UDOS-Software
Dienstprogramme

EAW *electronic*

P8000

transcribed in 10h by O. Lehmann during 2006-06-13 and 2008-07-20
Version 1.0 (2008-07-20)

U D O S - S o f t w a r e

Dienstprogramme

Diese Dokumentation wurde von einem Kollektiv des Kombinates

VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"

erarbeitet.

Herausgeber:
Kombinat
VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Hoffmannstrasse 15-26
BERLIN
1193

WAE/03-0101-02

Ausgabe: 05/87

Die vorliegende Dokumentation unterliegt nicht dem Aenderungsdienst.

Spezielle Hinweise zum aktuellen Stand der Softwarepakete befinden sich in README-Dateien auf den entsprechenden Vertriebsdisketten.

Dieser Band enthaelt folgende Unterlagen:

- Teil 1: EDIT
  Texteditor
  (Bearbeiter: R. Kuehle)

- Teil 2: FILE.DEBUG
  Diskettenmonitor
  (Bearbeiter: J. Zabel)

- Teil 3: UPROG
  EPROM-Programmer
  (Bearbeiter: L. Mielenz)

- Teil 4: UFORM
  Programm zur Textformatierung
  (Bearbeiter: J. Kubisch)

- Teil 5: RABUG
  Symbolisches Fehlersuchprogramm
  (Bearbeiter: J. Kubisch)

- Teil 6: DISKTEST
  Diskettentestprogramm
  (Bearbeiter: J. Zabel)

- Teil 7: SI
  Treiber zur seriellen Datenuebertragung
  (Bearbeiter: J. Zabel)

Gesamtinhaltsverzeichnis Seite

Teil 1: EDIT

Teil 2: FILE.DEBUG

Teil 3: UPROG

Teil 4: UFORM

Teil 1

E D I T

Texteditor

Inhaltsverzeichnis Seite

1. Einleitung

Das UDOS-Dienstprogramm TEXT EDITOR dient der Erstellung und Modifizierung von Anwnederdateien. Es besitzt folgende Merkmale:

- zeilenorientiertes Editieren (Zeilenzaehler enthaelt immer die Nummer der Zeile, auf die zuletzt zugegriffen wurde).
- automatischer Datentransport von und zur Diskette
- Behandlung von Zeichenketten innerhalb einer Zeile
- Bearbeitung und Erzeugung von Dateien des Typs A (ASCII)
- Behandlung von Zeilen mit einer max. Satzlaenge von 512 Zeichen

2. Programmaufruf

Der Editor wird ueber folgendes UDOS-Kommando aufgerufen:

%EDIT dateiname.S [optionen]

. dateiname	ist eine Standard UDOS-Datei	
. .S	ist ein Namenszusatz und kennzeichnet die Datei als Quelldatei	
. optionen	O=dateiname	spezifiziert einen Namen fuer die Rueckkehrkopie
	N	ohne Rueckkehrkopie
	RL=Satzlaenge	spezifiziert die Satzlaenge einer neu zu erstellenden Datei (max. 200H)

3. Meldungen des Editors

Fehlt beim Aufruf des Editors die Namensangabe, so erfolgt die Ausschrift:

NAME?

Der Nutzer sollte in diesem Fall einen Namen angeben oder ueber "RETURN" die Steuerung an das UDOS zurueckgeben.

Ist die spezifizierte Datei nicht vom Typ A, so wird die Nachricht

INVALID ATTRIBUTES:filename

auf die Konsole ausgegeben. Die Ausgabe der Nachricht kann auch beim GEt-Kommando erfolgen, wenn die einzufuegende Datei nicht vom Typ A ist oder eine ungueltige Satzlaenge aufweist.
Existiert die angesprochene Datei noch nicht, so geht der Editor automatisch zum Neuerfassen der Datei in den Input-

Mode ueber.

```
NEW FILE
INPUT
```

Existiert die angesprochene Datei bereits, so wird sie in den Arbeitsspeicher geladen und steht zum Editieren bereit.

```
EDIT
```

Eine nicht ordnungsgemaesse Durchfuehrung einer Disketten-Ein-/Ausgabeaktivitaet initiiert die Fehlerausschrift

```
I/O ERROR xx ON UNIT yy
```

xx kennzeichnet die Fehlerart; yy gibt die logische Geraetenummer an (04 fuer die zu editierende Datei; 05 fuer die Rueckkehrdatei)
Tritt beim Editiervorgang diese Fehlernachricht auf, so gibt es verschiedene Moeglichkeiten, diesen Fehler zu beseitigen.
Eine Moeglichkeit besteht darin, die Datei zu loeschen und die Rueckkehrkopie umzubenennen.

Beispiel:

```
%EDIT MYFILE
EDIT
>    :
     :
     :
>QUIT
%DELETE MYFILE
DELETE 2/MYFILE (Y/N/A/Q)?Y
%RENAME MYFILE.OLD MYFILE
%
```

Eine andere Moeglichkeit besteht darin, waehrend des Editierens ueber das DElete-Kommando die gesamte Datei zu loeschen und anschliessend ueber das GEt-Kommando die Rueckkehrkopie einzufuegen.

Beispiel:

```
%EDIT MYFILE
EDIT
>    :
     :
     :
>T
T>DE *
>GE MYFILE.OLD
>    :
     :
     :
>QUIT
%
```

4. Erfassen von Dateien

Mit dem Programmaufruf: EDIT dateiname wird auf der Diskette gesucht, ob eine Datei mit dem angegebenen Namen vorhanden ist. Wird die Datei nicht gefunden, steht der Editor im Eingabe-Mode, und es kann mit der Programmerfassung begonnen werden.

```
%EDIT MYFILE RL=200 ;MYFILE existiert noch nicht
                     und wird mit einer Satzlaenge
                     200H eroeffnet
NEW FILE
INPUT
     :
     :
     :
EDIT
>QUIT
%
```

5. Aendern von Dateien

Ist beim Programmaufruf: EDIT dateiname die Datei mit dem angegebenden Namen bereits vorhanden, wird sie als Rueckkehrdatei kopiert und in den Speicher zum Editieren geladen. Der Rueckkehrdatei wird der Dateiname der Originaldatei mit dem Zusatz ".OLD" zugeordnet.

Beispiel: Editieren einer bereits existierenden Datei mit der Standardzuweisung fuer die Rueckkehrdatei:

```
%EDIT MYFILE   ;MYFILE existiert bereits
EDIT           ;Standardzuweisung   fuer   die
                Rueckkehrdatei MYFILE.OLD
>    :
     :
     :
>QUIT
%
```

Beispiel: Editieren einer bereits existierenden Datei mit der Zuweisung fuer die Rueckkehrdatei ueber die O-Option.

```
%EDIT MYFILE O=2/BACKUP  ;MYFILE existiert
EDIT                     ;Zuweisung   fuer   die
>    :                    Rueckkehrdatei  BACKUP
     :                    Laufwerk 2
     :
>QUIT
%
```

6. Kommandos des Editors

Der Editor umfasst 24 Kommandos:

Again	Find	LIneno	PUt	Up
Bottom	GEt	Locate	PUTD	Verify
BRief	Goto	Macro	QUIT	Window
Change	Input	Next	Replace	Xecute
DElete	Join	Print	Top	

Das Aendern der Datei erfolgt zeilenweise mittels Editor-Kommando. Das Einfuegen muss im Imput-Mode geschehen.

Die Kommandos des Editors koennen in 2 Formen zusaetzlich modifiziert werden.
Die erste Form besteht darin, durch eine Zahl n, eine Wiederholung des Kommandos n-mal zu erreichen (Voreinstellung von n ist 1). Zum Beispiel Print 15 wuerde 15 Zeilen auf die Konsole ausgegeben. Das Zeichen "*" bewirkt eine Ausgabe der Zeilen auf die Konsole von der aktuellen Zeile bis zum Dateiende.
Die zweite Form bewirkt, dass durch eine Zeichenkette zwischen Begrenzungszeichen, eine Wiederholung des Kommandos bis die Zeichenkette gefunden wird, erfolgt. Zum Beispiel Print /LD A,B druckt ab aktueller Zeile bis einschliesslich der Zeile mit: ,LD A,B. Wird die spezifizierte Zeichenkette nicht gefnden im aktuellen Block, so meldete sich der Editor mit der Ausschrift:

```
STRING NOT IN BLOCCK
PROCEED?
```

Die Abfrage, ob im nachfolgenden Block weitergesucht werden soll, wird mit Y (ja) oder N (nein) beantwortet.

Das Modifikationszeichen "n" bzw. "zeichenkette" koennen bei folgenden Kommandos zur Anwendung kommen und sind durch mindestens ein Leerzeichen vom Kommandowort zu trennen:

DElete, Next, Print, PUt, PUTD, Up

Folgende Notationsformen werden bei den nachfolgenden Kommandobeshreibungen benutzt.

- wahlweise verwendete Kommandoteile werden in Klammern [] gesetzt
- Das Symbol | wird fuer ein logisches Oder verwendet; z.B. DE [n|zeichenkette] bedeutet entweder DE n oder DE /zeichenkette/
- Die Begrenzungszeichen fuer eine Zeichenkette sind /zeichenkette/
- Das Zeichen "'" kennzeichnet eine mehrfachee Benutzung des Kommandos
- Fuer den Aufruf eines Kommandos muessen mindestens die unterstrichenen Buchstaben angegeben werden.
- Die Kommandos koennen in grossen oder kleinen Buchstaben geschrieben werden

7. Kommandodefinition

7.1. Again [n]

Funktion:

Wiederholung des vorhergehenden Kommandos n-mal

Beispiel:

```
>P 3       ;Ausgabe 3 Zeilen
zeile 1
zeile 2
zeile 3
>A         ;Ausgabe der nachfolgenden
zeile 4     3 Zeilen
zeile 5
zeile 6
>J&U 4&P 4&
zeile 2
zeile 2
zeile 3
zeile 4
zeile 5
>A         ;Wiederholung des letzten
zeile 5     Kommandos (Print 4)
zeile 6
zeile 7
zeile 8
```

7.2. Bottom

Funktion:

Der Zeilenzaehler wird auf die letzte Zeile der Datei gesetzt und ausgegeben.

Beispiel:

```
>B
letzte zeile
>N
EOF
>
```

7.3. BRief

Funktion:

Die urspruenglich vorgesehene Textausgabe nach Bottom, Change, GEt, Locate, Next und Up erfolgt nicht (Brief-Mode)

Beispiel:

```
>P                    ;Ausgabe einer Zeile
DAS IST EINE ZEHN
>C /ZEHN/ELF
DAS IST EINE ELF      ;Ausgabe nach Change
>BR                    (Verbose Mode)
>C /ELF/ZEHN          ;keine Ausgabe nach
>P                     Change (Brief Mode)
DAS IST EINE ZEHN     ;Ausgabe der Zeile
>
```

7.4. Change /alte zeichenkette/neue zeichenkette[|n1 n2]

Funktion:

Ersetzt die "alte zeichenkette" durch die "neue zeichenkette".
n1 gibt die Anzahl der Zeilen an, in denen die "alte zeichenkette" gesucht und geaendert wird, Wird fuer n1 "*" angegeben, so erfolgt die Aenderung in allen folgenden Zeilen.
n2 gibt die Anzahl der in einer Zeile vorkommenden "alten zeichenketten" an, die geaendert werden sollen. Wird anstelle von n2 "*" angegeben, erfolgt die Aenderung aller in einer Zeile vorkommenden "alten zeichenketten".
Sollte die Aenderung auch noch in allen vorhergehenden Zeilen durchgefuehrt werden, muss vor dem Change-Kommando ein Top-Kommando stehen. Die Angabe von n1 und n2 sind optional. Der Standardwert betraegt 1.
Soll eine angegebene Zeichenfolge geloescht werden, so erfolgt das mit:

```
Change /zeichenkette//
```

Wird der spezifizierte Text zur Aenderung nicht gefunden, wird

```
NO CHANGE
```

ausgegeben.
Die Kommandoabarbeitung kann mit '?' unterbrochen werden.

