Klaus Kohl

Zusatzhandbuch

zum

KKF_8415

für den

mc-Z80-MINI-EMUF

Version 1.2/0

- Hinweise zum KKF_8415

- Informationen zum Hardware des mc-Z80-EMUF

- Beschreibung der Zusatzprogramme

Wichtige Hinweise:

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

Nachdruck und öffentliche Wiedergabe, besonders die Übersetzung in andere Sprachen ist nur mit schriftlicher Genehmigung des Autors erlaubt. Jede Vervielfältigung und Weitergabe vom Programm und den Beispielen wird strafrechtlich verfolgt. Die Rechte an der Dokumentation und dem KKF-System liegen bei Klaus Kohl.

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

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

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Vorwort

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

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

1.2 Das KKF für den mc-Z80-MINI-EMUF

Das KKF_8415 ist eine Anpassung des KK-FORTH an den mc-Z80-MINI-EMUF. Als Hauptprozessor wird das Z80-Derivat TMPZ84C015 verwendet. Dieser Prozessor besitzt einen Z80-Kern, hat aber schon eine SIO mit zwei seriellen Schnittstellen, eine PIO mit zwei 8Bit-Ports, eine CTC mit drei 16Bit-Zähler und eine Watchdog in einem Gehäuse.

Das dafür verfügbare KK-FORTH benötigt ein mindestens 16KByte großes EPROM am Speicheranfang und ein mindestens 4 KByte großes RAM am Speicherende. Die ausgelieferte Version ist für je 32KByte RAM und EPROM ausgelegt, was auch der üblichen Konfiguration entspricht. Es kann im EPROM ab Adresse $4000 ein Zusatzimage abgelegt und dabei auch ein Autostart-Programm erzeugt werden.

Die Kommunikation mit dem KKK_8415 erfolgt wie auch beim Monitor über die zweite serielle Schnittstelle mit 9600 Baud. Das mitgelieferte Terminalprogramm übernimmt beim Arbeiten mit dem KK-FORTH die Funktion des Fileservers und erlaubt dadurch das Laden von Programmen und die Speicherung des Zusatzimage. Die Baudrate kann problemlos sowohl beim KKF_8415 als auch im Terminalprogramm bis auf 38400 (bzw. 115200) Baud gesteigert werden.

1.3 Lieferumfang

Mit dieser Beschreibung erhalten Sie zwei Disketten (5.25" oder 3.5" - 360K). Die erste Diskette enthält das Kernprogramm KKF_8415.COM, ein Zusatzimage mit Assembler, Disassembler und Debugger. Zusätzlich zu den verschiedenen Tools sind auf der gleichen Diskette alle Beispielprogramme des Handbuches und zwei speziell für den EMUF erstellte Files abgelegt. Die zweite Diskette wird zu allen Terminalversionen mitgeliefert und enthält neben dem KKF_PC mit Assembler und Druckertreiber noch einen FORTH-Screeneditor und das Terminalprogramm.

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

KKF_8415-Diskette

```
READ     ME      2435  11.09.91  15.33
TERMINAL COM    28828  29.08.91  16.48  Terminalprogramm (COM2-9600)
FED      COM    23999  29.08.91  16.40  FORTH-Screeneditor
-KKF8415 ---        0  21.12.90   9.34
KKF_8415 COM    16541  11.09.91  15.30  Bitimage des KKF_8415-Kerns
ASSDISDE BIN     9686  11.09.91  15.32  Zusatzimage mit allen Tools
Z84C015  SCR     8192  24.08.91  16.43  Beispiele für den Z80-EMUF
RTCTEST  SCR     6144  21.08.91  23.17  Ansteuerung der Echtzeituhr
--FILES- ---        0  21.12.90   9.34
HD64180  ASM    70656  10.08.91  14.16  Z80- bzw. HD64180-Assembler
Z80DIS   SCR    20480  10.08.91  12.53  Z80-Disassembler
Z80DEBUG SCR    17408  10.08.91  14.04  Z80-Dekompiler/Debugger
NECP6_T  SCR    21504  29.08.91   9.38  Druckertreiber für KKF_8415
---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_8415

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_8415 (bzw. KKF_TERM für die zweite Diskette) ein. In dieses Unterverzeichnis kann man dann alle Files der ersten (zweiten) Diskette kopieren. Die Programme TERMINAL.COM und FED.COM sollten immer in diesem Unterverzeichnis gestartet werden, weil sie keinen automatischen Pfadwechsel unterstützen.

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 die ersten 16KByte des Files KKF_8415.COM in ein 16- oder 32KByte-EPROM gebrannt und an Stelle des Monitor-EPROM in den EMUF eingesetzt werden. Falls bei den Tests auch der Assembler, Disassembler und Debugger verfügbar sein soll, so ist zuerst das File KKF_8415.COM ab Adresse 0 und dann das Zusatzimage ASSDISDE.BIN ab Adresse $4000 in das EPROM zu übernehmen.

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

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

Sollte sich das KKF_8415 nicht melden, so ist zuerst mit einem Schnittstellentester zu ermitteln, ob die Leitungen richtig gekreuzt sind und ob das Terminalprogramm bei Tastendruck sendet. Anschließend kann man mit dem ursprünglichen Monitor-EPROM ermitteln, ob die Schnittstelle des EMUF's funktioniert. Falls alles zutrifft, so kann nur ein fehlerhaftes EPROM die Ursache sein.

Obwohl der mitgelieferte Editor FED.COM und der Editor des Terminalprogramms direkt auf den Bildschirmspeicher greifen, dürfte es nur wenige Probleme mit der verwendeten Hardware geben. Es wird sowohl der EGA-Modus 3 (80 Zeichen, mindestens 25 Zeilen) als auch der Herkules-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_8415

Bei der Erstellung oder beim Test neuer Programme für das KKF_8415 kann immer die Version mit dem Zusatzimage verwendet werden. Es entfällt dann das lästige Nachladen der Tools. Man sollte aber dann keine Applikationen damit erstellen, da der noch freie EPROM-Bereich doch schon sehr klein ist.

Bei `SAVESYSTEM Filename` wird nur das Zusatzimage gespeichert. Es kann dann ab der EPROM-Adresse $4000 zur Kopie des ursprünglichen EPROM's eingefügt werden. Das Image darf aber nicht länger als 16384 Bytes sein, weil es sonst nicht mehr in das EPROM paßt.

Wer das GAL umprogrammieren kann, hat auch die Möglichkeit zur Umstellung der Spiecherkonfiguration. Er könnte dann während der Programmentwicklung 48KByte RAM verwenden. Dazu ist im EPROM die RAM-Anfangsadresse auf $4000 und die Image-Adresse auf 0 zu setzen. Wird dann später das Image wieder bei ursprünglicher Konfiguration verwendet, so bleibt das Programm (z.B. Assembler) im ROM und der RAM-Bereich ist für Daten frei.

Wenn der KKF_8415-Kern läuft, so kann wie im Handbuch beschrieben gearbeitet werden. Das Terminalprogramm sendet alle von Tastatur eingegebenen Zeichen an das KKF_8415 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_8415 als auch von Tastatur aus gesperrt werden.

Bei allen Übertragungen muß 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.

Kapitel 2

Beschreibung

2.1 Allgemeines zum KKF_8415

Das KKF_8415 sind eine KKF-Version, bei dem der Kern in einem 16KByte-EPROM ab Adresse $0000 steht und ab Adresse $8000 ein 32KByte-RAM erwartet. Ab Adresse $4000 kann ebenfalls im EPROM oder (bei Umprogrammierung des GAL's) im EEPROM ein Zusatzimage stehen. Dieses Image wird beim Einschalten erkannt, an die richtige Speicheradresse kopiert und der ebenfalls im Image gespeicherten Variablen- und USER-Bereich übernommen. Durch Änderung der DEFER-Wörter im Zusatzimage können auch Autostart-Programme erstellt werden.

Als Schnittstelle verwendet das KKF_8415 den RS232-Port 2 mit 9600 Baud und 8 Datenbits. Da die Angabe für den Timer im SYSVAR-Bereich gespeichert sind, kann die Baudrate sowohl im EPROM als auch mit einem Zusatzimage geändert werden.

Der KKF-Kern kopiert beim Start die FORTH-Kernroutine (NEXT an das Ende des RAM's. Dadurch ist es möglich, daß auch Befehle des KKF-Kerns mit dem Debugger im Einzelschritt-Modus abgearbeitet werden können.

Um dem Anwender die Möglichkeit zur Interrupt-Programmierung offenzulassen, sind im EPROM die RST-Vektoren $10 bis $38 und der NMI auf eine Tabelle im RAM umgeleitet. Diese Tabelle wird beim Start des KK-FORTH so initialisiert, daß der Interrupt abgeschaltet und eine Fehlermeldung ausgegeben wird. Die RST-Vektoren $00 und $08 können nicht verwendet werden, da sie durch den SYSCON-Bereich belegt sind.

Der Prozessor verfügt über eine Watchdog, die extern durch einen Jumper mit dem Reset verbunden werden kann. Der hier erstmal schon vom System verwendete Vektor **'BOOT** zeigt auf eine Routine, welche die Watchdog abschaltet. Dadurch kann sogar während der Entwicklung die Verbindung hergestellt und durch Befehle des Files Z84C015.SCR die Watchdog gesteuert werden. Falls eine Applikation in ihrer Hauptschleife die Watchdog immer wieder zurücksetzt, sollte die Abschaltung durch Umleitung von **'BOOT** auf **NOOP** aus der Einschaltsequenz entfernt werden.

2.2 Speicheraufteilung

Die folgenden Adressen beziehen sich auf das KKF_8415 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		`CALL $FFEB : EI : RETI` (RST $10 umleiten)
$0018		`CALL $FFEE : EI : RETI` (RST $18 umleiten)
$0020		`CALL $FFF1 : EI : RETI` (RST $20 umleiten)
$0028		`CALL $FFF4 : EI : RETI` (RST $28 umleiten)
$0030		`CALL $FFF7 : EI : RETI` (RST $30 umleiten)
$0038		`CALL $FFFA : EI : RETI` (RST $38 umleiten)
$0040		Kopie der (NEXT-Routine
$0050		Kopie der Interrupt-Zeiger
$0066		`CALL $FFFD : EI : RETN` (NMI umleiten)
$006B		Anfangswerte des SYSVAR-Bereiches
$00BB ... $3F60		Kern-Dictionary

$3F61 ... $3F9C		Kopie des Variablenbereiches im Kern
$3F9D ... $3FFF		Kopie des TASK0-Bereiches (nur Anfang)
$4000		Anfangsadresse des Zusatzimage
$8000	(SYSVAR	RAM-Anfangsadresse
$8050 ... $F1B6	HERE	Freien Programmbereich
$F1B7	VDP @	Anfang des Variablenbereiches
$F247	HDP @	Anfang des Heap (beim Start leer)
$F247	TDP @	Anfang des Taskbereiches
$F5A1	TASK0	Anfang des USER-Bereiches im TASK0
$F8F9	FIRST	Diskpuffer
$FCFB	SYSVAR $2C + @	Anfang des Puffers für 8 Befehlszeilen
$FF8B	SYSVAR $30 + @	Anfang des Anwender-Arbeitsbereich (leer)
$FF8B	SYSVAR	Anfang des SYSVAR-Bereiches
$FFDB	' NOOP @ 3 -	(NEXT-Routine mit Zusätze
$FFEB		`JMP $13E3` (Fehlermeldung bei RST $10)
$FFEE		`JMP $13E3` (Fehlermeldung bei RST $18)
$FFF1		`JMP $13E3` (Fehlermeldung bei RST $20)
$FFF4		`JMP $13E3` (Fehlermeldung bei RST $28)
$FFF7		`JMP $13E3` (Fehlermeldung bei RST $30)
$FFFA		`JMP $13E3` (Fehlermeldung bei RST $38)
$FFFD		`JMP $13E3` (Fehlermeldung bei NMI)
$0000	LIMIT	Programmende+1

Der Prozessor startet am Anfang des EPROM's und findet dort im SYSCON-Bereich einen Sprung auf die Startroutine im KKF-Kern. Diese sorgt auch dafür, daß die (NEXT-Routine und die Interrupt-Zeiger an das Speicherende ab Adresse $FFDB kopiert werden.

Die RST-Vektoren $10 bis $38 und der NMI rufen indirekt über die Tabelle am Speicherende eine Fehlerroutine auf, weche den Interrupt abschaltet. Wie in einem Beispiel später noch gezeigt wird, kann der Vektor im RAM auf eigene Assembler-Definitionen umgeleitet werden. Da beim Einschalten der Interrupt-Mode 1 aktiviert ist, wird dazu der RST$38-Vektor verwendet.

Der KKF_8415-Kern bleibt im EPROM ab Adresse $00BB, erkennt aber ein evtl. wieder ins RAM zu übertragendes Zusatzimage ab $4000. Sowohl die Anfangsadresse des Image als auch die Anfangsadresse des RAM's kann durch Manipulation des SYSCON-Bereiches verändert werden.

2.2.1 Systemkonstanten

Die System-Konstanten enthalten Werte, die nur beim Systemstart berücksichtigt werden. Dabei sind im KKF_8415 die Angaben für Image- und RAM-Anfangsadresse veränderbar.

SYSCON	**BOOT-Routine aufrufen**
SYSCON + 4	**reserviert**
SYSCON + 8	**Speicheradresse des SYSVAR-Bereiches (fest auf $FF8B)**
SYSCON + 10	**Anfangsadresse des Zusatzimage ($4000, veränderbar)**
SYSCON + 12	**Anfangsadresse des RAM's ($8000, veränderbar)**
SYSCON + 14	**Endadresse + 1 des RAM's (fest auf $0000)**

Sowohl die Anfangsadresse des SYSVAR-Bereiches als auch die Endadresse des RAM's darf nicht geändert werden. Sie sind im SYSCON-Bereich mit $FF8B bzw. 0 angegeben und dienen nur zur Information.

Da man meistens ein 32KByte-RAM verwendet, wurde die Anfangsadresse des RAM's auf $8000 gesetzt. Falls man das Adreßdekoder-GAL verändert und ein zweites RAM einsetzt, kann man die RAM-Anfangsadresse auch auf $4000 heruntersetzten. Falls später dann ab Adresse $4000

wieder ein EPROM eingesetzt wird und dort dann das Zusatzimage ablegt, so erkennt das KKF_8415 dessen richtige Speicheradresse und läßt das Programm im EPROM.

2.2.2 Systemvariablen

Beim Start des KKF_8415 werden entweder die Orginalwerte ab Adresse $006B oder die Werte vom Anfang des Zusatzimage in den SYSVAR-Bereich übernommen. Zusätzlich wird noch der RAM-Anfang mit einer Kopie dieses Bereiches versorgt. Diese Kopie dient gleichzeitig als Header eines neuen Zusatzimages.

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

SYSVAR + 4 **Prozessorkennung**
$0224 System läuft auf einem Z84C015

SYSVAR + 6 **Hardwarekennung/Betriebssystem**
$0111 mc-Z80-MINI-EMUF mit SIO2-Interface

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

SYSVAR + 10 **Attribut-Wort**

