 emit ;
 7 | : ((cls       ( -- ) ( Bildschirm durch rollen löschen )
 8     $0101 command! error ;
 9 | : ((maxat     ( -- xmax ymax ) ( Default: 40*8 )
10     $0104 command! error  com>w  com>w ;
11 | : ((at        ( x y -- ) ( nächste Zeile einrücken )
12     $0103 command!  ?DUPIF nip nip error THEN
13     w>com  w>com ;
14 | : ((at?       ( -- x y ) ( Videoposition abfragen )
15     $0102 command! error  com>w  com>w ;


         Screen # 5
 0 \ testkey  ((key?                                ( 03.08.91/KK )
 1 | : testkey     ( -- )
 2     &1000 BEGIN 1- dup
 3           WHILE $1f p@  2 and  2 xor
 4           UNTIL 0= IF exit THEN
 5     (b/2_del   3 $1f p!          ( Startbit, RTS zurücknehmen )
 6     0  7                             ( Anfangswert, Anzahl-1 )
 7     FOR    (b/2_del  (b/2_del
 8             $1f p@ 2 and IF $100 or ELSE $0ff and noop, THEN 2/
 9     NEXT (b/2_del  (b/2_del                ( Stopbit abwarten )
10     lastkey ! ;
11
12 | : ((key?     ( -- f )
13     lastkey @ 0<  IF   1 $1f p!   testkey THEN
14     lastkey @ 0<  IF   3 $1f p!   testkey THEN
15     lastkey @ 0< not ;
```

```
         Screen # 6
 0 \ ((key  ((string                                  ( 03.08.91/KK )
 1 | : ((key       ( -- char )
 2     lastkey @  dup 0< not  IF -1 lastkey ! exit THEN  drop
 3     1 $1f p!                                    ( RTS ausgeben )
 4     0  7                                ( Anfangswert, Anzahl-1 )
 5     BEGIN  $1f p@  2 and  2 xor  UNTIL  (b/2_del   ( Startbit )
 6     3 $1f p!                               ( RTS zurücknehmen )
 7     FOR    (b/2_del  (b/2_del
 8            $1f p@ 2 and IF $100 or ELSE $0ff and noop, THEN 2/
 9     NEXT (b/2_del  (b/2_del ;              ( Stopbit abwarten )
10
11 | : ((STRING    ( addr len -- ) ( Empfang eines Strings )
12     ?DUPIF  1- FOR key over c! 1+ NEXT  THEN  drop ;
13
14
15



         Screen # 7
 0 \ ((editstring-Zusätze                             ( 27.06.91/KK )
 1 | : del's       ?DUPIF 1- FOR #del_out emit NEXT THEN ;
 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                                     ( 01.06.91/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 $1b 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                          ( 01.06.91/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             $18   OF del-line                      ENDOF
10             $08   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                                        ( 03.06.91/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>$                                ( 27.06.91/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     BEGIN key tuck over c! 1+ swap 0= UNTIL drop ;
14
15
```

```
         Screen # 12
 0 \ command!                                      ( 02.06.91/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?  &27 emit  key_err?       ( ESC-Befehl schicken )
 6     w>com  com>w ;
 7
 8 \ Bei COMMAND!-Error ein oder zwei Parameter verwerfen
 9 | : 1com!       command!  ?DUPIF nip rdrop exit THEN ;
10 | : 2com!       command!  ?DUPIF nip nip rdrop exit THEN ;
11
12
13
14
15


         Screen # 13
 0 \ Diskverwaltung: ((file? ... ((file-close      ( 03.08.91/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!  ?DUPIF >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   ( 27.06.91/KK )
 1 | : ((file-read         ( seg addr len id -- error )
 2     $0209 command!  ?DUPIF  nip nip nip nip  exit  THEN
 3     w>com  dup w>com   com>w ?dup  IF  nip nip nip  exit  THEN
 4     ?DUPIF 1- FOR 2dup key -rot lc! 1+ NEXT THEN 2drop  0 ;
 5
 6 | : ((file-write        ( seg addr len id -- error )
 7     $020a command!  ?DUPIF  nip nip nip nip  exit  THEN
 8     w>com  dup w>com
 9     ?DUPIF 1- FOR 2dup lc@ emit 1+ NEXT THEN  2drop
10     com>w ;
11
12 | : ((file-free         ( dev -- free. max. 0 | error )
13     $020d 1com!   w>com   com>w ?dup ?exit
14     com>d  com>d  0 ;
15


         Screen # 16
 0 \ ((file-first  ((file-next                      ( 03.08.91/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     msbaud    ((emit?  ((emit  ((type   ((cr  ((del
 3               ((bell   ((cls   ((maxat  ((at  ((at? ;
 4
 5   &05 input  UTable: (input
 6     msbaud    ((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                                ( 04.08.91/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           ( -- )          ?open  scr @  r# @   ((ed ;
13   : l           ( scr -- )      ?open         0      ((ed ;
14
15
```

Index

(FILE-POS! 21
,A 21, 32
,C 22, 33
?BRANCH 21
+UCode 12
+UCODE 32
>ERRORTEXT 21
>PRINTER 29
32Bit-Adressen 18
Anwender-Arbeitsbereich 14
Attribut-Wort 11
Befehlsaufbau 14
Bitinversion 26
BRANCH 21
Carry-Bit 26
CODE? 27
Copyright 2
CRC-Prüfsumme 26
CREATE-DOES>-Konstruktion 17
D>PTR 19, 34
Daten- und Returnstackbefehle 22
Dekompiler 26
DI 28
dir 21
Disassembler 26
Diskpuffer 13
Editor 29
Editor-Tastenbelegung 40
EI 28
Fehlerliste 38
G! 20
G@ 20
Glossar-Zusatz 31
Handlenummer 21
Hardware-Multiplikator 27
Installation 7
Interrupt 5
Interrupt-Tabelle 10, 14
INTTEST 28
KEY 23
KKF_FG-Diskette 6
KKF-Terminaldiskette 6
Kompilieren der Codefeldadresse 21
Kontrollstrukturen 21
LIST: 29
MEM_EDIT.SCR 29
Multiplikation 5
NEC-P6 29
NECP6.SCR 29
NECP6_T.SCR 28
NEXTBRANCH 21
OF(..... 27
OPTICOMP.SCR 27
Optimierende Kompiler 27
P! 20, 35
P@ 20, 36
P7_TERM.SCR 41
PC! 9
PC@ 9
PC_TERM.SCR 30
PERFORM 21
Portbefehle 20
Programmlisting 29, 41
ptr 18
PTR>D 19, 36
Quadratwurzel 26
RTX2001A? 27, 36
RTXDIS.SCR 26
RTXTEST.SCR 28
SFLAG 12
Sieb des Eratosthenes 28
SIEVE.SCR 28
Speicheraufteilung 9
Systemkonstanten 10
Systemvariablen 10
Sytem-Arbeitsbereich 14
Taskbereich 13
Terminalbefehle 39
Terminalprogramm 30
Timer0 28
UCode 12
UCODES.SCR 26
UFLAG 12
USER-Bereich des KKF_FG 13
USER-Bereich des Prozessors 20
VCREATE-VDOES>-Konstruktion 17