Beispiel:

```
>P                    ;Ausgabe einer Zeile
ADD A,B
>C /A,B/HL,DE/        ;Ersetzen von 'A,B'
>ADD HL,DE             in 'HL,DE'
>T                    ;Zeile 0
T>C /IX/IY/* *        ;Ersetzen aller in der Datei
                       vorkommenden 'IX' in 'IY'
>
```

7.5. DElete [n|/zeichenkette[/]]

Funktion:

Loeschen von Zeilen der Datei, beginnend mit der aktuellen Zeile.
Ist n spezifiziert, so werden n Zeilen geloescht.
Bei Angabe einer Zeichenkette werden alle Zeilen bis, aber nicht inklusive, zur Zeile mit der angefuehrten Zeichenkette geloescht.

Beispiel:

```
>P 5                ;Ausgabe 5 Zeilen
zeile 1
zeile 2
zeile 3
zeile 4
zeile 5
>U 4
zeile 1
>DE 2               ;Loeschen 2 Zeilen
>P
zeile 3
>
```

7.6. Find /zeichenkette[/]

Funktion:

Dient der Einstellung des Zeilenzaehlers auf die lt. "zeichenkette" angefuehrten Zeile. Dieses Kommando wird verwendet, um Namen im Programm aufzufinden.

Beispiel:

```
>P 5                ;Ausgabe 5 Zeilen
LD A,B
ADD A,C
JR Z,LOOP1
INC A
LOOP1: LD D,A
>U 4
LD A,B
>F /LOOP1:/         ;Finden "LOOP1:"
LOOP1: LD D,A
```

7.7. GEt [dateiname]

Funktion:

Laedt eine auf der Diskette befindliche Datei in den Arbeitsspeicher, beginnend mit der naechstfolgenden Zeile. Ist kein "dateiname" angegeben, wird die durch ein PUt- oder PUTD-Kommando erzeugte temporaere Datei eingefuegt. Die einzufuegende Datei muss vom Typ A (ASCII) sein und eine Satzlaenge kleiner oder gleich 200H haben.
Ist das nicht der Fall, wird die Nachricht

```
INVALID ATTRIBUTES:dateiname
```

ausgegeben.

Beispiel:

```
>P 4                ;Ausgabe 4 Zeilen
zeile 1
zeile 2
zeile 3
zeile 4
>U2
zeile 2
> PUT 2
>P
zeile 4
>B
zeile 4
>GET                ;Einlesen temporaere Datei
zeile 3
>T
T>P *
zeile 1
zeile 2
zeile 3
zeile 4
zeile 2
zeile 3
EOF
>
```

7.8. Goto n

Funktion:

Stellt den Zeilenzaehler auf die Zeilennummer n ein.

Beispiel:

```
>P 3                ;Ausgabe 3 Zeilen
zeile 1
zeile 2
zeile 3
>T
T>G 3               ;Zeilenzaehler Zeile 3
zeile 3
>
```

7.9. Input [textzeile]

Funktion:

Die angegebene Textzeile wird im Anschluss an die laufende Zeile eingefuegt. Ohne Texteingabe geht der Editor in den Input-Mode, wo beliebig viele Textzeilen nach der laufenden Zeile eingefuegt werden koennen. Der Ruecksprung in den Editor-Mode erfolgt mit "RETURN" in die neu einzufuegende Zeile.

Beispiel:

```
>P                      ;Ausgabe 1 Zeile
zeile 1
>I line 1A              ;Einfuegen 'line 1A' in die
                         Datei
P
line 1A
>I
INPUT                   ;Input-Mode
line 1B
line 1C
line 1D
                        ;RETURN
EDIT
>T
T>P *                   ;Ausgabe aller Zeilen
line 1A
line 1B
line 1C
line 1D
EOF
>
```

7.10. Join &command&[command&]

Funktion:

Die aufgefuehrten Kommandos (max. 512 Zeichen) werden mit "RETURN" sofort ausgefuehrt. Zwischen den Kommandos und Begrenzungszeichen sind Leerzeichen nicht erlaubt. Als Begrenzungszeichen kann jedes beliebige Zeichen verwendet werden, es darf jedoch nicht in den Kommandos selbst auftreten.

Beispiel:

```
>J #T#L /LOOP1#C /1/2#U 3#P 5
T                               ;Top-Kommando
LOOP1:                          ;Locate-Kommando
LOOP2:                          ;Change-Kommando
LD A,B                          ;Up-Kommando
LD A,B                          ;Print-Kommando
DEC B
JR Z;LOOP3
LOOP2:
DEC BC
```

7.11. LIneno

Funktion:

Gibt die Zeilennummer der laufenden Zeile aus.

Beispiel:

```
>P
zeile 10
>LI
10                          ;Ausgabe Zeilennummer
>
```

7.12. Locate

Funktion:

Stellt den Wert des Zeilenzaehlers auf die Zeile ein, die der Zeile mit der angegebenen Zeichenkette folgt.

Beispiel:

```
>L /200                     ;Locate-Funktion
LD HL,200H
>A                          ;Wiederholung des Kom-
STRING NOT IN BLOCK          mandos
PROCEED?Y
LD DE,200H
>
```

7.13. Macro &command&[command&]

Funktion:

Die aufgefuehrten Kommandos (max. 512 Zeichen) werden mit "RETURN" im Makropuffer abgespeichert. Zwischen den Kommandos und Begrenzungszeichen sind Leerzeichen nicht erlaubt. Als Begrenzungszeichen kann jedes beliebige Zeichen verwendet werden, es darf jedoch nicht in den Kommandos selbst auftreten.
Mit dem Kommando "Xecute" ist die abgespeicherte Kommandofolge jederzeit abarbeitbar.

Beispiel:

```
>M &T&L /A,B/&C /A,B/A,C/     ;Laden Makropuffer
>P 5                          ;Ausgabe 5 Zeilen
LOOP:
LD A,(HL)
ADD A,B
INC HL
DEC B
> U 10
JR NZ,LOOP
>X                            ;Abarbeitung
                               Kommandos
T                             ;Top-Kommando
ADD A,B                       ;Locate-Kommando
ADD A,C                       ;Change-Kommando
>
```

7.14. Next [n|/zeichenkette[/]]

Funktion:

Der Zeilenzaehler wird um n Zeilen erhoeht bzw. auf die Zeile mit der entsprechenden Zeichenkette eingestellt.

Beispiel:

```
>P                              ;Ausgabe 1 Zeile
zeile 3
>N 5
zeile 8                         ;5 Zeilen weiter
>T
>T P2
zeile 0
zeile 1
>N /3/                          ;Zeilenzaehler auf
LD A,3                           die Zeile die eine
>                                '3' enthaelt
```

7.15. Print [n|/zeichenkette[/]]

Funktion:

Beginnend ab der laufenden Zeile werden die naechsten n Zeilen oder bis zum ersten Auftreten der angegebenen Zeichenkette ausgegeben. Mit "?" kann die Ausgabe unterbrochen werden.

Beispiel:

```
>P 3                ;Ausgabe 3 Zeilen
zeile 1
zeile 2
zeile3
>P /7/              ;Ausgabe bis zum Zeichen '7'
zeile 4
zeile 5
LD B,7
>
```

7.16. PUt [n|/zeichenkette[/[dateiname [RL=m]]]]

Funktion:

Ausgabe auf einer Diskettendatei mit entsprechenden Dateinamen. n spezifiziert die Anzahl der Zeilen, waehrend bei Angabe einer Zeichenkette, bis zu dieser (aber nicht einschliesslich dieser) Zeile ausgegeben wird. Ueber RL kann eine Satzlaenge bis max. 200H vereinbart werden.
Ohne Namensangabe wird die temporaere PUT/GET-Datei verwendet, vereits vorhandener Text wird dabei ueberschrieben.

Beispiel:

```
>PU 3                  ;Ausgabe 3 Zeilen in die
                        temporaere PUT/GET-Datei
>PU /A,B/              ;Ausgabe aller Zeilen bis
                        zur Zeichenkette in die
                        temporaere PUT/GET-Datei
>PU 6 PUT.FILE RL=200 ;Ausgabe 6 Zeilen in die
                        Datei PUT.File
```

7.17. PUTD [n|/zeichenkette[/[dateiname]]]

Funktion:

Entspricht dem Kommando PUt, jedoch werden die ueberspielten Daten in der Anwenderdatei geloescht.

Beispiel:

```
>T
T>P 5                  ;Ausgabe 5 Zeilen
zeile 1
zeile 2
zeile 3
zeile 4
zeile 5
>T
T>PUTD 3               ;Ausgabe 3 Zeilen in tempo-
                        raeren PUT/GET-Datei und
                        loeschen der 3 Zeilen
>B
zeile 5
>GE                    ;Einlesen der temporaeren
zeile 3                 Datei
>T
T>P *
zeile 4
zeile 5
zeile 1
zeile 2
zeile 3
EOF
>
```

7.18. QUIT

Funktion:

Schliessen der Anwenderdatei, wobei die Steuerung an das OS zurueckgegeben wird.

Beispiel:

```
%EDIT MYFILE
EDIT
>    :
     :
     :
>QUIT                    ;Abschliessen der Datei
%                         und zurueck in OS
```

7.19. Replace [textzeile]

Funktion:

Der angegebene Text ueberschreibt den Text in der laufenden Zeile, wobei der Abstand vom Kommando genau ein Leerzeichen etragen muss.
Ist keine 'textzeile' angegeben, geht der Editor in den Input-Mode. Der ueber die Tastatur eingegebene Text wird eingefuegt, wobei die laufende Zeile mit dem alten Text geloescht wird. Ueber "RETURN" wird der Input-Mode verlassen.

Beispiel:

```
>P 2                 ;Ausgabe 2 Zeilen
zeile 4
zeile 5
>r line 4A           ;Ueberschreiben mit
                      'line 4A'
>
zeile 4
>P 2
zeile 4
line 4A
>R                   ;Input-Mode
INPUT
line 4AA
line 4AB
line 4AC
>U 4
zeile 3
P5
zeile 3
zeile 4
line 4AA
line 4AB
line 4AC
>
```

7.20. Top

Funktion:

Setzt den Zeilenzaehler auf die Zeile 0, d.h. vor die erste Zeile der Anwenderdatei.

Beispiel:

```
>P 3                        ;Ausgabe 3 Zeilen
zeile 1
zeile 2
zeile 3
>T
T>I line 0                  ;Einfuegen 'line 0'
>P 4
line 0
zeile 1
zeile 2
zeile 3
```

7.21. Up [n|/zeichenkette[/]]

Funktion:

Der Zeilenzaehler wird um n Zeilen verringert bzw. auf die Zeile mit der spezifizierten Zeichenkette eingestellt. Die Mitteilung 'STRING NOT IN BLOCK' zeigt an, dass die Zeichenkette nicht gefunden wurde. Das Kommando wird damit beendet.

Beispiel:

```
>P                          ;Ausgabe 1 Zeile
zeile 10
>U 4
zeile 6
>U /3
line 3
>B                          ;Ende Datei
zeile 900
>U /line 3/
STRING NOT IN BLOCK         ;Zeichenkette nicht ge-
                             funden
>P
zeile 504
>
```

7.22. Verify

Funktione:

Verlaesst den Brief-Modus und gibt die Textzeile aus nach den Kommandos:
Bottom, Change, Find, GEt, Locate, Next, Up

Beispiel:

```
>BR                     ;Brief-Modus
>N 3
>V
>N 3                    ;Verify-Kommando
zeile 20
>
```

7.23. Window

Funktion:

Zeigt die Zeilennummern der ersten und letzten Textzeile im Editierungspuffer an,

Beispiel:

```
>W
0001
0500
>
```

7.24. Xecute

Funktion:

Ausfuehrung die mit dem Kommando "M" im Makropuffer gespeicherte Kommandofolge.