```
%0100001100000010
 ^^              ROM/RAM-Version
  ^^             nur 64KByte verfügbar
    ^^           Ohne Floatingpoint-Unterstützung
      ^          Ein-/Ausgabe über Software-SIO
       ^         Fileinterface über Software-SIO
        ^^       Returnstack in Hardware
          ^^     Datenstack in Hardware
            ^    Kein Align beim 32Bit-Zugriff notwendig
             ^   Kein Align beim 16Bit-Zugriff notwendig
              ^^ Reihenfolge der Daten: d2 d3 d0 d1
```

SYSVAR + 12 **Zeiger auf Kopie der Systemvariablen**
$006B Wird beim Start auf Speicheranfang gesetzt

SYSVAR + 14 **VOC-LINK**
$1FC8 Zeigt auf das als einziges vorhandene FORTH-Vokabular

SYSVAR + 16 **DP**
$3F61 Der KKF_8415-Kern hat eine Länge von 16225 Bytes

SYSVAR + 18 **VDP**
$9000 Adresse wird beim Kaltstart ermittelt

SYSVAR + 20 **VLEN**
$003C Im KKF_8415-Kern werden 60 Bytes für Variablen verwendet

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

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

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

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

SYSVAR + 30 TASK
$0000 Adresse wird beim Kaltstart ermittelt

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

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

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

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

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

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

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

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

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

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

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

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
$0004 Takt wird durch 4 geteilt (entspricht 9600 Baud bei 4.9MHz)

SYSVAR + 64 frei
SYSVAR + 66 frei
SYSVAR + 68 frei
SYSVAR + 70 frei
SYSVAR + 72 frei
SYSVAR + 74 frei

SYSVAR + 76 SFLAG
Bit 0=1: Keine Start-Befehlszeile verfügbar (ist auf 1 gesetzt)
Bit 1=1: Keine Warnung durch **CREATE** ausgeben
Bit 2=1: Keine Zeilenpuffer verwenden
Bit 3=1: Die Schnittstelle ist schon initialisiert
Bit 4-15: Werden im KKF_8415 nicht verwendet

SYSVAR + 78 UFLAG
Frei für Anwender

Zur Kommunikation mit dem Terminalprogramm wird die zweite serielle Schnittstelle verwendet. Bei der Umstellung der Ein-/Ausgabe auf **(OUTPUT** bzw. **(INPUT** wird Bit 3 von **SFLAG** untersucht. Ist es noch auf 0, so wird die Schnittstelle erst initialisiert und dabei der in SYSVAR+&62 gespeicherte Teilerwert verwendet. Man kann durch Veränderung dieses Wertes im EPROM und im Zusatzimage eine andere Baudrate einstellen (1=38400; 2=19200; 128=300 Baud). Später wird noch beschrieben, wie man andere Baudraten auch ohne EPROM-Änderungen einstellen kann.

Die 6 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_8415 ist nur ein Task eingerichtet. Dieser Task enthält neben den beiden, jeweils für 256 Einträge vorbereiteten Stackbereiche noch Speicher für **WORD** und Zahlenstringerzeugung (84 Bytes ab **WDP@**), den Stringbereich (256 Bytes ab **PAD**) und einen 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 Daten in TASKADDR@+&16 bis TASKADDR@+&32 führen zu undefinierten Reaktionen. Die oben genannten Werte des KKF_8415-Kerns führen zu folgender Speicheraufteilung:

$F247	WDP@	Anfang des Arbeitsbereich für diesen Task - Ablagebereich für WORD
	…	
		- Arbeitsbereich für <# bis #>
$F29B	PAD	Anfang des Stringbereich - 256 Bytes für Strings - Platz für 256 Stackeinträge
$F59B	S0 @	Ende des Datenstacks
	…	(danach 6 Bytes frei)
$F5A1	TASKADDR@	Anfangsadresse des USER-Bereiches - Programm/Daten für Taskwechsel - USER-Variablen
$F6A1	TASKADDR@ $100 +	Eingabepuffers für **QUERY** (= TIB - 2)
	…	
		- Platz für 256 Returnstackeinträge
$F8F3	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 das System und für den Anwender, die (NEXT-Routine 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 bei Zugriff auf Screenfiles überschrieben wird.

Sytem-Arbeitsbereich

Wenn man das unveränderte KKF_8415 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 durch den KKF-Kern nicht verändert. Da aber beim Start der Stack an den Anfang des SYSVAR-Bereiches gesetzt und einige Werte abgelegt werden, sind die hintersten Werte nach einem Neustart des KKF_8415 verändert.

Die (NEXT-Routine

Da es sich bei dem Z84C015 um keinen FORTH-Prozessor handelt, muß der Start des nächsten FORTH-Befehls durch eine kleine Assemblerroutine erledigt werden. Damit man mit einem Debugger den FORTH-Kern und die eigenen Befehle auch schrittweise Ausführen kann, wurde diese Routine in das RAM ausgelagert. Die Routine besteht aus folgende Assemblerbefehle:

```
$FFDB   $C1   POP BC          ( BPOP-Adresse )
$FFDC   $D5   PUSH DE         ( DPUSH-Adresse )
$FFDD   $E5   PUSH HL         ( HPUSH-Adresse )

$FFDE   $0A   LD A,(BC)       ( (NEXT-Routine )
$FFDF   $6F   LD L,A
$FFE0   $03   INC BC
$FFE1   $0A   LD A,(BC)
$FFE2   $67   LD H,A
$FFE3   $03   INC BC
$FFE4   $5E   LD E,(HL)
$FFE5   $23   INC HL
$FFE6   $56   LD D,(HL)
$FFE7   $EB   EX DE,HL
$FFE8   $E9   JP (HL)
```

Um diese Routine zu verstehen, muß man den Prozessor und den Aufbau der FORTH-Befehle kennen. Die eigentliche Routine beginnt erst bei $FFDE und holt die Codefeldadresse des nächsten Befehls bei gleichzeitiger Erhöhung des FORTH-Programmzeiger BC in das 16Bit-Register HL. Die in der Codefeldadresse gespeicherten Adresse der zu dem Befehl gehörenden Assemblerroutine wird dann in das Registerpaar DE geladen, wobei die Codefeldadresse in HL nur um 1 erhöht wird. Nach Vertauschung des Inhalts der Registerpaare HL und DE wird dann die Assemblerroutine des Befehls aufgerufen.

Die vor der (NEXT-Routine abgelegten Befehle bringen noch ein oder zwei Werte auf den Datenstack oder holen vorher noch den gespeicherten Programmzeiger vom Stack. Da dies häufige Befehlsfolgen des KKF-Kerns sind, sparen sie durch diese Anbindung einige Bytes im EPROM.

Interrupt-Tabelle

Bei der Interrupt-Tabelle am Speicherende handelt es sich nur um Sprünge auf eine Fehlerroutine. Die Rücksprungadresse wird nicht benötigt, weil sie schon im RST-Vektor auf den Stack gebracht wurde. Wie später noch an einem Beispiel gezeigt wird, kann die Zieladresse des Sprunges auch eine eigene Routine sein. Falls aber die Fehlerroutine angesprungen wird, so sperrt diese alle Interrupts und bringt eine Fehlermeldung. Auf dem Stack bleibt dabei die Rücksprungadresse in den RST- oder NMI-Vektor und kann zur Ermittlung der Fehlerursache herangezogen werden.

2.3 Aufbau der Befehle

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

Ein Befehl besteht aus:

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

Die höchsten drei Bits in der Namensfeldadresse werden für die Eigenschaft des Befehls verwendet. Die unteren 5 Bits geben dann die Länge des folgenden Befehlsnamen an. Es können deshalb bis zu 31 Zeichen verwendet werden. 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 Assemblerdefinitionen im KKF_8415-Kern. Sie sind für die Tätigkeit des Befehls verantwortlich.

:-Definitionen am Beispiel :

```
2815  F7 27 01 3A DB 00 7B 04
```

$2815/6	$27F7	Verkettung zum vorherigen Befehl
$2817	$01	Namensfeldadresse (Länge: 1 Zeichen)
$2818	$3A	Befehlsname :
$2819/A	$00DB	Adresse von (DIC
$281B/C	$047B	Highlevel-Teil (beginnt mit CURRENT)

Konstanten am Beispiel #B/BLK

```
02B2  A8 02 06 23 42 2F 42 4C 4B 12 01 00 04
```

$02B2/3	$02A8	Verkettung zum vorherigen Befehl
$02B4	$06	Namensfeldadresse (Länge: 6 Zeichen)
$02B5 ...	$23 ...	Befehlsname #B/BLK
$02BB/C	$0112	Adresse von (CON
$02BD/E	$0400	Wert der Konstanten

32Bit-Konstanten am Beispiel PI

```
$3.1415 2Constant pi

8050  FF 3A 02 50 49 0B 01 15 14 03 00
```

$8050/1	$3AFF	Verkettung zum vorherigen Befehl (EPROM)
$8052	$02	Namensfeldadresse (Länge: 2 Zeichen)
$8053/4	$50 $49	Befehlsname PI
$8055/6	$010B	Adresse von (2CON
$8057..A	$0003.1415	Wert der 32Bit-Konstanten

Variablen am Beispiel FILE-ID

```
0557  4B 05 07 46 49 4C 45 2D 49 44 FF 00 00 00
```

$0557/8	$054B	Verkettung zum vorherigen Befehl
$0559	$07	Namensfeldadresse (Länge: 7 Zeichen)
$055A ...	$46 ...	Befehlsname FILE-ID
$0561/2	$00FF	Adresse von (VAR
$0563/4	$0000	Offset in den Variablenbereich

32Bit-Variablen am Beispiel COUNTER

```
2Variable counter

805B  50 80 07 43 4F 55 4E 54 45 52 FF 00 3C 00
```

$805B/C	$8050	Verkettung zum vorherigen Befehl
$805D	$07	Namensfeldadresse (Länge: 7 Zeichen)
$805E ...	$43 ...	Befehlsname COUNTER
$8065/6	$00FF	Adresse von (2VAR (gleich mit (VAR)
$8067/8	$003C	Offset in den Variablenbereich

USER-Variablen am Beispiel >IN

```
04A1  97 04 03 3E 49 4E 61 01 3A 00
```

$04A1/2	$0497	Verkettung zum vorherigen Befehl
$04A3	$03	Namensfeldadresse (Länge: 3 Zeichen)
$04A4 ...	$3E ...	Befehlsname >IN
$04A7/8	$0161	Adresse von (USER
$04A9/A	$003A	Offset in den USER-Bereich

Vokabulare am Beispiel FORTH

```
1FBE  A1 1F 05 46 4F 52 54 48 6E 01 00 00 1E 00
```

$1FBE/F	$1DDE	Verkettung zum vorherigen Befehl
$1FC0	$05	Namensfeldadresse (Länge: 5 Zeichen)
$1FC1 ...	$46 ...	Befehlsname FORTH
$1FC6/7	$016E	Adresse von (VOC
$1FC8/9	$0000	Verkettung zum nächsten Vokabular (fehlt)
$1FCA/B	$001E	Offset in den Variablenbereich
(VDP @ $1E + ergibt $F229)		
$F229/A	$3AFF	Zeiger zur BOOT-Linkfeldadresse

DEFER-Definitionen am Beispiel ERRORTEXT@

```
37CC  BB 37 0A 45 52 52 4F 52 54 45 58 54 40 1B 01 30 00
```

$37CC/D	$37BB	Verkettung zum vorherigen Befehl
$37CE	$0A	Namensfeldadresse (Länge: 10 Zeichen)
$37CF ...	$45 ...	Befehlsname ERRORTEXT@
$37D9/A	$011B	Adresse von (DEFER
$37DB/C	$0030	Offset in den Variablenbereich
(VDP @ $30 + ergibt $F23B)		
$F23B/C	$37DD	Zeigt auf die versteckte Routine

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

LABEL-Definitionen am Beispiel HEREADDR

```
label hereaddr

F23A  FF 3A A8 48 45 52 45 41 44 44 52 36 F2
F236  79 01 50 80
```

$F23A/B	$3AFF	Verkettung zum vorherigen Befehl
$F23C	$A8	Namensfeldadresse (Länge: 8 Zeichen) (Befehl ist Immediate und Indirect)
$F23E ...	$48	Befehlsname HEREADDR
$F245/6	$F236	Zeiger auf Codefeldadresse (im Heap)
$F236/7	$0179	Adresse von (LABEL
$F238/9	$8050	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 D>PTR