Beispiel:

```
>M &U 3 &P 6&           ;Laden Makropuffer
>P
zeile 30
>X                      ;Ausfuehren Kommandos
zeile 27                ;U 3
zeile 27                ;P 6
zeile 28
zeile 29
zeile 30
zeile 31
zeile 32
>
```

8. Zusammenfassung der UDOS-Editor Kommandos

Kommando-Abkuerzung	Kommando-Name	Kommando-Parameter
A	Again	[# of times]
B	Bottom	
BR	Brief	
C	Change	/del string/new/string/ [# of lines[# of times per line]]
DE	Delete	[# of lines|/string/]
F	Find	/string/
GE	Get	[filename]
G	Goto	line #
I	Input	[textline]
J	Join	&command&command...
LI	Lineno	
L	Locat	/string/
M	Macro	&command&command...ü
N	Next	[line #|/string/]
P	Print	[# of lines|/string/]
PU	Put	[# of lines|/string/ [filename[record length]]]
Q	Quit	
R	Replace	[text line]
T	Top	
U	Up	[# of lines|/string/]
V	Verify	
W	Window	
X	Xecute	

Teil 2

F I L E . D E B U G

Diskettenmonitor

Inhaltsverzeichnis Seite

1. Einfuehrung

Das Systemprogramm FILE.DEBUG dient zur Ueberpruefung und eventuellen Reparatur von Saetzen und Dateien einer UDOS-Diskette. Es bezieht sich auf den im Band "UDOS-Systemhandbuch" der P8000-Dokumentation angegebenen Aufbau einer Diskette bzw. Datei.

FILE.DEBUG arbeitet mit vier Puffern:
- Datenpuffer: Anfangsadresse = Startadresse + 0000H
  enthaelt die gelesenen bzw. die zu schreibenden Daten der Datenfelder der Sektoren
  (Laenge des Puffers = 2000H = max. Spurlaenge)
- Maskenpuffer: Anfangsadresse = Startadresse + 0900H
  gibt an, welche Bits aus dem Vergleichspuffer zum Vergleich mit dem Datenpuffer beim S-Kommando herangezogen werden sollen
  (Laenge des Puffers = 100H)
- Vergleichspuffer: Anfangsadresse = Startadresse + 0A00H
  gibt das Bitmuster an, mit dem der Inhalt der Datenfelder der Sektoren beim S-Kommando verglichen werden soll
  (Laenge des Puffers = 100H)
- Zeigerblockpuffer: Anfangsadresse = Startadresse + 0B00H
  enthaelt nach dem 0-Kommando und folgenden Kommandos den aktuellen Zeigerblock der Datei
  (Laenge des Puffers = 100H)

Die Diskadresse DA wird jeweils als vierstellige Hexadezimalzahl ein- bzw. ausgegeben. Die beiden ersten Ziffern der Diskadresse enthalten die Spurnummer, die restlichen Ziffern enthalten in Bit 7,6,5 die Laufwerksnummer und in Bit 4,3,2,1,0 die Sektornummer.

Beispiel:
  DA = 1625: Spur 16, Sektor 05 Laufwerk 1
  DA = 276E: Spur 27, Sektor 0E Laufwerk 3
  DA = 0000: Spur 00, Sektor 00 Laufwerk 0

Die Arbeit auf der Diskette erfolgt entsprechend der eingetragenen Diskettenkonfiguration (s. Band "UDOS-Systemhandbuch" der P8000-Dokumentation, Abschn. 4.37):

Diskettentyp	Spurnummer	Sektornummer	max. Recordlaenge
Typ 2 und 3 (40 Spuren/ 16 Sektoren)	0 - 27H	0 - 0FH	1000H
Typ 4 (80 Spuren/ 16 Sektoren)	0 - 47H	0 - 0FH	1000H
Typ 5 (80 Spuren/ 32 Sektoren)	0 - 47H	0 - 1FH	2000H

Nach Aufruf von FILE.DEBUG meldet sich das Programm mit dem PROMPT-Zeichen "#". Jetzt kann eines der moeglichen Kommandos eingegeben werden.

2. Kommandouebersicht

R [diskadresse] [recordlaenge]
Record lesen (Read)

W [diskadresse] [recordlaenge]
Record schreiben (Write)

O dateiname
Datei eroeffnen (Open)

T [Dateiname]
Leselauf durch die Datei (Trace)

L -
aktuellen Stand der internen Diskadresse und der Recordlaenge anzeigen (List)

D -
Sprung in den U880-Softwaremonitor (Debug)

S -
Datei durchsuchen nach bestimmten Sektorinhalt (Search)

Q -
Rueckkehr ins Betriebssystem

3. Kommandos

3.1. Kommando "R" (Read)

Syntax: R [Diskadresse] [recordlaenge]

Das Kommando ermoeglicht das Lesen der Datenfelder einzelner bzw. aufeinanderfolgender Sektoren. Die Recordlaenge muss nicht angegeben werden, sie wird dann implizit auf 256 gesetzt. Standardmaessig werden Zugriffe auf das Laufwerk 0 durchgefuehrt. wird ein anderes Laufwerk gewuenscht, muss es in der Diskadresse mit angegeben werden. Der Inhalt des Datenfeldes (der Datenfelder) wird in den Datenpuffer gelesen. ausserdem werden die Diskadresse (ohne Laufwerkangabe) und die Recordlaenge ueber Konsole ausgegeben.
Ein R-Kommando ohne Parameter ist nur nach vorherigem O-Kommando moeglich. Die implizite Recordlaenge ist dann gelcih der Recordlaenge der Datei, die mit dem O-Kommando eroeffnet wurde.

Beispiel:
```
#R 1625 100      (Lesen Spur 16, Sektor 5, Laufwerk 1)
DA=1605  RL=100
#...
```

3.2. Kommando "W" (Write)

Syntax: W [Diskadresse] [recordlaenge]

Das Kommando ermoeglicht das Schreiben der Datenfelder einzelner oder aufeinanderfolgender Sektoren (Records). Ansonsten gelten die Hinweise fuer das R-Kommando. Zusaetzlich wird nochmals abgefragt, ob die Schreiboperation durchgefuehrt werden soll. Ein "Y" oder "y" startet die Schreiboperation, jede andere Eingabe verhindert sie.

Beispiel:
```
#W 0003 100      (Schreiben Spur 0, Sektor 3, Laufwerk 0)
DA=0003  RL=100 ?Y
#...
```

3.3 Kommando "O" (Open)

Syntax: O dateiname

Das Kommando sucht auf dem angegebenen Laufwerk in dem Directory der Diskette nach dem angegebenen Dateinamne und liest den Deskriptorrecord in den Datenpuffer und den ersten Zeigerblock in den Zeigerblockpuffer. Der Dateiname kann eine Laufwerkangabe enthalten. Es werden die Adressen der durchsuchten Directorysektoren, des Deskriptors der gefundenen Datei, des ersten und des letzten Datenrecords der Datei sowie des ersten Zeigerblocks der Datei ueber Konsole ausgegeben (ohne Laufwerkangabe).
Nach einem erfolgreichen "Open" kann mit dem Kommando "R" (ohne Parameter) vorwaerts Record fuer Record der Datei in den Datenpuffer eingelesen werden. Mit der Eingabe von "^" kann dieser Vorgang auch rueckwaerts ablaufen. Ebenso sind nach erfolgreichem "Open" die Kommandos "T" (ohne Parameter) und "T ^" moeglich.

Beispiel:
```
#O 0/STATUS
DA=1600  RL=100     (Deskriptor des Directorys)
DA=1605  RL=100     (durchsuchter Directoryrecord)
DA=160A  RL=100     (durchsuchter Directoryrecord)
DA=0D0F  RL=100     (Deskriptor der Datei STATUS)
        DSA=160A  FRA=0D03  LRA=0C05
        ZBL=0D02
#...
(DSA: Adresse des Directorysektors, in dem der Dateiname
      "STATUS" steht
 FRA: erster Datenrecord der Datei "STATUS"
 LRA: lezter Datenrecord der Datei "STATUS"
 ZBL: erster Zeigersektor der Datei "STATUS")
```

3.4. Kommando "T" (Trace)

Syntax: T dateiname

Fuer die angegebene Datei (Laufwerkangabe moeglich) wird ein O-Kommando ausgefuehrt. Ist die Datei vorhanden, werden fortlaufend R-Kommandos bis zum Dateiende oder bis zum Auftreten eines Fehlers erzeugt.
Ist fuer die Datei schon ein O-Kommando ausgefuehrt wurden, koennen auch folgende Kommandos aufgerufen werden:

T (ohne Parameter):	fortlaufende Leseoperationen vorwaerts durch die Datei
T ^	fortlaufende Leseoperationen rueckwaerts durch die Datei bis der Deskriptor der Datei erreicht ist

Bei allen Kommandos wird der Inhalt des Zeigerblockpuffers nach Bedarf aktualisiert.

Beispiel:

```
#T 0/STATUS
DA=1600  RL=100     (Deskriptor des Directorys)
DA=1605  RL=100     (durchsuchter Directoryrecord)
DA=160A  RL=100     (durchsuchter Directoryrecord)
DA=0D0F  RL=100     (Deskriptor der Datei STATUS)
        DSA=160A  FRA=0D03  LRA=0C05
        ZBL=0D02
DA=0D03  RL=100     (Datenrecords der Datei STATUS)
DA=0D04  RL=100
DA=0C05  RL=100
#...
```

(DSA: Adresse des Directorysektors, in dem der Dateiname "STATUS" steht
FRA: erster Datenrecord der Datei "STATUS"
LRA: lezter Datenrecord der Datei "STATUS"
ZBL: erster Zeigersektor der Datei "STATUS")

3.5. Kommando "L" (List)

Syntax: L

Das Kommando zeigt den aktuellen Stand von Da und RL an.

3.6. Kommando "D" (Debug)

Syntax: D

Das Kommando ermoeglicht den Uebergang in den U880-Softwaremonitor mit allen implementierten Kommandos (besonders: Anzeige und Aenderung der Pufferinhalte mit Kommando "D"). Durch Eingabe von "Q" erfolgt die rueckkehr zu FILE.DEBUG.

3.7. Kommando "S" (Search)

Syntax: S

Das Kommando sucht Sektoren auf der Diskette im zuletzt angesprochenen Laufwerk, deren Inhalt mit dem Vergleichspuffer uebereinstimmt. Der Vergleich wird bitweise fuer gesetzte Markenbits im Markenpuffer durchgefuehrt. die Diskadressen der Sektoren, bei denen Uebereinstimmung besteht, werden ausgegeben. Der Suchlauf kann durch Druecken einer Taste abgebrochen werden.

Beispiel:
Das Einrichten der Puffer erfolgt mit dem D-Kommando.
Maskenpuffer: 4900H: 00 FF FF FF 00 00 ...
Vergleichspuffer: 4A00H: xx D I R xx xx ...
#S
1605 (Der einzige Sektor, der in der Byteposition 1,2,3 die ASCII-Werte fuer "DIR" enthaelt, ist der Sektor 1605 (Spur 16, Sektor 05))
#...

3.8. Kommando "Q" (Quit)

Das Kommando bewirkt die Rueckkehr ins Betriebssystem UDOS.

Teil 3

U P R O G

EPROM-Programmer

Inhaltsverzeichnis Seite

1. Einfuehrung

Mit Hilfe des Programms UPROG sind im wesentlichen die folgenden Funktionen moeglich:

- Programmieren von EPROMs mit dem Inhalt einer Diskettendatei.
- Schreiben des Inhalts eines oder mehrerer EPROMs auf eine Diskettendatei.
- Duplizieren von EPROMs des gleichen Typs bzw. unterschiedlicher Typen.
- Anzeigen des Inhalts eines EPROMs auf der Konsole.
- Anzeigen und Modifizieren einzelner Bytes eines EPROMs
- Pruefen von EPROMs.

Programmiert werden koennen die EPROM-Typen:

U2708, U2716, U2732, U2732A, U2764

2. Aufrufen von UPROG

Nach dem Erscheinen des Promptzeichens (%) des UDOS-Betriebssystems kann UPROG durch das Kommando UPROG aufgerufen werden; also:

```
%UPROG
```

UPROG meldet sich mit '#', Danach kann ein entsprechendes Kommando einschliesslich der erforderlichen Parameter fuer die gewuenschte Funktion eingegeben werden.

3. Kommandouebersicht

PROGRAM parameter	Programmieren von EPROMs
SEPARATE parameter	Programmieren von EPROMs mit Bytes, die entwerde eine gerade oder eine ungerade Adresse haben
FILE parameter	Schreiben von EPROM-Inhalten auf Diskette
NEXT	Wiederholte Anwendung der Kommandos PROGRAM, SEPARATE und FILE, zwecks Bearbeitung mehrerer EPROMs
BYTE	Programmieren von EPROMs mit einzelnen Bytes
DUPLICATE parameter	Duplizieren von typgleichen EPROMs
COPY parameter	Duplizieren von typungleichen EPROMs
LIST parameter	Anzeigen des EPROM-Inhalts
TEST parameter	Testen des EPROMs
AGAIN	Kommandowiederholung
QUIT	Ruecksprung in das Betriebssystem

4. Notationsform

Fuer die Notation eines Kommandos ist es ausreichend, wenn nur der erste Buchstabe geschrieben wird. Es kann die Gross- oder Kleinschreibweise verwendet werden. Die Typbezeichnungen von EPROMs muessen stets gross geschrieben werden.

Bei der Erlaeuterung der Kommandos bzw. der Parameter bedeutete:

/	oder
[]	optional

5. Bedeutung der Kommandoparameter

pname spezifiziert die Typbezeichnung der folgenden moeglichen EPROMs:

U2708, U2716, U2732, U2732A, U2764

Beachte: Fuer den Typ U2732A ist U273A anzugeben.

Wird anstelle des Typs ein '*' notiert, dann erscheinen nach Abschluss des Kommandos auf der Konsole die Fragen:

EPROM OR BIPOLARE(E/B)? Es ist ein E einzugeben

NUMBER OF BYTES(HEX)? Die Antwort ist abhaengig vom Typ des EPROMs

Typ	Eingabe
U2708	400
U2716	800
U2732	1000
U2732A	1000
U2764	2000

Beachte: Wird 1000 eingegeben, dann erscheint auf der Konsoke die Frage:

2732A OR EQUIVALENT(Y/N)?

Im Falle eines U2732A bzw. equivalenten Typs ist die Frage mit 'Y' zu beantworten.

begadr spezifiziert die Adresse, ab der begonnen werden soll, den EPROM zu programmieren oder zu lesen.

Beachte: Der U2708 muss stets ab Adresse 0 programmiert werden.

numbytes spezifiziert die Anzahl der Bytes, mit denen der EPROM programmiert werden soll oder die vom EPROM gelesen werden sollen.
Beachte: Der U2708 muss vollstaendig programmiert werden.

E/O spezifiziert, dass der EPROM mit Bytes, die entweder eine gerade (even) oder eine ungerade (odd) Adresse haben, programmiert wird.
Diese Eigenschaft ist speziell fuer die Programmierung von EPROMs, die fuer 16-Bit-Mikrorechner bestimmt sind, wichtig.

M spezifiziert die Moeglichkeit des Anzeigens bzw. des Modifizieren einzelner Bytes des Datenpuffers vor dem Programmieren oder nach dem Lesen des EPROMs.
Waehrend der Abarbeitung des Kommandos erscheint auf der Konsole die Frage:

ADDRESS TO MODIFY?

Wird eine Adresse eingegeben, dann wird der Inhalt dieses Bytes angezeigt. Der Anwender hat jetzt folgende Moeglichkeiten der Eingabe:

- Eingabe eines hexadezimalen Wertes zwecks Modifikation des angezeigten Inhalts des Bytes.
- Eingabe eines <cr> zwecks Anzeigen des naechsten Bytes
- Eingabe eines '^' zwecks Anzeige des vorhergehenden Bytes.
- Eingabe eines '!' zwecks Ruecksprung zur Frage ADRESS TO MODIFY?.

Wird unmittelbar nach der Frage ein 'Q' eingegeben, dann wird diese verlassen und das Kommando weiter abgearbeitet.

V spezifiziert, dass der Inhalt des Ziel-EPROMs mit dem des Quell-EPROMs verglichen wird. Der Programmierschritt wird uebersprungen.

entadr spezifiziert die Entryadresse der Datei, die erzeugt wird, wenn der EPROM-Inhalt auf Diskette geschrieben wird.

reclen spezifiziert die Recordlaenge der Datei, die erzeugt wird, wenn der EPROM-Inhalt auf Diskette geschrieben wird.

6. Ausschriften waehrend der Kommandoabarbeitung

Vor jedem Programmieren, Lesen und Vergleichen von EPROMs werden die folgenden Fragen auf der Konsole angezeigt:

```
READY TO PROGRAM?     (Bereit zum Programmieren?)
READY TO READ?        (Bereit zum Lesen?)
READY TO VERIFY?      (Bereit zum Vergleichen?)
```

Ist die Bereitschaft vorhanden, so ist ein 'Y' oder <cr> einzugeben. Wenn keine Bereitschaft da ist, dann ist irgend ein anderes Zeichen einzugeben. Das Kommando wird verlassen und UPROG meldet sich mit '#'.

Sollte waehrend des Pruefens des EPROMs ein Fehler (oder mehrere) gefunden werden, dann erscheint auf der Konsole die folgende Nachricht:

```
PROGRAMMING ERROR AT nnn
SOURCE CONTENTS aa
DESTINATION CONTENTS bb
```

aa ist der Inhalt des Quell-EPROMs bzw. der Datei.
bb ist der Inhalt des Ziel-EPROMS bei der Adresse nnn.

wird ein Fragezeichen (?) eingegeben, dann wird das Pruefen so lange unterbrochen, bis erneut ein Fragezeichen eingegeben wird.
Die Eingabe von 'ESCAPE' bewirkt, dass das Pruefen abgebrochen wird und UPROG sich mit '#' meldet.

7. Kommandos

7.1. PROGRAM

Dieses Kommando ermoeglicht das Programmieren von EPROMs mit dem Inhalt einer Diskettendatei.

Syntax:

```
P[ROGRAM]  dateiname pname/* [B=begadr][N=numbytes]
                             [M][V]
```

Die Datei mit dem Namen 'dateiname' wird, beginnend bei der Adresse 'begadr' und der Byteanzahl 'numbytes', in den Datenpuffer geschrieben. Dieser Datenpuffer kann mit Hilfe der Option 'M' modifiziert werden. anschliessend wird der EPROM programmiert und geprueft.
Sollte der EPROM vor dem Programmieren nicht geloescht sein, dann erscheint auf der Konsole die folgende Frage:

```
PROM NOT ERASED-PROGRAM ANYWAY?

(EPROM nicht geloescht-trotzdem programmieren?)
```

Soll der EPROM programmiert (ueberprogrammiert) werden, ist ein 'Y' einzugeben, andernfalls ein 'N'. Das Kommando wird verlassen und UPROG meldet sich mit '#'.

Wird die Option 'V' spezifiziert, dann wird der Programmierschritt uebersprungen und der EPROM wird nur geprueft.

Standardwerte der Parameter:

begadr : niedrigste Dateiadresse
numbytes : Anzahl der Bytes im EPROM

7.2. SEPARATE

Dieses Kommando ermoeglicht das Programmieren von EPROMs mit Bytes, die entweder eine gerade oder eine ungerade Adresse haben. diese Eigenschaft des Kommandos ist speziell fuer die Programmierung von EPROMs, die fuer 16-Bit-Mikrorechner bestimmt sind, wichtig.

Syntax:

```
S[EPARATE]  dateiname pname/* [B=begadr][N=numbytes]
                              [M][V][E/O]
```

Die Datei mit dem Namen 'dateiname' wird, beginnend bei der Adresse 'begadr' und der Byteanzahl 2*'numbytes', in den Datenpuffer geschrieben. Dieser Datenpuffer kann mit Hilfe der Option 'M' modifiziert werden. Anschliessend wird der EPROM in Abhaengigkeit der Option 'E/O' programmiert und geprueft.

```
Option 'O'(odd)  : Alle Bytes mir ungeraden Adressen
                   ab der Adresse 'begadr'.
Option 'E'(even): Alle Bytes mit geraden Adressen ab
                   der Adresse 'begadr+1'.
```

Sollte der EPROM vor dem Programmieren nicht geloescht sein, dann erscheint auf der Konsole die folgende Frage:

```
PROM NOT ERASED-PROGRAM ANYWAY?

(EPROM nicht geloescht-trotzdem programmieren?)
```

Soll der EPROM programmiert (ueberprogrammiert) werden, ist ein 'Y' einzugeben, andernfalls ein 'N'. Das Kommando wird

Wird die Option 'V' spezifiziert, dann wird der Programmierschritt uebersprungen und der EPROM wird nur geprueft.

Standardwerte der Parameter:

```
begadr   : niedrigste Dateiadresse
numbytes : Anzahl der Bytes im EPROM
```

7.3. FILE

Dieses Kommando ermoeglicht das Schreiben von EPROM-Inhalten auf Diskette.

Syntax:

```
F[ILE]  pname/* dateiname [B=begadr][N=numbytes][M]
                          [E=entadr][RL=reclen]
```

Es werden Bytes der Anzahl 'numbytes' vom EPROM ab der Adresse 'begadr' in den Datenpuffer eingelesen. Dieser Datenpuffer kann mit Hilfe der Option 'M' modifiziert werden. Anschliessend wird der Inhalt des Datenpuffers in die Datei mit dem Namen 'dateiname' geschrieben. Die Datei hat die Entryadresse 'entadr' und die Reordlaenge 'reclen'.

soll der Inhalt von mehr als einem EPROM in die Datei geschrieben werden, dann wird fuer den 2.,3.,... EPROM das NEXT-Kommando verwendet.
Die Datei wird aus diesem Grunde erst durch ein folgendes anderes Kommando abgeschlossen.

Standardwerte der Parameter:

begadr : 0
numbytes : Anzahl der Bytes im EPROM
entadr : 0
reclen : entsprechend des UDOS-Betriebssystems

7.4. NEXT

Dieses Kommando ermoeglicht, dass bei der Anwendung des vorherigen PROGRAM- bzw. SEPARATE-Kommandos mehr als ein EPROM programmiert werden kann. Das kann erforderlich sein, wenn die Kapazitaet der entsprechenden Datei groesser ist als die des EPROMs. Bei der Anwendung des FILE-Kommandos kann der Inhalt von mehreren EPROMs in die gleiche Datei geschrieben werden.