```
30BD  B3 30 25 44 3E 50 54 52 19 08
```

$30BD/E	$30B3	Verkettung zum vorherigen Befehl
$30BF	$25	Namensfeldadresse (Länge: 5 Zeichen) (Befehl ist Indirect)
$30C0 ...	$44	Befehlsname D>PTR
$30C5/6	$0819	Codefeldadresse von SWAP

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

8050  3A F2 0B 33 44 2D 43 4F 4E 53 54 41 4E 54 DB 00
8060  CD 26 26 08 D7 12 19 08 D7 12 D7 12 54 28 CD 45
8070  01 84 08 14 0F 19 08 E6 0A 84 08 14 0F 19 08 E6
```

$8050/1	$F23A	Verkettung zum vorherigen Befehl
$8052	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8053 ...	$33	Befehlsname 3D-CONSTANT
$805E/F	$00DB	Adresse von (DIC
$8060 ...	$26CD	Create-Teil (endet mit (;code)
$806E ... $8070	$CD $45 $01	(does> aufrufen
$8071/2 ...	$0884	Does>-Teil (endet mit exit)

```
8085  50 80 03 31 32 33 6E 80 01 00 02 00 03 00
```

$8085/6	$8050	Verkettung zum vorherigen Befehl
$8087	$03	Namensfeldadresse (Länge: 3 Zeichen)
$8088 ...	$31	Befehlsname 123
$808B/C	$806E	Aufruf des Does>-Teil von 3D-CONSTANT
$808D/E	$0001	x-Wert
$808F/90	$0002	y-Wert
$8091/2	$0003	z-Wert

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

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

```
8093  85 80 0B 33 44 2D 56 41 52 49 41 42 4C 45 DB 00
80A3  A1 27 B1 01 00 00 15 13 B1 01 00 00 15 13 B1 01
80B3  00 00 15 13 54 28 CD 53 01 19 08 AA 09 7B 0A EB
```

$8093/4	$8085	Verkettung zum vorherigen Befehl
$8095	$0B	Namensfeldadresse (Länge: 11 Zeichen)
$8096 ...	$33	Befehlsname 3D-VARIABLE
$80A1/2	$00DB	Adresse von (DIC
$80A3 ...	$27A1	VCreate-Teil (endet mit (;code)
$80B9 ... $80BB	$CD $53 $01	(vdoes> aufrufen
$80BC ...	$0819	VDoes>-Teil (endet mit exit)

```
80C4  93 80 07 56 45 43 54 4F 52 31 B9 80 3C 00
```

$80C4/5	$8093	Verkettung zum vorherigen Befehl
$80C6	$07	Namensfeldadresse (Länge: 7 Zeichen)
$80C7 ...	$56	Befehlsname VECTOR1
$80CE/F	$80B9	Aufruf des VDoes>-Teil von 3D-VARIABLE
$80D0/1	$003C	Offset in den Variablenbereich

CODE-Definition am Beispiel DROP

```
090C  F3 08 04 44 52 4F 50 15 09 E1 C3 DE FF
```

$090C/D	$08F3	Verkettung zum vorherigen Befehl
$090E	$84	Namensfeldadresse (Länge: 4 Zeichen) (Befehl ist Immediate)
$090F ...	$44 ...	Befehlsname DROP
$0913/4	$0915	Adresse der nachfolgenden Routine
$0915	$E1	hl pop,
$0916 ... $0918	$C3 $DE $FF	next,

PROC-Definition am Beispiel INTNOOP

```
Proc intnoop   ( -- ) ( nichts tun )
   ret,
  End-Proc
```

```
80D2  C4 80 07 49 4E 54 4E 4F 4F 50 FA 00 C9
```

$80D2/3	$80C4	Verkettung zum vorherigen Befehl
$80D4	$07	Namensfeldadresse (Länge: 7 Zeichen) (Befehl ist Immediate)

$80D5 ...	$49	Befehlsname INTNOOP
$80DC/D	$00FA	Adresse von (DIC
$80DE	$C9	ret,

2.4 Informationen zu KKF_8415-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.

2.4.1 32Bit-Adressen

Der Z80 und damit auch der Z84C015 hat nur einen Adressbereich von 64KByte. Die Pointer-Befehle stehen deshalb bei dieser KKF-Version nicht zur Verfügung.

Da es aber auch Z80-Derivate mit gebankter Speicheradressierung gibt (siehe HD64180), wurde bei den Filebefehlen die Angabe einer Pointeradresse beibehalten. Damit die Aufrufe wie in anderen KKF-Versionen beibehalten werden können, stehen weiterhin die Befehle **CS@** , **DS@** , **SS@** , **D>PTR** und **PTR>D** zur Verfügung. Dabei wird als Pointerangabe zuerst die Banknummer (immer 0) und darüber die Speicheradresse verwendet. Die Umrechnungsbefehle sind ALIAS-Definitionen von **SWAP** .

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

2.4.2 Adreß-Align

Ein Align der Adresse ist weder bei 16Bit- noch bei 32Bit-Zugriff notwendig. Die Befehle **ALIGN** , **ALIGNED** , **VALIGN** und **HALIGN** sind Alias-Definitionen von **NOOP** und als Immediate gekennzeichnet. Sie werden deshalb in :-Definitionen nicht kompiliert.

2.4.3 Länge der Datentypen

Der Z84C015 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 KKF-Versionen zuerst ein Längenbyte, dann die Zeichenkette und zum Schluß noch eine Null abgelegt. Nach Erhöhung der Adresse des Längenbytes um 1 kann der String sofort für alle Filebefehle herangezogen werden.

16Bit-Worte haben eine Länge von zwei Adressen. Es wird immer das niederwertige Byte zuerst abgelegt.

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

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

2.4.4 Portbefehle

Der Z84C015 verfügt über einen eigenen Portbereich mit 256 Adressen. Auf diesen Bereich kann mit den Befehlen **P@** und **P!** zugegriffen werden, wobei immer nur das niederwertig Byte

geschrieben oder geholt wird. Die untersten 32 Adressen werden schon durch interne Register des Prozessors belegt.

2.4.5 Filebefehle

Alle Filebefehle des KKF_8415 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 beachtet 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 Daten- und Returnstack

Für den Datenstack wird der vom Prozessor für die Ablage von Rücksprungadressen bei Unterprogrammaufrufe vorgesehene Stackzeiger SP verwendet. Dadurch können die relativ schnellen PUSH- und POP-Befehle zum Zugriff auf die Datenstackwerte verwendet werden.

Der Returnstack wird statt dessen mit dem 16Bit-Register IY nachgebildet. Analog zu SP wird dabei vor Ablage eines Wertes der Zeiger erniedrigt und dann das niederwertige Byte an der niedrigeren Adresse abgelegt.

Jeder Taskbereich hat zwei Speicherbereiche mit jeweils 512 Bytes, in dem bis zu 256 Werte abgelegt werden können. Die Endadresse des Bereiches ist in der USER-Variable **S0** bzw. **R0** gespeichert. Um bei Stackunterlauf keine anderen Daten zu überschreiben, sind über den Stacks noch ein Bereich mit 6 Bytes freigehalten. Dies genügt aber nur, um im Interpretermodus bei den meisten Befehlen vor Zerstörung wichtiger Daten eine Fehlermeldung zu erhalten. In eigenen Programmen sollte man Stackunterläufe durch **?STACK** abfangen.

2.4.7 Texte für Fehlermeldungen ändern

Beim KKF_8415 sind die Texte zu den Fehlermeldungen im EPROM untergebracht. Falls dort neue Texte übernommen werden sollen, so müssen mit einem geeigneten Monitorprogramm (oft beim EPROM-Programmierer vorhanden) folgende Schritte durchgeführt werden:

- Erstellen und Speicherung einer neuen Fehlermeldungs-Tabelle
  * Alte Dictionaryende merken (`$7B @ hex u.`)
  * Restlänge des Image merken (`$7F @ $100 + hex u.`)
  * Anfang der alten Fehlertabelle ermitteln (`>errortext @ hex u.`)
  * Mit CREATE ERRTEXT den Anfang einer Tabelle definieren
  * Fehlernummer und Texte mit **,** und **$,** kompilieren
  * Mit einem `0 ,` die Tabelle abschließen

```
   * Tabelle abspeichern ( Länge merken : here errtext -  hex u. )
        " NEWTEXT.BIN" 1+ #rw (file-create error
        dup  ds@ drop errtext  rot  over here swap -  swap (file-write error
        (file-close error
```

- EPROM verändern
  * Rest des Image (Länge siehe oben) an neue Adresse bringen
    Quellenadresse: Dictionaryende
    Zieladresse: Alte Fehlertabelle-Adresse + neue Tabellenlänge
    (Zieladresse wird noch für **DP** gebraucht)
  * Neue Fehlertabelle ab alter Fehlertabelle-Adresse laden
  * **DP** (Adresse $7B) auf Zieladresse des Imagerest setzen (Low-Byte zuerst)

Da das EPROM schon sehr voll ist, darf die neue Tabelle nur um maximal 15 Bytes länger als die bisher verwendete Liste sein. Falls aber doch ein größerer Speicherbereich zur Verfügung gestellt werden soll, so ist bei einem 32KByte-EPROM nur eine Verschiebung der Image-Anfangsadresse notwendig.

2.5 Zusatzhinweise

Ein Blick hinter die "Kulissen" des KKF-Systems soll in dem letzten Kapitel der Beschreibung des KKF_8415-Kerns die Initialisierungssequenz und die Ein-/Ausgabe verdeutlichen. Die Realisierung der Schnittstellenbefehle des KKF_8415 ist als Listing (Files P7_TERM.SCR) im Anhang E angegeben.