Syntax:

N[EXT]

War das vorherige Kommando PROGRAM oder SEPARATE, dann gilt:

Die naechsten Bytes der Anzahl 'numbytes' werden von der Datei mit dem Namen 'dateiname', der beim PROGRAM- oder SEPARATE-Kommando notiert wurde, gelesen. Die Adresse 'begadr' wird zuvor automatisch um die Byteanzahl 'numbytes' erhoeht. Der weitere Ablauf ist dann wie beim PROGRAM- oder SEPARATE-Kommando.

War das vorherige Kommando FILE, dann gilt:

Der Inhalt des 2.,3.,...EPROMs wird in die Datei mit dem Namen 'dateiname', die beim FILE-Kommando notiert und eroeffnet wurde, geschrieben.

Beachte: Die Optionen sind stets die des vorherigen Kommandos.

7.5. BYTE

Dieses Kommando ermoeglicht das Programmieren (Korrigieren) einzelner Bytes eines EPROMs.
Soll ein bereits programmierter EPROM korrigiert werden, dann sind die folgenden Bedingungen fuer eine Wertigkeitsaenderung der Bits des ausgewaehlten Bytes zu beachten:

```
'Eins' ---> 'Null'    moeglich
'Eins' ---> 'Eins'       "
'Null' ---> 'Null'       "
'Null' ---> 'Eins'    nicht moeglich
```

Syntax:

```
B[YTE]  pname/*
```

Unmittelbar nach dem Start des Kommandos erscheint auf der Konsole die Frage:

```
ADDRESS TO MODIFY?
```

Wird eine Adresse eingegeben, dann wird der Inhalt dieses Bytes angezeigt. Der gesamte Inhalt des EPROMs wird nicht in den Datenpuffer geschrieben. Der Anwender hat jetzt folgende Moeglichkeiten der Eingabe:

- Eingabe eines hexadezimalen Wertes zwecks Modifikation des angezeigten Inhalts des Bytes.
  Sollte der eingegebene Wert nicht programmierbar sein, weil bei einem oder mehreren Bits des Bytes Wertigkeitsaenderungen nicht moeglich sind, dann erscheint auf der Konsole die Nachricht:

  ```
  BIT ALREADY PROGRAMMED    (Bit bereits programmiert)
  ADDRESS TO MODIFY?
  ```

  Nach erfolgter Modifikation wird der Inhalt des naechsten Bytes angezeigt.

- Eingabe eines <cr> zwecks Anzeigen des naechsten Bytes.
- Eingabe eines '^' zwecks Anzeige des vorhergehenden Bytes.
- Eingabe eines 'Q' zwecks Ruecksprung zu der Frage ADDRESS TO MODIFY?

Wird unmittelbar nach der Frage ein 'Q' eingegeben, dann wird das Kommando verlassen und UPROG meldet sich mit '#'.

Beachte: Fuer den EPROM U2708 ist dieses Kommando nicht verwendbar.

7.6. DUPLICATE

Dieses Kommando ermoeglicht das Duplizieren oder das Vergleichen von typgleichen EPROMs.

Syntax:

```
D[UPLICATE]  pname/* [B=begadr][N=numbytes]
                    [M][V]
```

Der Inhalt des Quell-EPROMs wird, beginnend bei der Adresse 'begard' und der Byteanzahl 'numbytes', in den Datenpuffer geschrieben. dieser Datenpuffer kann mit Hilfe der Option 'M' modifiziert werden.
Der Quell-EPROM kann jetzt gegen den Ziel-EPROM ausgetauscht werden. Der Ziel-EPROM wird mit dem Inhalt des Datenpuffers, beginnend bei der Adresse 'begadr' und der Byteanzahl 'numbytes', programmiert und geprueft.

Wird die Option 'V' spezifiziert, dann wird der Programmierschritt uebersprungen und der Ziel-EPROM wird nur geprueft, d.h., der Inhalt des Ziel-EPROMs wird mit dem des Quell-EPROMs verglichen.

Standardwerte:

begadr : 0
numbytes : Anzahl der Bytes im EPROM

7.7. COPY

Dieses Kommando ermoeglicht das Duplizieren oder das Vergleichen von typungleichen EPROMs.

Syntax:

```
C[OPY}  S:pname [B=begadr1][N=numbytes1]
        D:pname [B=begadr2][N=numbytes2][M][V]
```

Der Inhalt des Quell-EPROMs mit der Syntax S:pname (S=source) wird, beginnend bei der Adresse 'begadr1' und der Byteanzahl 'numbytes1', in den Datenpuffer geschrieben. Dieser Datenpuffer kann mit Hilfe der Option 'M' modifiziert werden.
Der Quell-EPROM kann jetzt gegen den Ziel-EPROM mit der Syntax D:pname (D=destination) ausgetauscht werden. Der Ziel-EPROM wird mit dem Inhalt des Datenpuffers, beginnend bei der Adresse 'begadr2' und der Byteanzahl 'numbytes2', programmiert und geprueft.

Wird die Option 'V' spezifiziert, dann wird der Programmierschritt uebersprungen und der Ziel-EPROM wird nur geprueft, d.h., der Inhalt des Ziel-EPROMs wird mit dem des Quell-EPROMs verglichen.

Standardwerte:

```
begadr1   : 0
numbytes1 : Anzahl der Bytes im EPROM
begadr2   : 0
numbytes2 : numbytes1
```

7.8. LIST

Dieses Kommando ermoeglicht das Auslisten des Inhalts eines EPROMs auf der Konsole.

Syntax:

```
L[IST]  pname/* [B=begadr][N=numbytes]
```

Der Inhalt des Quell-EPROMs wird, beginnend bei der Adresse 'begadr' und der Byteanzahl 'numbytes', in den Datenpuffer geschrieben und dann auf der Konsole ausgelistet.
Nach jeweils 20 Zeilen erscheint auf der Konsole die Frage:

```
CONTINUE?        (fortsetzen?)
```

Sollen die naechsten Zeilen ausgelistet werden, so ist ein 'Y' oder <cr> einzugeben. Wenn das Auslisten abgebrochen werden soll, dann ist irgendein anderes Zeichen einzugeben. Das Kommando wird verlassen und UPROG meldet sich mit '#'. Wird waehrend des Auslistens des EPROM-Inhalts ein Fragezeichen (?) eingegeben, dann wird das Auslisten solange unterbrochen, bis erneut ein Fragezeichen eingegeben wird.

Standardwerte:

begadr : 0
numbytes : Anzahl der Bytes im EPROM

Bemerkung:

Wenn der Anwender den Inhalt eines EPROMs ausdrucken moechte, ist es am zweckmaessigsten, wenn mit Hilfe des UPROG-Kommandos 'FILE' eine Hilfsdatei auf der Diskette angelegt wird und diese dann mit dem UDOS-Kommando 'DUMP' auf dem Drucker ausgegeben wird.

Kommandofolge:

```
%ACTIVATE $LP
%DEFINE 3 $LP
%UPROG
#FILE pname dateiname
#QUIT
$DUMP dateiname
```

7.9. TEST

Dieses Kommando ermoeglicht die Ermittlung des 'CRC' eines EPROMs.

Syntax:

T[EST] pname/* [B=begadr][N=numbytes]

Das 'CRC' wird fuer den Bereich gebildet, der bei der Adresse 'begadr' beginnt und die Byteanzahl 'numbytes' hat. Das Ergebnis erscheint auf der Konsole in der Form:

PROM-CRC: hexadezimaler Wert

7.10. AGAIN

Dieses Kommando ermoeglicht es, das vorherige Kommando zu wiederholen, sofern es ein PROGRAM-, SEPARATE-, BYTE-, DUPLICATE-, COPY- oder LIST-Kommando war.

Syntax:

A[GAIN]

War das vorherige Kommando PROGRAM, SEPARATE,DUPLICATE oder COPY, dann gilt:

Der Ziel-EPROM wird mit dem Inhalt des Datenpuffers programmiert und geprueft.

War das vorherige Kommando BYTE oder LIST, dann gilt:

Es werden alle Funktionen des Kommandos wiederholt.

Beachte: die Optionen sind stets die des vorherigen Kommandos

7.11. QUIT

Dieses Kommando ermoeglicht den Ruecksprung in das UDOS-Betriebssystem.

Syntax:

Q[UIT]

Bevor sich das UDOS-Betriebssystem mit seinem Promptzeichen (%) meldet, werden eventuell noch nicht geschlossene Dateien geschlossen.

Teil 4

U F O R M

Programm zur Textformatierung

Inhaltsverzeichnis Seite

1. Ueberblick

1.1. Verwendungszweck

Das Programm UFORM ist durch mannigfaltige Funktionen zur Textformatierung fuer die Vereinfachung der Aufbereitung von Dokumenten bestimmt. UFORM verarbeitet eine oder mehrere Eingabedateien, die mit einem Texteditor aufbereitet bzw. erstellt wurden und erzeugt eine Ausgabedatei, die den formatierten Text enthaelt. Die in der Eingabedatei eingefuegten Kommandos verwendet UFORM zur Formatierung des Textes. Die Eingabedatei wird dabei nicht veraendert.

1.2. Textformatierung

A. Der Text wird in Seiten aufgeteilt.
   Spezifiziert werden koennen:
   - Laenge
   - Breite
   - Linker und rechter Rand
   - Oberer und unterer Rand
     (einschliesslich Seitennummern)

B. Es wird in verschiedenen Modi gearbeitet.
   Ausgewaehlt werden koennen:
   - Zeilenabstand: einfach, doppelt, usw.
   - Ausrichten / Nichtausrichten des Textes
   - Aufgefuellter / Nichtaufgefuellter Text:
     (s. Textumbruch)

C. Es werden Kommandos verarbeitet, die
   - Tabulatoren setzen
   - Zeilen einruecken
   - Textbloecke einruecken
   - Leerzeilen erzeugen
   - Zeilen zentrieren
   - eine Unterbrechung der Ausgabe ermoeglichen
   - eine neue Zeile bzw. Seite beginnen

D. Text kann ohne Kommandos formatiert werden.

UFORM besitzt Standardwerte und -modi. Dadurch kann ein Dokument in Seiten aufgeteilt werden, wenn die Eingabedatei wenige oder keine Kommandos enthaelt. die Anfangswerte sind in der Zusammenfassung der Kommandos enthalten.

1.3. Aufbereitung der Eingabedateien

Eingabedateien enthalten den zu formatierenden Text, gemischt mit den Kommandos zur Formatierung. Die Kommandos sind im Anhang dieses Dokumentes beschrieben. Jedes Kommando muss auf einer separaten Zeile erscheinen. Zur Erstellung der Eingabedatei, die Textzeilen und Kommandozeilen enthaelt, wird ein Standardeditor verwendet:

Textzeilen - koennen in beliebiger Form eingegeben werden.

Beispiel:

```
UFORM ist zur Vereinfachung
der Aufbereitung von Dokumenten
durch ...
```

Kommandozeilen - teilen UFORM mit, wie der Text zu formatieren ist

Beispiel:

```
.JU    Ausrichten des Textes
.PL 63 Setzen der Seitenlaenge auf 63 Zeilen
```

Kommandozeilen beginnen immer mit einem Punkt in der ersten Spalte, unmittelbar gefolgt von einem zweistelligen Kommandonamen. Gelegentlich muss ein Parameter folgen (wie z.B. die Anzahl der Zeilen).

1.4. Der formatierte Text

Der formatierte Text (Ausgabe) kann ueber jedes Geraet, das spezifiziert wird, ausgegeben werden (einschliesslich Bildschirm, Drucker, Diskettendateien). Fehlernachrichten, die von UFORM erzeugt werden, werden ueber den Bildschirm ausgegeben, unabhaengig davon, wohin der Text ausgegeben wird.