2.5.1 Initialisierung des Systems

Der KKF-Kern bleibt beim KKF_8415 im EPROM. Nur die Systemvariablen, der USER-Bereich, die (NEXT-Routine, die Interrupt-Vektoren und ein evtl. vorhandenes Zusatzimage wird wieder an die richtige Adresse im RAM kopiert. Der genaue Ablauf der Initialisierung geschieht wie folgt:

- Initialisierungen: DI und IM1 (keine Interrupts, Modus 1)
- Stack vor SYSVAR-Bereich setzen
- Suche ob ein Zusatzimage vorhanden ist
  * Ins RAM übertragen, wenn Zieladresse von Orginaladresse abweicht
  * wenn Image fehlt, dann RAM-Anfang für SYSVAR-Kopie verwenden
- Endadresse des Image ermitteln
- Adressen von UWORK, SWORK und FIRST ermitteln
- Alle Tasks kopieren und dabei die Adressen korrigieren
- Heap- und Variablenbereich kopieren
- (NEXT-Routine und Interrupt-Vektoren kopieren
- SYSVAR-Bereich noch in den RAM-Anfang kopieren
- FORTH mit Task0 starten
- Filevariablen und Zeilenpuffer löschen, **SINGLETASK**
- Schnittstellen initialisieren
- '**BOOT** , '**COLD** und dann **ABORT** aufrufen

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.

Initialisierung der Schnittstelle

Bei 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 wird der Timer 2 auf den im SYSVAR-Bereich angegebenen Teilerwert gesetzt und die Schnittstelle zurückgesetzt.

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

```
$01  sflag $0e -  !            ( Teilerangabe auf 1 setzen )
sflag @  $fff7 and  sflag !    ( Flag löschen )
(output                        ( SIO initialisieren )
```

Zeichenein-/ausgabe

Die Schnittstelle besitzt einen Puffer für ein Zeichen. Da ohne Handshake gearbeitet wird, führt der Empfang eines weiteres Zeichens zum Überschreiben des alten Wertes. Obwohl es möglich währe, wird keine Fehlermeldung erzeugt, wenn dies geschieht.

Beim Ausgeben eines Zeichens genügt das Einschreiben des Wertes in den Port der Schnittstelle. Es kann deshalb während des Senden eines Zeichens mit dem Programm fortgefahren werden.

Filebefehle

Wie schon in Kapitel 2.4.5 angesprochen, sind alle Filewörter im KKF_8415 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.

Kapitel 3

Zusatzprogramme

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

3.1 HD64180.SCR

Ausgangsbasis zu diesem Assembler war die Z80-Version von Heinz Schnitter. Vom Autor wurde eine übersichtlichere Syntax zur Erstellung der Tabellen entwickelt und die Liste um Befehle des Z80-Derivat HD64180 erweitert. Zusätzlich geben Kontollstrukturen die Möglichkeit zur Einbindung von Kurzsprüngen.

Der tabellengetriebene Assembler kennt alle gültigen HD64180-Befehle. Da für einen Befehl auch alle erlaubten Adressierungsarten gespeichert sind, kann es nicht mehr vorkommen, daß durch unglückliche Angaben Bitmuster erzeugt werden, die nicht erlaubt sind. Darüber hinaus erlauben diese Tabellen eine sehr einfache Erweiterung des Assemblers für andere Z80-Derivate.

Obwohl der Assembler alle HD64180-Befehle erkennt und außerdem noch die Stuktur-Erweiterung besitzt, hat er nur eine Länge von ca. 5KByte. Beim Kompilieren kann diese Größe aber erheblich überschritten werden, weil die Befehlsnamen erst am Ende wieder vergessen werden.

Aufbau der Assemblerbefehle

Wie bei FORTH üblich, ist auch dieser Assembler in Postfix-Notation. Es wird also zuerst der erste Operand, dann die Adressierungsart und vor dem Opcode evtl. noch ein zweiter Operand angegeben. Dabei wurde versucht, die Zilog-Mnemonics möglichst beizubehalten. Um dies zu erreichen, ist die Adressierungsart beider Operanden zusammengefaßt.

```
di,
hl inc,
hl ) inc,
hl ), a ld,
a , 5 ld,
a ,( $20 ld,
```

Hier auch zwei Warnungen bei Verwendung des Assemblers:

- Alle hexadezimale Zahlen sollten mit einer führenden Null oder mit Prefix angegeben werden, da die Register einfach mit dem entsprechenden Buchstaben angegeben werden.

- Gefährlich sind die Befehle **,** , **I** und **L** . Sie werden häufig verwendet (Ablegen von Werten; Aufrufen des Editors), haben aber im Assembler eine andere Bedeutung.

Befehle des Assemblers

Wie schon im Handbuch erwähnt, werden Codedefinitionen mit **CODE** oder **PROC** eingeleitet. Falls zu einer CREATE- oder VCREATE-Definition eine Assembler-Runtimeroutine erstellt werden soll, so muß diese Definition mit **;CODE** eingeleitet werden. Nach allen drei Befehlen befindet man sich im Interpretermodus und das Assembler-Vokabular wird als erstes durchsucht. Mit **END-CODE** bzw. **END-PROC** wird dann die Definition beendet. In den meisten Fällen muß man

durch Sprung zur (NEXT-Routine für eine Fortsetzung des FORTH-Programmes sorgen. Dazu sind die speziellen Makrobefehle **NEXT,** , **HPUSH,** , **DPUSH,** und **BPOP,** definiert. Da es sich um einfache Sprünge zu den entsprechenden Routinen am Speicherende handelt, dürfen auch noch Bedingungen mit nachfolgendem Komma angegeben werden.

CODE (name ; --)
Anfang einer Code-Definition. Bei der Verwendung des so definierten FORTH-Befehls wird direkt die nachfolgende Codesequenz ausgeführt. Der Befehl selbst ist erst nach Abschluß mit END-CODE ausführbar.

;CODE (-- ;C: 0 --)
Entspricht dem FORTH-Befehl **DOES>** , nur wird an Stelle einer High-Level-Definition die nachfolgende Assembler-Routine abgearbeitet. Beim Einsprung zeigt das Register DE auf CFA+1 des auszuführenden Befehls.

PROC (name ; --)
Anfang einer Definition, die sich wie eine CREATE-Definition verhält. Es wird also bei Verwendung des Namens die entsprechende Parameterfeldadresse auf dem Stack abgelegt. Andere CODE-Definitionen können deshalb diese Prozeduren einfach mit `<name> CALL,` als Unterprogramm aufrufen. Sie werden deshalb meisten mit einem **RET,** abgeschlossen.

END-CODE (--)
Abschluß der Code-Definition. Es muß vorher dafür gesorgt werden, daß der nächste FORTH-Befehl abgearbeitet werden kann. Dies geschieht üblicherweise durch Rücksprung in den Interpreter mittels **NEXT,** . Es wird durch **END-CODE** kein Code mehr erzeugt, sondern nur die lokalen Labels aufgelöst und der neu definierte Befehl verfügbar gemacht.

END-PROC (--)
Analog END-CODE.

NEXT, (--)
Ein Sprung auf die (NEXT-Routine am Speicherende wird kompiliert. Es können wie bei allen anderen JP,-Definitionen noch Bedingungen angegeben werden.

HPUSH, (--)
Vor der Ausführung der (NEXT-Routine wird noch der Wert des HL-Registers auf den Stack gelegt. Es können wie bei allen anderen JP,-Definitionen noch Bedingungen angegeben werden.

DPUSH, (--)
Vor der Ausführung von HPUSH wird noch der Wert des DE-Registers auf den Stack gelegt. Es können wie bei allen anderen JP,-Definitionen noch Bedingungen angegeben werden.

BPOP, (--)
Vor der Ausführung von DPUSH wird noch der oberste Stackeintrag zum BC-Register gebracht. Es können wie bei allen anderen JP,-Definitionen noch Bedingungen angegeben werden.

Z80-Register

Im Assembler sind alle Register des Prozessors verfügbar. Einige davon werden aber schon für das KK-FORTH verwendet und müssen deshalb gerettet werden. Nur wenn eine Index-Angabe folgt, ist an Stelle von **IX** (bzw. **IY**) der Befehl **IX+** (bzw. **IY+**) zu verwenden.

SP	FORTH-Datenstackzeiger
IY	FORTH-Returnstackzeiger
BC	FORTH-Befehlszeiger
HL	Arbeitsregister enthält Inhalt des aktuellen Codefeldes
DE	Enthält Codefeldadresse+1

A, H, L, D, E und IX sind frei verwendbar
Der zweite Registersatz ist ebenfalls frei verwendbar

Für den Assembler sind alle Register definiert:

A	B	C	D	E	F	H	L	I	R
BC	DE	HL	AF	AF'	SP	IX	IY		
IX+	IY+								

Bedingungen für Sprünge

Neben den Register und den Werten (Zahlen oder Adressen) können auch Sprungbedingungen als Operand angegegeben werden. Obwohl der gleiche Befehl **C** wieder vorkommt, entstehen keine Konflikte dadurch. Der Assembler erkennt automatisch die richtige Zuordnung.

NZ	Z	NC	C	PO	PE	P	M

Adressierungsarten:

Insgesamt gibt es 5 Adressierungsarten für den Z80-Assembler. Sie werden zwischen den beiden Operanden bzw. vor dem Opcode angegeben.

Art	Kommentar	Beispiel
	Immediate-Befehle	EI
	Direkt-Befehle	DE INC
)	Indirekt	HL) INC
),	Indirekt-Direkt	HL), A LD
,(	Direkt-Indirekt	A ,(IX+ 5 LD
,	Direkt-Direkt	A , L LD

Lokale Labels und Strukturen

Sehr selten werden Definitionen geschrieben, die ohne Sprünge auskommen. Meistens werden, je nach Bedingung, verschiedene Teile einer Definition abgearbeitet. Die Assemblerbefehle **CALL,** , **JR,** und **JP,** benötigen dazu die entsprechenden Adressen. Im HD64180.ASM werden dazu zwei Wege gegangen:

Lokale Labels:

Innerhalb einer CODE-Definition werden auch lokale Einsprungadressen festgelegt. Diese Adressen werden dann am Ende einer Definition korrigiert. Da dabei immer 16 Bit reserviert werden, können diese Referenzen nur in Befehle verwendet werden, die auch 16Bit-Werte speichern. Dazu zählen absolute Sprünge, Unterprogrammaufrufe und Lade-/Speicheroperationen.

Die Adressen werden mit den Befehlen bis **10$:** markiert und in Befehlen mit **1$** bis **10$** angegeben. Dabei können Labels auch schon verwendet werden, bevor die entsprechende Einsprungadresse markiert wird. Erst am Ende der Definition wird überprüft, ob alle Referenzen

aufgelöst wurden. Die Labels gelten nur innerhalb der CODE-Definition, können also danach nicht mehr verwendet werden.

Beispiel:
```
Code 16*   ( n -- n*16 )
      hl pop,           ( TOS in HL bringen )
      a , 4 ld,         ( Akku mit 4 laden )
  1$: l sla,            ( L-Register shiften )
      h rlc,            ( Übertrag in HL )
      a dec,            ( Akku erniedrigen )
      nz , 1$ jp,       ( insgesamt 4 mal )
      hpush,            ( Ergebnis zum Stack, dann Zurück )
 End-Code  ( Ende der Definition )
```

Als einzige Ausnahme für Verwendung des lokalen Labels innerhalb von Relativsprüngen kann der DJNZ,-Befehl angeführt werden. Da dabei die entsprechende Adresse schon vorher festgelegt wird, liefert n$ auch die tatsächliche Speicheradresse.

Strukturen

Für die Verwendung in relativen Vorwärtssprüngen sind lokale Labels nicht geeignet. In Anlehnung an normale FORTH-Strukturen werden deshalb im Assembler folgende Befehlsgruppen verwendet.

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

Diese Wörter kompilieren die entsprechenden Relativsprünge inklusive der richigen Adresse. Dabei erwarten **IF,** , **WHILE,** und **UNTIL,** eine Angabe, unter welcher Bedingung (**NZ** , **Z** , **NC** oder **C**) die entsprechende Routine nach dem Befehl ausgeführt werden soll. Im Code werden dann die umgekehrten Bedingung kompiliert (aus `C IF,` wird JR NC,Adresse). Auch hier wird die korrekte Strukturierung und eine Sprungweite unter 128 Bytes gefordert.

Beispiel:
```
Code $1234=   ( n -- f )
      hl pop,           ( TOS in HL bringen )
      de , $1234 ld,    ( DE=Vergleichswert)
      a xor,            ( C-Flag löschen )
      hl , de  sbc,     ( HL-DE setzt Z-Flag
      nz IF,            ( Kompiliert JR Z,... )
            hl , -1 ld,       ( True-Flag )
      ELSE,             ( Kompilier  JR ... )
            hl , 0 ld,        ( False-Flag )
      THEN,             ( Setzt Adressen ein )
      hpush,            ( Ergebnis zum Stack, dann Zurück )
 End-Code    ( Ende der Definition )
```

Beispiele

Ich erspare mir hier die Aufstellung der einzelnen Befehle. Wenn man sich das Programmlisting des Assemblers ansieht, dann fällt sowieso auf, daß diese Liste Befehl für Befehl alphabetisch sortiert eingegeben wurde.

Hier nun einige Beispiele, um sich einigermaßen mit der Assembler-Syntax vertraut zu machen. Auf der linken Seite wird die normale Assembler-Mnemonics und dann auf der rechten Seite die FORTH-Sequenz gegenübergestellt. VAR1 ist die Speicheradresse von Daten, die identisch mit einer FORTH-Variablenadresse sein könnte. An Stelle von Konstanten wurde einfach der hexadezimale Wert $1234 verwendet.

Z80-Assembler	FORTH-Assembler
EI	EI
RST 038h	$38 RST
DEC A	A DEC
DEC (HL)	HL) DEC
LD HL,VAR1	HL , VAR1 LD
LD (HL),VAR1	HL), VAR1 LD
LD HL,(VAR1)	HL ,(VAR1 LD
BIT 6,(IX+4)	6 ,(IX+ 4 BIT
CALL C,01234h	C , $1234 CALL

Verwendung von (USER-)Variablen

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

Beispiel: Zähler erhöhen

```
Variable co
Code co+  ( Zähler erhöhen )
      de , co vdp @ - ld,      ( Offset )
      hl ,( vdp ld,            ( Variablenanfang )
      hl , de add,             ( Zieladresse )
      hl ) inc,                ( Low-Byte erhöhen )
      nz , next,               ( Ende ? )
      hl inc,                  ( zum High-Byte )
      hl ) inc,                ( High-Byte erhöhen )
      next,                    ( zurück zum FORTH )
 End-Code  ( Ende der Definition )
```

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

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

3.2 Z80DIS.SCR

Der Disassembler ist dem CPM-volksFORTH entnommen und wurde von Ulrich Hoffmann entwickelt. Beim Laden von Z80DIS.SCR wird ein Vokabular mit Namen **TOOLS** angelegt. In diesem Vokabular stehen danach folgende Befehle zur Verfügung:

dis	(addr ; --)	Ab addr disassemblieren
ndis	(addr n --)	n Bytes ab addr disassemblieren
displace	(-- addr)	Variable enthält Offset für alle Zugriffe

Der Disassembler generiert immer 10 Zeilen und wartet dann auf eine Bestätigung. Mit CTRL+X kann die Ausgabe abgebrochen werden. Die ausgegebene Syntax entspricht dabei der ZILOG-Mnemonics.

3.3 Z80DEBUG.SCR

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

Der FORTH-Dekompilier

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

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

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

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

Der FORTH-Debugger

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

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

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

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 Z84C015.SCR

Dieses File enthält einige kleine Bespiele für den mc-Z80-MINI-EMUF. Obwohl CODE-Definitionen erzeugt werden müssen, kann auf das Laden des Assemblers verzichtet werden. Die benötigten Codes werden hier einfach mit **C,** und **,** kompiliert, was viele Assemblerspezialisten noch sehr bekannt sein dürfte. Jeder der nachfolgend beschriebenen Screens verfügt auch noch über zusätzliche Komentare.

Watchdog verwalten

Der Prozessor verfügt über einen Watchdog-Timer, der durch Schließen eines Jumpers aktiviert werden kann. Da eine Routine in **'BOOT** die Watchdog deaktiviert, ist dies auch beim Start des KKF_8415 möglich. Mit den auf Screen 2 aufgeführten Befehlen kann dann die Watchdog aktiviert, zurückgesetzt und wieder deaktiviert werden. Nach der Aktivierung mit **WD-ON** ist mindestens einmal pro Sekunde **WD-RESET** aufzurufen, weil sonst ein Reset erzeugt wird. Der Befehl **WD-OFF** schaltet die Watchdog wieder ab.

wd-on	(--)	Watchdog mit 2^22 Takte starten
wd-reset	(--)	Watchdog zurücksetzen
wd-off	(--)	Watchdog abschalten

Verwendung der Timer

Die CTC im Z84C015 verfügt über vier Zähler, die in verschiedenen Modes mit internen oder externen Takt arbeiten können. Die im Screen 3 definierten Befehle haben die Kanalnummer und einige Modes als Konstanten aufgeführt und ermöglichen die Initialisierung und das Auslesen des Zählers.

#ctc0	(-- $10)	Adresse des Zähler 0
#ctc1	(-- $11)	Adresse des Zähler 1
#ctc2	(-- $12)	Adresse des Zähler 2
#ctc3	(-- $13)	Adresse des Zähler 3
#ti/16-mode	(-- $07)	Timer-Mode mit Teilung durch 16
#ti/256-mode	(-- $27)	Timer-Mode mit Teilung durch 256
#iti/16-mode	(-- $87)	Interrupt-Mode mit Teilung durch 16
#iti/256-mode	(-- $A7)	Interrupt-Mode mit Teilung durch 256
#co-mode	(-- $47)	Externer Zähler
ctc!	(n mode port --)	Zähler initialisieren
ctc@	(port --)	Zähler auslesen

Bei **CTC!** muß neben dem Zählerwert noch der Modus und die Portadresse angegeben werden. Es wird beim Modus zwischen freilaufenden Timer, Interrupt-Timer und Zähler unterschieden. Bei den Timern kann noch angegeben werden, ob der interne Takt durch 16 oder durch 256 geteilt werden soll.