2. Anwendung von UFORM

Ist die Eingabedatei aufbereitet und UFORM im System vorhanden, kann UFORM durch Eingabe mitgeteilt werden, welche Dateien formatiert werden sollen:

Beispiel: %UFORM datei

Der vollstaendige Inhalt von "datei" wird formatiert. Der formatierte Text erscheint auf dem Bildschirm. Die Eingabedatei "datei" wird nicht veraendert. Ueber die Ausgabe-Auswahlbedingungen kann UFORM veranlasst werden, den formatierten Text ueber den Drucker oder eine Diskettendatei auszugeben.

3. Optionale Ausgabeparameter

Die Ausgabe-Auswahlbedingungen koennen verwendet werden, um die Ausgabe ueber ein anderes Geraet als den Bildschirm (O=), das Drucken bestimmter Seiten des formatierten Dokumentes (P=) oder das Warten zwischen dem Druck von Seiten zur Ueberpruefung oder zum Papierwechsel (W=) zu veranlassen.

Die Auswahlbedingungen sind in derselben Zeile, in

welcher der Aufruf von UFORM erfolgt, anzugeben. Sie enthalten einen einzelnen Buchstaben unmittelbar gefolgt von einem Gleichheitszeichen (=) und danach das Argument. Die Auswahlbedingungen koennen als Gross- oder Kleinbuchstaben angegeben werden. Leerzeichen duerfen die Teile der Auswahlbedingung nicht trennen. Die Auswahlbedingungen koennen in beliebiger Reihenfolge nach dem Dateinamen angegeben werden.

3.1. Benennen der Ausgabedatei (O=)

Standardmaessig erfolgt die Ausgabe ueber den Bildschrim. die Auswahlbedingung kann verwendet werden, um die ausgabe ueber ein anderes Geraet vorzunehmen.

Beispiel: %UFORM datei1 datei2 O=ausgabedatei

"datei1" und "datei2" werden formatiert, die Ausfabe erfolgt in die Datei mit dem Namen "ausgabedatei".

%UFORM Handbuch O=$PRINTER

formatiert "Handbuch" und druckt den formatierten Text ("$PRINTER" ist die Druckerdatei)

%UFORM testdatei

formatiert "testdatei", da keine Auswahlbedingung angegeben wurde, erfolgt die Ausgabe ueber den Bildschirm

3.2. Angabe der auszugebenden Seiten (P=)

Gewoehnlich werden alle Seiten eines Dokumentes ausgegeben. Durch die Verwendung dieser Auswahlbedingung kann die Ausgabe bestimmter Seiten des formatierten Dokumentes veranlasst werden.

```
%UFORM datei P=3 Ausgabe der 3.Seite

%UFORM datei P=-3 Ausgabe vom Anfang bis
                  zur 3.Seite
%UFORM datei P=+3 O=datei.f
        Ausgabe von Seite 3 bis zum Ende
        in die Datei "datei.f"
```

3.3. Unterbrechen der Ausgabe (W=)

Bei Verwendung dieser Auswahlbedingung wartet UFORM nach dem Drucken einer festgelegten Anzahl von Seiten und setzt die Ausgabe erst nach dem Druck einer beliebigen Taste auf der Tastatur fort.

Beispiel: %UFORM datei W=1 warten nach jeder Seite

%UFORM diese_datei W=2 warten nach jeder Seite, deren Nummer ein Vielfaches von 2 ist..

4. Kommandosyntax

Kommandozeilen beginnen immer mit einem Punkt in der ersten Spalte. Jedes Kommando hat einen zweistelligen Namen. Zur Erhoehung der Verstaendlichkeit koennen die Namen auch in einer laengeren form geschrieben werden, wenn die ersten beiden Buchstaben unveraendert bleiben.

Beispiel:	.FILL	oder	.FI
	.PLENGTH	oder	.PL

Die beiden Buchstaben koennen gross oder klein, aber nicht gemischt angegeben werden.

Jede Zeile, die mit einem Punkt beginnt, wird als Kommandozeile betrachtet. Jeder nicht erkannte Kommandoname wird ignoriert. (Druck einer Textzeile, die mit einem Punkt beginnt, s.u. Behandlung von Spezialzeichen)

5. Kommando-Argumente

Nach einigen Kommandos wird nie ein Argument angegeben, solche wie:

.FI	Beginn Fuellmodus
.NF	Ende Fuellmodus

Nach den meisten Kommandos wird ein numerisches Argument erwartet, das eine ganze Zahl (bis zu 5 Ziffern), moeglicherweise mit einem vorangestellten Plus(+) oder Minus(-), ist.

Diese Argumente spezifizieren bestimmte Parameter (wie den rechten Rand oder die Seitenlaenge), die in einen "absoluten" oder "relativen" Wert umgewandelt werden. Zwischen Kommando und Argument muss ein Leerzeichen stehen.

5.1. Absolute Argumente

Diese Argumente setzen Parameter auf den spezifizierten Wert.

Beispiel:	.RM	65	setzt den rechten Rand auf Spalte 65
	.LS	2	setzt den Zeilenabstand auf 2 (doppelter Abstand)

5.2. Relative Argumente

Relative Argumente aendern den aktuellen Wert eines Parameters. Sie haben ein vorangestelltes Plus(+) oder Minus(-), um den aktuellen Wert eines Parameters zu erhoehen oder zu vermindern.

Beispiel:	.IN +10	Text um 10 Positionen nach rechts einruecken
	.RM -10	Einruecken des rechten Randes um 10 Positionen nach links

Es ist zu beachten, dass zwischen Vorzeichen und Zahl kein Leerzeichen erlaubt ist.

5.3. Standardvorgaben

Bei Weglassen eines Argumentes wird automatisch ein sinnvoller "Standard" - Wert verwendet.

Beispiel:

.CE	Zentrieren einer Textzeile
.LS	einfacher Zeilenabstand
.In	Setzt den Zeilenanfang auf den linken Rand (Beendet das Einruecken eines Textblockes)

Die Standardwerte fuer die Seitenanordnung und die Textmodi sind dieselben wie anfangs.

5.4. Unzulaessige Parametervorgaben

Wird versucht, einen Parameter auf einen Wert zu setzen, der groesser (oder kleiner) als das Maximum (oder Minimum) ist, wird anstelle des geforderten Wertes der Wert auf das Maximum (oder Minumum) gesetzt. Das Programm wird fortgesetzt und gibt Mitteilungen auf dem Bildschirm aus. (wertebereiche s.u. Implementierungshinweise)

6. Aufbau der Seiten

6.1. Seitengestaltung

A .Erzeugen des oberen und unteren Randes, entweder:

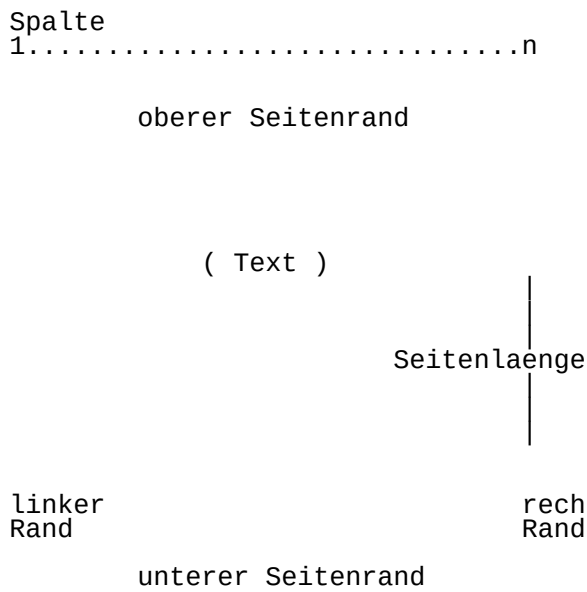
- nur Leerzeilen oder
- einschliesslich Titel und moeglicherweise die aktuelle Seitennummer

B .Begrenzen der Zeilenanzahl auf das spezifizierte Maximum

C. Begrenzen der Druckpositionen in der Zeile auf ein festgelegtes Maximum

D .Versetzen des Textes vom linken Seitenrand um die Anzahl der festgelegten Positionen

6.2. Seitenformat



6.3. Seitennummerierung

UFORM teilt den Text in Seiten ein und kennt immer die aktuelle Seitennummer, ganz egal, ob der Druck der Nummern gefordert wird oder nicht. Um die Seitennummern zu drucken, muss man sich entscheiden, ob die Nummern am Seitenanfang oder -ende erscheinen sollen. Entsprechend wird die Kopf- bzw. Fusszeile festgelegt (siehe Titelzeilen).

Soll nur ein Teil eines Dokumentes formatiert werden und die erste erzeugte Seite eine andere Nummer als eins erhalten, kann das Kommando .BP benutzt werden.

7. Kommandos fuer die Seitengestaltung

7.1. Seitenlaenge

.PL	n	setzt die Seitenlaenge auf n Zeilen

Das ist die maximale Anzahl von Zeilen, die auf einer einzelnen Seite ausgegeben werden koennen. Die Seitenlaenge schliesst alle Zeilen am Seitenanfang und -ende mit ein

(einschliesslich Leerzeilen)
Ist das Seitenende erreicht, erzeugt UFORM einen Seitenvorschub (Form Feed), bevor eine neue Seite begonnen wird. Einige Geraete erzeugen einen eigenen Seitenvorschub, wenn die Anzahl der Zeilen ein bestimmtes Maximum erreicht hat.
Um einen zusaetzlichen Seitenvorschub zu vermeiden, empfiehlt sich die Angabe einer Seitenlaenge <=60 fuer die Ausgabe an den meisten Geraeten.

7.2. Linker Rand

.PO n versetzen der Seite um n Positionen

Der vollstaendige Textkoerper wird um n Positionen vom linken Rand der physischen Seite versetzt.
Dieses Kommando bestimmt, wo sich der linke Rand befindet, d.h., die Spalte 1 beginnt in jeder Zeile nachdem n Leerzeichen ausgegeben werden. Vor dem linken Seitenrand kann nicht gedruckt werden.

7.3. Rechter Rand

.RM n setzen des rechten Randes auf Spalte n

Die Lage der n-ten Spalte ist relativ zur Spalte 1, die durch die Festlegung des linken Randes bestimmt ist und nicht mit dem linken Rand der physischen Seite uebereinstimmen muss.

7.4. Titelzeilen

.HE t erzeugt Kopfzeile t am Anfang jeder Seite
.FO t erzeugt Fusszeile t am Ende jeder Seite

Der Titel t wird in 3 Teile geteilt:

1. Teil: links ausgerichtet
2. Teil: in der Mitte
3. Teil: rechts ausgerichtet

Der Titel erstreckt sich zwischen linkem und rechtem Rand.

Beispiel:

.HE 'teil 1''teil 2''teil 3'	3teiliger Titel
.HE ''-#-''	Seitennummer in der Mitte
.HE ''UFORM 1.5''	Mitte
.FO 'Ende der Seite -#-'''	links ausgerichtet
.FO '''-#-'	Seitennummern rechts ausgerichtet

Das Seitennummernzeichen (#) wird durch die aktuelle Seitennummer ersetzt, wenn es in einem Titel erscheint.

Der Apostroph (') wurde in diesen Beispielen als Trennzeichen benutzt; es koennen aber auch andere Zeichen verwendet werden. Das erste Zeichen (ein anderes als Leerzeichen oder Tabulator) nach dem Kommandonamen wird als Trennzeichen angenommen. Mindestens 3 Trennzeichen muessen angegeben werden, auch wenn ein Titelteil weggelassen wird. Das letzte Trennzeichen kann weggelassen werden.

Sind die Titelteile zu lang, um zwischen die Raender zu passen, werden sie ueberlagert, und einige Zeichen koennen verloren gehen.

7.5. Oberer und unterer Rand

Der Rand fuer Seitenanfang und -ende, der ausgewaehlt wurde, wird auf jeder Seite beruecksichtigt und besteht aus 3 Teilen.