Interrupt-Zähler

Ein Beispiel für die Verwendung des Zählers und der Interrupt-Tabelle am RAM-Ende ist in den Screens 4 und 5 aufgelistet. Dabei werden die drei benötigten Code-Definitionen ohne Assembler erzeugt.

co	(-- addr)	Feste Speicheradresse des Zählers
co+	(-- addr)	Interrupt-Routine
ei	(--)	Interrupt erlauben
di	(--)	Interrupt sperren
inttest	(--)	Vollständiges Programm

Der Befehl **INTTEST** merkt sich den alten Interruptvektor und überschreibt ihn durch die Adresse der Routine **CO+** . Nach Initialisierung des Timer 0 wird noch die verwendete Speicheradresse auf 0 gesetzt und der Interrupt erlaubt. Bis eine Taste gedrückt wird, bleibt der Cursor in der gleichen Zeile und gibt immer wieder den Inhalt der Speicherzelle aus. Nach Abschaltung des Interrupts wird wieder der alte Vektor zurückgeschrieben.

Weil sich die Adresse einer Variable verändern würde, wird in diesem Beispiel der Zählerwert in den SYSVAR-Bereich gelegt. Dadurch ist die Veränderung des Variablen- und Heap-Bereich bei laufendem Interruptprogramm möglich.

Da bei höherer Baudrate die Ausgabe schneller als die Anzeige des Terminalprogramm sein kann, ist ein Überlauf des Puffers von 16384 Zeichen möglich. Es wird dann die Anzeige zerstört, aber sonst kein Fehler verursacht.

Portansteuerung

Als letztes soll noch die ebenfalls verfügbare PIO mit zwei 8Bit-Ports zu ihrem Recht kommen. Die im Screen 6 untergebrachten Konstanten und Befehle erlauben die Initialisierung, die Einstellung der Richtung (Eingabe oder Ausgabe) und das Lesen bzw. Schreiben des Ports.

#pio-a	(-- $1c)	Portadresse des Kanal A
#pio-b	(-- $1e)	Portadresse des Kanal B
dir!	(w port --)	Richtung setzen (1=Eingang)
pio@	(port -- w)	Port abfragen
pio!	(w port --)	Port setzen

Wenn man z.B. ein Led mit einem Wiederstand zwischen Port A-Bit 0 und Masse hängt, so kann dieses LED sehr einfach angesprochen werden.

```
0 #pio-a dir!                ( Port A auf Ausgang setzen )
1 #pio-a pio!                ( Bit 1 auf 1 = LED einschalten )
0 #pio-a pio!                ( Bit 1 auf 0 = LED ausschalten )
```

Bei Tests wurden auch schon Interruptprogramme mit der Ansteuerung des LED's betraut. Durch geeignete Tastverhältnisse kann dabei die Helligkeit des LED fast kontinuierlich heruntergesetzt werden.

3.6 RTCTEST.SCR

Es ist auch eine Echtzeituhr auf dem mc-Z80-MINI-EMUF vorgesehen. Die durch das File RTCTEST.SCR zur Verfügung gestellten Befehle erlauben das Auslesen und Setzen dieser Uhr. Dabei wird aber ein Datenformat verwendet, daß beim PC in der Filebehandlung üblich ist. Dieses Format gibt Datum und Uhrzeit in einem 32Bit-Wert mit einer Genauigkeit von zwei Sekunden an.

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

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

3.7 SIEVE.SCR

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

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

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

3.8 NECP6.SCR

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

Programmlisting

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

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

                ( Befehle ohne Copyright-Meldung )
                ( File ; -- )       6 Screens pro Seite (0-3; 1-4;2-5)
SLIST:          ( File ; -- )       3 Screens mit Shaddowscreens pro Seite
DLIST:          ( File ; -- )       Zwei A5-Seiten mit 4 Screens pro A4-Seite
```

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

Druckerbefehle

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

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

3.9 MEM_EDIT.SCR

Auf der zweiten Diskette ist ein eigenständiges Editorprogramm untergebracht, daß das gesamte File in den externen Speicher des PC's ladet und dort editiert. Zusätzlich erkennt dieses

Programm auch noch den Unterschied zwischen EGA- und Herkules-Karte und stellt sowohl die Speicheradresse als auch die Farbe der Ausgaben darauf ein.

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

```
KKF_PC  include mem_edit.scr
```

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

3.10 PC_TERM.SCR

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

```
KKF_PC  include pc_term.scr
```

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

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

Anhang A

Glossar-Zusatz

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

Bezeichnung der Stackparameter:

(C: ...)	Veränderungen auf dem Datenstack während des Kompilierens
(R: ...)	Veränderungen auf dem Returnstack
(name ; ...)	Es wird noch ein Befehlsname erwartet
flag	Flag (0=ff=Falsch; sonst tf=Wahr)
b	Byte
n	Einfachgenauer, vorzeichenbehafteter Wert
+n	Einfachgenauer, positiver Wert oder 0
u	Einfachgenauer, vorzeichenloser Wert
w	Nicht definierter, einfachgenauer Wert
addr	Adresse
sys	Systemabhängige Speicheradresse
lfa	Linkfeld-Adresse
nfa	Namensfeld-Adresse
cfa	Codefeld-Adresse
pfa	Parameterfeld-Adresse
csa	Counted-String-Adresse
d	Doppeltgenauer, vorzeichenbehafteter Wert
+d	Doppeltgenauer, positiver Wert oder 0
ud	Doppeltgenauer, vorzeichenloser Wert
wd	Nicht definierter, doppeltgenauer Wert
ptr	32Bit-Zeiger (bank: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_8415 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_8415 auf 79 Zeichen gesetzt, um kein Zeilenumbruch bei der Befehlseingabe zu verursachen.

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

'BOOT (--) D
Diese DEFER-Routine ist im KKF_8514 auf eine Routine umgeleitet, welche die Watchdog des Prozessors deaktiviert.

(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_8415 V1.2/0 nicht verfügbar.

(MFREE (ptr -- error)
Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

(MRELOC (ptr len. -- error)
Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

,A (cfa --) DX
Der angegebene 16Bit-Wert wird unverändert ins Dictionary übernommen.

,C (w --) DX
Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

?BRANCH (f --) 83 R

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

AT (x y --) UV

Dieser Befehl wird unter Verwendung von **COMMAND!** realisiert und kann deshalb nur mit einem KKF-Terminal verwendet werden. Die Statuszeilen 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 benötigt noch einen Inline-Offset, der zum aktuellen Programmzeiger addiert wird.

BYE (--) 83

Nach dem Schließen des noch offenen Files wird auf Adresse 0 gesprungen. Dort steht normalerweise ein Sprung auf die Boot-Routine, die dann die eigenliche Initialisierung durchführt. Bis auf die schon veränderten Prozessorregister (z.B. CTC und PIO) reagiert das KKF_8415 wie nach dem Einschalten.

CELL+ (addr1 -- addr2) ANSI

Beim KKF_8415 ist **CELL+** eine ALIAS-Definition von **2+** .

CELLS (n -- addr) ANSI

Beim KKF_8415 ist **CELLS** eine ALIAS-Definition von **2*** .

CHAR+ (addr1 -- addr2) ANSI

Beim KKF_8415 ist **CHAR+** eine ALIAS-Definition von **1+** .

CHARS (n -- len) ANSI I

Beim KKF_8415 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.

CS@ (-- ptr)

CS@ liefert im KKF_8415 einen Pointer auf den Anfang der aktuellen Codebank. Da der Prozessor Z84C015 nur über eine Bank verfügt, wird der Pointerwert 0. zurückgeliefert.

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)

DS@ liefert im KKF_8415 einen Pointer auf den Anfang der aktuellen Datenbank. Da der Prozessor Z84C015 nur über eine Bank verfügt, wird der Pointerwert 0. zurückgeliefert.

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 --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

L2! (dw ptr --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

L2@ (ptr -- dw)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

L@ (ptr -- w)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LC! (char ptr --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LC@ (ptr -- char)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LCMOVE (ptr1 ptr2 len --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LCMOVE> (ptr1 ptr2 len --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LDUMP (ptr len --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LFILL (ptr len char --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LIMIT (-- sys)

LIMIT ist im KKF_8415 auf $0000 (gesamter Speicher einschließlich Interrupt-Tabelle) gesetzt und kann nicht verändert werden.

LMOVE (ptr1 ptr2 len --)

Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

LWFILL (ptr count w --)
Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

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_8415 nicht verwendet.

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

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

P@ (addr -- w)
Es wird nur die Portadresse addr ausgelesen.

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

PC! (char addr --)
Dieser Befehl ist im KKF_8415 V1.2/0 nicht verfügbar.

PC@ (addr -- char)
Dieser Befehl ist im KKF_8415 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 zuerst der Returnstackzeiger um zwei erniedrigt und dann der Wert gespeichert (Low-Byte zuerst).

RP! (addr --) R

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

RP@ (-- addr)

RP@ liefert die aktuelle Wert des Returnstackzeigers.

S0 (-- addr) U

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

SFLAG (-- sys) SV

Folgende Bits von **SFLAG** werden im KKF_8415 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)

SS@ liefert im KKF_8415 einen Pointer auf den Anfang der aktuellen Datenbank. Da der Prozessor Z84C015 nur über eine Bank verfügt, wird der Pointerwert 0. zurückgeliefert.

SYSCON (-- $0000)

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

SYSVAR (-- $FF8B)

Der SYSVAR-Bereich befindet sich am RAM-Ende.

SYSVARLEN@ (-- $50)

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

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

IO_TERM.SCR