Ein Teil ist der laufende Titel (welcher eine Leerzeile sein kann) und die anderen 2 Teile sind die Anzahl der Leerzeilen, welche vor und nach der Titelzeile gewuenscht werden.

```
.AH  n          Ausgabe n Leerzeilen vor Kopfzeile
.HE  t          Kopfzeile (s.o.)
.BH  n          Ausgabe n Leerzeilen nach Kopfzeile

.AF  n          Ausgabe n Leerzeilen vor Fusszeile
.FO  t          Fusszeile (s.o.)
.BF  n          Ausgabe n Leerzeilen nach Fusszeile
```

Ist fuer das HE- oder FO-Kommando kein Argument angegeben, besteht der Titel aus einer Leerzeile. Es ist zu beachten, dass ein solcher titel auch die Ausgabe einer Zeile veranlasst, so dass der gesamte Rand so aufgebaut ist:

Zeilen vor Titelzeile + 1 Titelzeile + Zeilen nach Titelzeile

8. Textumbruch

8.1. Mehrfacher Zeilenabstand

```
.LS  n   setzt den Zeilenabstand auf n
```

Ist n=1 einfacher Zeilenabstand
n=2 doppelter Zeilenabstand usw.

8.2. Fuellmodus

.FI Auffuellen der nachfolgenden Ausgabezeilen

In diesem Modus wird jede Ausgabezeile mit der maximalen Anzahl von Woertern, die zwischen den linken und den rechten Rand passen, aufgefuellt. Dabei interessiert nicht die Anzahl der Woerter, die in jeder Eingabezeile stehen. UFORM ueberprueft jedesmal, ob ein Eingabewort in die aktuelle Ausgabezeile passt, ohne den rechten Rand zu erreichen. Wenn ja, wird das Wort in die aktuelle Zeile eingefuegt, wenn nicht, wird eine neue Zeile begonnen.

Es ist zu beachten, dass die Zeilenlaenge beruecksichtigt wird, aber ein gerader rechter Rand nur erzeugt wird, wenn auch der Ausrichtungsmodus aktiv ist.

Im Fuellmodus werden alle Tabulatoren (ausschliesslich Tabulatoren am Anfang der Eingabezeile) in Leerzeichen umgewandelt und nur als Trennzeichen behandelt. Zeilen, die Tabulatoren enthalten, sollten generell nicht im Fuellmodus ausgegeben werden.

Mehrfache Leerzeichen und Tabulatoren, die nicht am Anfang oder Ende einer Seite stehen, werden in ein Leerzeichen umgewandelt. Immer wenn ein Satztrennungszeichen gefunden wird (Punkt, Doppelpunkt, Fragezeichen, Ausrufezeichen, rechte Klammer), und zwei Leerzeichen folgen, werden sie uebernommen. In jeder Eingabezeile, die mit einem dieser Zeichen endet, werden in der Ausgabedatei zwei Leerzeichen hinzugefuegt.

Enthaelt eine Eingabezeile ein einzelnes Wort, welches so lang ist, dass es nicht zwischen linken und rechten Rand passt, wird es irgendwie ausgegeben.

8.3. Nicht-Fuellmodus

.NF nicht Auffuellen der nachfolgenden Zeilen

In diesem Modus werden die Ausgabezeilen nicht mit Worten aufgeufellt; sie sind den Eingabezeilen identisch (ausschliesslich des Einrueckens unter Verwendung des Einrueckungskommandos). Der Nicht-Fuellmodus sollte angewendet werden, wenn Eingabezeilen Tabellen, Diagramme und anderen Text enthalten, in denen sich zu uebernehmende Tabulatoren und Leerzeichen befinden.

In diesem Modus erfolgt keine Ueberpruefung des rechten Randes. Man sollte sich daher sicher sein, dass man diesen nicht ueberschreitet.

8.4. Ausrichtungsmodus

.JU ausrichten der nachfolgenden Zeilen

In diesem Modus wird ein gerader rechter Rand erzeugt. Dieses Dokument wurde im Ausrichtungsmodus erstellt. Ausgerichteter Text wird genau wie aufgefuellter Text erstellt, ausser dass zusaetzliche Leerzeichen zwischen den Woertern eingefuegt werden, um einen ausgerichteten rechten Rand zu

erhalten.
Dieser Modus wird nur eingestellt, wenn der Fuellmodus gewaehlt wurde.

8.5. Nicht-Ausrichtungsmodus

.NJ nicht Ausrichten der nachfolgenden Zeilen

Dieses ist der Standardmodus und kann benutzt werden, um das Ausrichten des rechten Randes abzustellen.

9. Einruecken

9.1. Einruecken eines Textblockes

.IN n Einruecken um n Positionen

Die Ausgabe des Textes beginnt gewoehnlich am linken Rand (in Spalte 1). Das Einrueckungskommando bewirkt den Beginn der nachfolgenden Textzeilen nach der Ausgabe von n Leerzeichen. Der Text wird solange eingerueckt, bis ein Kommando erscheint, das die Anzahl der einzurueckenden Positionen ruecksetzt.
Dieser Abschnitt wurde mir ".IN 5" begonnen. Dadurch wurde der linke Rand auf Spalte 5 eingestellt. durch ".ON" wird die Anzahl der einzurueckenden Positionen wieder auf Null gesetzt. Es ist zu beachten, dass der rechte Rand nicht veraendert wird.
Manchmal ist es einfacher, relative Argumente zu verwenden. Diesem Abschnitt wurde ein ".IN +10" und ein ".RM -10" vorangestellt, um sowohl den linken als auch den rechten Rand um 10 Positionen zu verschieben. Dieser Abschnitt wird gefolgt von ".IN -10" und ".RM +10", um die Raender wieder auf ihre vorigen Werte einzustellen. Diese Methode erlaubt es, absolute Spaltennummern, zu ignorieren und ist nuetzlich, wenn verschachtelte Niveaus von Einrueckungen gewuenscht werden.

9.2. Einruecken einer Textzeile

.TI n Einruecken der folgenden Zeile
um n Positionen

Dadurch wird der Text um n Positionen relativ zum aktuellen Einrueckungswert verschoben. dies gilt fuer die nachfolgende Textzeile.
Einigen Abschnitten wurde ein ".TI 5" vorangestellt, so dass die erste Zeile in jedem Abschnitt um 5 Positionen eingerueckt wurde und die anderen Zeilen nicht veraendert

wurden. Ist n kleiner als der aktuelle Einrueckungszaehler, wird ein "Rueckwaerts"- Einruecken durchgefuehrt.

9.3. Einruecken mittels Leerzeichen und Tabulatoren

Jede Zeile die mit fuehrenden Leerzeichen oder Tabulatoren beginnt, bewirkt den Beginn einer neuen Ausgabezeile, die wie gefordert eingerueckt word. Der Text, der nach einem Tabulator folgt, beginnt mit dem naechsten Tabulator-Stop.

Dadurch ist es moeglich, individuell Zeilen durch Eingabe von Leerzeichen oder Tabulatoren einzuruecken, wenn man diese Moeglichkeit anstelle des Einfuegens eines Kommandos (.IN oder .TI) bevorzugt.

10. Zentrieren und Unterstreichen

10.1. Zentrieren von Textzeilen

.CE	n	Zentrieren der naechsten n Textzeilen

Die Textzeilen werden zwischen linkem und rechtem Rand gemittelt. Wenn eine Zeile nicht passen sollte, beginnt sie am linken Rand und erreicht den rechten Rand.

Leerzeilen werden wie Textzeilen mitgezaehlt. Tabulatoren sollten vermieden werden. Fuehrende Leerzeichen werden beruecksichtigt.

Wenn man eine groessere Anzahl von Zeilen zentrieren moechte ohne zu zaehlen, kann man wie folgt vorgehen:

```
.CE 100
        Textzeilen
.CE 0
```

Nur die Zeilen zwischen diesen Kommandos werden zentriert, da das Kommando ".CE 0" das Zentrieren beendet.

10.2. Unterstreichen

Woerter, Buchstaben, Zeichen und Ziffern koennen im Fettdruck oder mit Unterstreichungen unter Verwendung des Standard- Steuerkodes erzeugt werden. Vorausgesetzt wird, dass der benutzte Drucker entsprechend ausgeruestet ist. Zum Unterstreichen kann in der folgenden Zeile das Ueberstreichungszeichen benutzt werden.

11. Erzeugen von Leerzeilen

11.1. Einfuegen von Leerzeilen durch Kommando

.SP n Einfuegen n Leerzeilen

Dieses Kommando erzeugt n Leerzeilen, es sie denn, die Seite ist vorher zu Ende. In diesem Fall werden die restlichen Zeilen nicht beachtet, d.h., sie entfallen. die Anzahl der erzeugten Zeilen n wird durch Veraenderungen im Zeilenabstandmodus nicht beeinflusst.

11.2. Uebernahme von Leerzeilen aus Eingabedatei

Leerzeilen werden mit in die Ausgabedatei uebernommen. Dadurch wird die Erzeugung von Leerzeilen ohne Verwendung des Kommandos (.SP) ermoeglicht.

Anders als bei der Anwendung des Kommandos (.SP) wird die Anzahl der Leerzeilen in der Ausgabedatei entsprechend des eingestellten Zeilenabstandsmodus verdoppelt, verdreifacht, usw.. Das ist nuetzlich fuer die Erzeugung von Leerzeilen zwischen Absaetzen.

Bei Erreichen des Seitenendes werden die restlichen Leerzeilen (anders als beim Kommando .SP) auf der naechsten Seite ausgegeben.

11.3. Leerzeilen am Seitenanfang

Das (SP)-Kommando erzeugt keine Leerzeilen am Seitenanfang. Wuenscht man solche, muss mindestens die 1. (Leer-) Zeile in der Eingabedatei tatsaechlich vorhanden sein.

12. Neue Zeilen und Seiten

12.1. Beginn einer neuen Zeile

.BR Beginn einer neuen Zeile

Manchmal ist es wuenschenswert, dass eine Zeile in der ausgabedatei als eine neue Zeile begonnen wird, obwohl die bisherige Zeile nicht aufgefuellt war. Dieses Kommando bewirkt, dass die naechste Textzeile als neue Zeile beginnt.

Zeilenunterbrechungen entstehen automatisch in einigen Faellen. Viele Kommandos bewirken eine Unterbrechung und erzwingen den Beginn einer neuen Zeile (Siehe "Bruch" in Befehlszusammenfassung). Zeilen, welche mit Leerzeichen beginnen, verursachen stets den Beginn auf einer neuen Zeile. Zeilenunterbrechungen sind nur im Fuellmodus wirksam.

12.2. Beginn einer neuen Seite

```
.BP n Beginn einer neuen Seite mit Nummer n
```

Dieses Kommando veranlasst, dass die folgende Textzeile auf einer neuen Seite beginnt; auch dann wenn auf der bisherigen Seite noch Platz war. Die folgende Seite wird fortlaufend nummeriert, falls die Angabe des Parameters n fehlt. Wird mit n nummeriert, folgt die Seite mit Nummer n. Man kann (.BP n) am Beginn des Textes benutzen, um die erste Seite abweichend von 1 zu nummerieren.

Zur Erzeugung von Leerseiten (z.B. fuer spaeteres Einfuegen von Zeichnungen usw.), die nur Seitenanfang und Seitenende haben, benutzt man folgende Befehlsfolge:

```
.BP
<Leerzeile>
.BP
```

Wenn die Leerzeile fehlt, wird die Seitenzahl zweimal erhoeht, aber keine Leerseite erzeugt.