```
         Screen # 0
 0 \\ Interface-Routinen für die 84C015-SIO1      ( 09.08.91/KK )
 1
 2 System:         KKF_8415 V1.2/0
 3 Änderung:       09.08.91  KK: Baud-Teiler aus Kern (Norm:4)
 4
 5                     Hinweise:
 6  - Verwendet immer SIO1 des 84C015 mit Parameter  8,n,1
 7  - Teiler wird aus SYSVAR-Bereich entnommen:
 8     1: 38400 Baud ;  2: 19200 Baud ;  4: 9600 Baud
 9  - Fileinterface und Zusatzbefehle mittels ESC-Code
10
11
12
13
14
15

         Screen # 1
 0 \ Loadscreen                                   ( 02.06.91/KK )
 1
 2   &02 load          \ BDOS-Aufruf (im EMUF nicht verwendet)
 3   &03 &09 thru      \ Ein-/Ausgabe
 4   &10 &15 thru      \ Fileinterface
 5   &16 &17 thru      \ Vektorisierung und Initialisierung
 6
 7
 8
 9
10
11
12
13
14
15

         Screen # 2
 0 \ sio-init                                     ( 09.08.91/KK )
 1 | Code sio-init   ( -- ) ( SIO auf 9600 Baud einstellen )
 2      a ,( sflag ld,  3 , a bit,  nz , next, ( initialisiert ? )
 3      8 or,  sflag ), a ld,                     ( Flag setzen )
 4      a , $45 ld,  $12 ), a out,    ( Takt/256/Teilerkonstante )
 5      a ,( p_baud ld,  $12 ), a out,         ( Teilerkonstante )
 6      a , $01 ld,  $1b ), a out,  a , $00 ld,  $1b ), a out,
 7      a , $03 ld,  $1b ), a out,  a , $c1 ld,  $1b ), a out,
 8      a , $04 ld,  $1b ), a out,  a , $4c ld,  $1b ), a out,
 9      a , $05 ld,  $1b ), a out,  a , $68 ld,  $1b ), a out,
10      next,
11     End-Code
12
13
14
15
```

```
         Screen # 3
 0 \ ((emit? ((emit ((type ((cr ((del ((bell       ( 08.08.91/KK )
 1 | Code ((emit?  ( -- f ) ( Test des Output-Ports )
 2     hl , -1 ld,  a ,( $1b in,  4 and,  nz , hpush,
 3     hl inc,  hpush,                              End-Code
 4 | Code ((emit   ( char -- ) ( nur an Konsole )
 5     BEGIN,  a ,( $1b in,  4 and,  nz UNTIL,
 6     hl pop,  a , l ld,  $1a ), a out,  next,     End-Code
 7 | Code ((type   ( addr n -- ) ( nur an Konsole )
 8     de pop,  hl pop,  de inc,
 9     1$: de dec,  a , e ld,  d or,  z , next,
10         BEGIN,  a ,( $1b in,  4 and,  nz UNTIL,
11         a ,( hl ld,  $1a ), a out,  hl inc,  1$ jp,    End-Code
12 | : ((cr        ( -- )  #cr emit   $0a emit ;
13 | : ((del       ( -- )  #delout emit  #bl emit  #delout emit ;
14 | : ((bell      ( -- )  7 emit ;
15

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

         Screen # 5
 0 \ ((key?  ((key  ((string                       ( 03.06.91/KK )
 1 | Code ((key?   ( -- f ) ( Ausgabestatus holen )
 2     hl , -1 ld,  a ,( $1b in,  1 and,  nz , hpush,
 3     hl inc,  hpush,
 4    End-Code
 5 | Code ((key    ( -- char ) ( Zeichen von Konsole )
 6     BEGIN,  a ,( $1b in,  1 and,  nz UNTIL,
 7     a ,( $1a in,  l , a ld,  h , 0 ld,  hpush,
 8    End-Code
 9 | Code ((string   ( addr len -- )
10     de pop,  hl pop,  de inc,
11     1$: de dec,  a , e ld,  d or,  z , next,
12         BEGIN,  a ,( $1b in,  1 and,  nz UNTIL,
13         a ,( $1a in,  hl ), a ld,  hl inc,  1$ jp,
14    End-Code
15
```

```
         Screen # 6
 0 \ ((editstring-Zusätze                          ( 20.08.91/KK )
 1 | : del's       0 ?DO #delout emit LOOP ;
 2 | : cur-left    1 del's  >r  $1.0001 d-  r> ;
 3 | : cur-right   >r >r  dup 1 -type  1+  r> 1+  r> ;
 4 | : del-line    over del's  dup spaces  del's  -  0. ;
 5 | : del-char    ( addr pos max -- addr pos max-1 )
 6     1-  dup >r  over >r  swap -  >r  dup 1+ over r@ cmove
 7     dup r@ -type space  r> 1+ del's  r> r> ;
 8 | : ins-char    ( addr pos max char -- addr+1 pos+1 max+1 )
 9     -rot  2dup 2>r  swap -  >r  over dup 1+ r@ cmove> over c!
10     dup r@ 1+ -type  r> del's  1+ 2r> $1.0001 d+ ;
11 | : saveline    ( addr pos max -- addr pos max )
12     swork  dup p_llen @ +  p_llines @ 1- p_llen @ *  cmove>
13     dup swork 1+ c! >r  dup swork c!
14     2dup -  swork 2+  r@ p_llen @ umin  move  r> ;
15

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

         Screen # 8
 0 \ ((editstring  2. Teil                         ( 08.08.91/KK )
 1                  dup r@ -type  r> swap  r@ +  r> rot
 2                  2dup swap -  del's   r>  getkey  dup #esc <>
 3           UNTIL nip
 4        THEN
 5        CASE #cr    OF  sflag @ 4 and 0= IF saveline THEN
 6                        rot drop exit                ENDOF
 7             $13    OF over    IF cur-left THEN       ENDOF
 8             $04    OF 2dup u< IF cur-right THEN      ENDOF
 9             #brk   OF del-line                       ENDOF
10             #delin OF over IF cur-left del-char THEN ENDOF
11             $07    OF 2dup <> IF del-char THEN       ENDOF
12                    over span @ u<  over $1f u>  and
13                    IF dup >r ins-char r> THEN
14        ENDCASE
15     REPEAT ; -2 allot
```

```
         Screen # 9
 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 # 10
 0 \ w>com ... com>$                                ( 02.06.91/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 # 11
 0 \ command!                                       ( 08.08.91/KK )
 1 | : key_err?    ( com -- com | error mit EXIT )
 2     key?  IF  drop err_command!  rdrop  exit  THEN ;
 3
 4   : command!    ( com -- error ) ( Befehl senden )
 5     key_err?  #esc emit  key_err?      ( ESC-Befehl schicken )
 6     w>com  com>w ;
 7
 8 \ Bei COMMAND!-Error ein oder zwei Parameter verwerfen
 9 | : 1com!       command!  ?dup IF nip rdrop exit THEN ;
10 | : 2com!       command!  ?dup IF nip nip rdrop exit THEN ;
11
12
13
14
15
```

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

         Screen # 14
 0 \ Diskverwaltung: ((file-read ... ((file-free   ( 03.06.91/KK )
 1 | : ((file-read         ( seg addr len id -- error )
 2     $0209 command!  ?dup IF  nip nip nip nip  exit  THEN
 3     w>com  dup w>com   com>w ?dup  IF  nip nip nip  exit  THEN
 4     string  drop  0 ;             ( Für Schnelle Übertragung )
 5
 6 | : ((file-write        ( seg addr len id -- error )
 7     $020a command!  ?dup IF  nip nip nip nip  exit  THEN
 8     w>com  dup w>com   type   drop
 9     com>w ;
10
11 | : ((file-free         ( dev -- free. max. 0 | error )
12     $020d 1com!   w>com   com>w ?dup ?exit
13     com>d  com>d  0 ;
14
15
```

```
         Screen # 15
 0 \ ((file-first  ((file-next                      ( 01.06.91/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 # 16
 0 \ standard-i/o's                                 ( 03.06.91/KK )
 1   &10 output UTable: (output
 2     sio-init  ((emit?  ((emit  ((type   ((cr  ((del
 3               ((bell   ((cls   ((maxat  ((at  ((at? ;
 4
 5   &05 input  UTable: (input
 6     sio-init  ((key?  ((key  ((string  ((editstring  ((query ;
 7
 8   &13 disc   UTable: (disc
 9     noop      ((file?       ((file-create  ((file-delete
10               ((file-open   ((file-close
11               ((file-size   ((file-pos!    ((file-pos@
12               ((file-read   ((file-write   ((file-free
13               ((file-first  ((file-next ;
14
15


         Screen # 17
 0 \ standard-io  v  l                              ( 03.06.91/KK )
 1   : standard-io     ( -- )
 2     p_cinput  @ 6 - execute
 3     p_coutput @ 6 - execute
 4     p_cdisc   @ 6 - execute ;
 5
 6 ( Editor des Terminalprogrammes aufrufen )
 7 | : ((ed        ( scr offset -- )
 8    file-fcb @  >r  close   $0001 command!
 9    IF   2drop
10    ELSE swap w>com w>com  r@ 1+ $>com  com>w drop
11    THEN r> (open ;
12   : v           ( -- )          scr @  r# @   ((ed ;
13   : l           ( scr -- )              0      ((ed ;
14
15
```

Index