13. Weitere Kommandos

13.1. Zusammenhaengender Text auf einer Seite

```
.NE      Platz fuer n Textzeilen wird
         benoetigt
```

Wenn man bestimmte Zeilen eines Textes zusammenhaengend auf einer Seite wuenscht, kann man dieses Kommando benutzen. Das kann auch den vorzeitigen Beginn einer neuen Seite erzwingen. Das Kommando bewirkt, dass mindestens n Druckzeilen auf der laufenden Seite verfuegbar sein muessen. Anderenfalls wird die naechste Textzeile auf den Beginn der neuen Seite verlegt. Das Kommando bewirkt eine Seitenunterbrechung, wenn weniger als n Zeilen auf der Seite verfuegbar sind.

13.2. Setzen von Tabulatoren

```
.TA n Tabulator im Intervall von n setzen
```

Die Tabulatoren erscheinen in den Spalten mit einem Vielfachen von n. Sie werden fuer Textteile verwendet, die im Nicht-Fuellmodus ausgegeben werden sollen sowie bei Absaetzen und Einrueckungen.

13.3. Unterbrechen der Ausgabe

.WA Warten auf den Druck einer Taste der Tastatur

die Ausgabe wird unterbrochen, bis eine Taste der Tastatur gedrueckt wird. die Ausgabe wird danach fortgesetzt mit dem Druck bis zum Ende des Dokumentes oder bis ein anderes Wartekommando wirksam wird. Das wird nach erfolgten Korrekturen und ihrer Ueberpruefung per Konsole benutzt.

Man beachte: Erfolgt die Ausgabe ueber ein anderes Geraet, wartet UFORM auch auf die Betaetigung einer Taste!

14. Behandlung von Spezialzeichen

14.1. Verwendung des Unterstreichungszeichens zur Unterdrueckung von Spezialfunktionszeichen

Zeichen, die dem Unterstriechungszeichen (_) folgen, werden unabhaengig von ihrer sonstigen Funktion gemaess ihrer "woertlichen" Bedeutung behandelt. Folgt dem Unterstreichungszeichen ein Zeichen ohne spezielle Bedeutung, so wird es wie ueblich ausgedruckt.

Beispiele:

_

Das folgende Leerzeichen wird nicht als Trennzeichen zwischen zwei Worten, auch nicht als entfernbarer Zwischenraum, gewertet. Dadurch kann man erreichen, dass zwei Woerter in einer Zeile oder eine Folge von Leerzeichen gedruckt werden.

_#

In Titelzeilen wird das Doppelkreuz (#) nicht durch die aktuelle Seitennummer ersetzt, sondern wie ein gewoehnliches Zeichen gedruckt.

_.

So kann am Anfang einer Zeile ein Punkt angegeben werden, ohne dass die Textzeile als Kommandozeile interpretiert wird.

__

Das Unterstreichungszeichen ist selbst ein Spezialzeichen. Um ein Unterstreichungszeichen zu drucken, muessen zwei Unterstreichungszeichen eingegeben werden.

14.2. Verwendung eines anderen Zeichens zu diesem Zweck

.EC x das Zeichen x wird anstelle des Unterstreichungszeichens verwendet

Wenn der Text viele Unterstreichungszeichen enthaelt, kann man fuer die Behandlung von Spezialzeichen ein anderes Zeichen festlegen. Dieses Zeichen kann ein beliebiges Zeichen, ausser Leerzeichen, Doppelkreuz und Punkt, sein.

15. Implementierungshinweise

Die maximal zulaessige Zeicheneingabe pro Zeile betraegt 256. die maximal zulaessige Zeichenzahl pro Ausgabezeile (einschliesslich der nicht-druckbaren Zeichen) betraegt 512. Der Minimalwert fuer die Kommando-Argumente>= 0 . Der Maximalwert fuer die Kommando-Argumente <=9999.
Um zu garantieren, dass eine Seite wenigstens 1 Zeile enthaelt, muessen die Argumente folgende Bedingung erfuellen.

Oberer Rand + Unterer Rand < Seitenlaenge

Eine andere Relation muss lauten:

Einrueckungswert + zeitweilige Einrueckung < rechter Rand

16. Zusammenfassung der Kommandos

16.1. Seitengestaltung

Kommando	Bruch	Standard	Funktion
.AF n	nein	2	Leerzeilen vor Fusszeile
.AH n	nein	2	" vor Kopfzeile
.BF n	nein	2	" nach Fusszeile
.BH n	nein	2	" nach Kopfzeile
.FO t	nein	Leerz.	Titelzeile (Fusszeile)
.HE t	nein	Leerz.	" (Kopfzeile)
.PL n	nein	60	Seitenlaenge
.PO n	ja	0	linker Rand
.RM n	ja	60	rechter Rand

16.2. Textgestaltung

Kommando	Bruch	Standard	Funktion
.LS n	nein	1	Zeilenabstand
.FI	ja	an	Fuellmodus
.FI	ja	aus	kein Fuellmodus
.JU	ja	aus	Justierung rechter Rand
.NJ	ja	an	keine Justierung

16.3. Weitere Kommandos:

Kommando	Bruch	Standard	Funktion
.BP n	ja	+1	neue Seite
.BR	ja		neue Zeile
.CE n	ja	1	Zentrierung
.EC x	nein		s.u. Spezialzeichen
.IN n	ja	0	Einruecken Block
.NE n	manchmal	0	Textzusammenfassung
.TA n	ja	8	Tabulatoren setzen
.TI n	ja	5	Einruecken 1 Zeile
.SP n	ja	1	Leerzeilen erzeugen
.WA	nein		Unterbrechen der Ausgabe

Teil 5

R A B U G

Symbolisches Fehlersuchprogramm

Inhaltsverzeichnis Seite

1. Einfuehrung

RABUG ist ein symbolisches Fehlersuchprogramm (Debugger) fuer U880-Assemblerprogramme. Es ist weit leistungsfaehiger als der UDOS - PROM-DEBUGGER. Seine wesentlichen Merkmale sind:

- Bis zu 8 Haltepunkte koennen gesetzt werden. (Jeweils mit Durchlaufzaehler, mit dem festgelegt werden kann, wie oft ein Haltepunkt durchlaufen werden soll, bevor er angezeigt wird.)

- Arbeit im Schrittbetrieb oder Ausfuehrung einer vorgegebenen Anzahl von Schritten.

- Register- und Speicherinhalte koennen in HEX8-, HEX16- und in ASCII-Kode gelesen und beschrieben werden.

- Die Arbeit mit verschieblichen Modulen ist ohne manuelle Adressberechnung moeglich, d.h., es kann auch mit relativen Adressen gearbeitet werden.

- Es steht ein maskierbarer Suchbefehl zur Verfuegung, der einen bezeichneten Speicherbereich nach einem vorgegebenen Muster (max. 4 Bytes) durchsucht.

- Die Verwendung von Marken aus dem Quellenprogramm ist moeglich (Schnittstelle zur Assemblersymboltabelle).

- Kommandos sind 1 Zeichen lang. Es ist kein abschliessendes CR noetig.

- Als Besonderheit verfuegt RABUG ueber die Moeglichkeit, U880-Befehle zu assemblieren und zu reassemblieren, d.h., Maschinenkode-Speicherinhalt kann unmittelbar als Befehl zur Anzeige gebracht werden. Umgekehrt ist die direkte eingabe von Befehlen in ein Maschinenkode-Programm moeglich.

- RABUG ist ca. 8 KByte lang. Dazu werden ca. 1 KByte fuer die User-Symboltabelle (fuer je 30 Seiten Text) benoetigt.

- Die User-Symboltabelle wird immer unmittelbar im Abschluss an RABUG geladen.

2. Vereinbarungen

RABUG verwendet verschiedene spezielle Zeichen. In den anschliessenden Beispielen werden sie wie folgt dargestellt:

^<Zeichen>	bedeutet Control-Taste (CTRL) + <Zeichen>
$	bedeutet Escape-Taste (ESC), sofern nicht angegeben ist, dass tatsaechlich die $-Taste gemeint ist.
CR	bedeutet Wagenruecklauf-Taste
LF	bedeutet Zeilenschaltung-Taste
ESC	bedeutet Escape-Taste (ESC)
DEL	bedeutet Delete-Taste
*	RABUG's Bereitschaftszeichen (leitet die meisten Beispiele ein)

Um klarzumachen, wer bei einem Eingabedialog was geschrieben hat, wird die Eingabe durch den Anwender in dieser Weise unterstrichen. Ausgaben von RABUG werden in normaler Schreibweise dargestellt.

3. RABUG - Aufruf, Eintritt und Austritt

Im Gegensatz zum PROM-Debugger muss RABUG explizit in den Speicher geladen werden.
Es werden zwei RABUG-Versionen zur Verfuegung gestellt:

RABUG40 (4000H - 60FFH)
RABUGC0 (C000H - E0FFH)

Um das User-Programm "EXAMPLE" mit RABUG40 auszutesten, muesste das Kommando

%EXAMPLE,RABUG40

eingegeben werden. Es laedt beide Programme und startet RABUG40, das sich mit dem Bereitschaftszeichen "*" meldet. Fuer die weitere Vorgehensweise kann die RABUG-Kommandouebersicht (Anhang) verwendet werden.

Wird ein Anwenderprogramm gestartet, in dem Unterbrechungspunkte gesetzt wurden, so geht die Steuerung an RABUG zurueck, sobald ein Unterbrechungspunkt erreicht wird. Wird waehrend des Laufes eines Anwenderprogramms die NMI-Taste betaetigt, geht die Steuerung ebenfalls an RABUG zurueck (hilfreich, wenn sich z.B. das Anwenderprogramm in einer endlosen Schleife befindet).

Austritt:

Die Steuerung kann an UDOS zurueckgegeben werden mit dem ^Q-Kommando:

```
*^Q
%          (Steuerung ist an UDOS zurueckgegangen)
```

Alle Unterbrechungspunkte sind geloescht.

Anwendersymbole:

Es ist moeglich, in RABUG mit Symbolen aus dem Quellprogramm zu arbeiten. Als Vorraussetzung dafuer muessen beim Assemblieren und beim Linken bestimmte Optionen angegeben werden. Sie sorgen dafuer, dass eine binaere Datei angelegt wird, die die Symboltabelle enthaelt.

Das Beispiel zeigt die Vorgehensweise:

```
%ASM MOD1 (S)
%ASM MOD2 (S)
%PLINK $=4400 MOD1 MOD2 (SY)
```

Die S-Option im ASM-Kommando veranlasst den Assembler, die Symbole an die binaere Datei anzuhaengen, so dass der Linker sie in einer binaeren Symboltabellendatei zusammenfassen kann. Die SY-Option im Linkkommando sorgt dafuer, dass diese besagte Datei erzeugt wird (mit der Namenserweiterung ".SYM"). Dieser Dateiname kann dann in RABUG in das E-(environment) Register geholt werden (s. Abschn. 4).

4. Ausdruecke, Symbole und Displacements

Viele Eingaben fuer RABUG sind Ausdruecke. Jeder Ausdruck kann aus den unten beschriebenen Elementen bestehen. Die Elemente eines Ausdrucks koennen in unterschiedlichen Eingabemodi eingegeben werden. Die einzelnen Elemente koennen durch die unten angegebenen Operationszeichen miteinander verbunden werden.

Elemente in Ausdruecken:

Jedes Element hat einen 16-Bit-Zahlenwert. Ob der Wert als 16-Bit-Wert behandelt wird oder nicht, geht aus dem Zusammenhang hervor. Bei der Berechnung von Ausdruecken wird jedoch 16-Bit-Arithmetik verwendet.

Gueltige Elemente sind:

<hex number> Die letzten Zahlen des eingegebenen Wertes werden verwendet. Fuer A-F koennen grosse oder kleine Buchstaben verwendet werden.

<hex number>' Die letzten vier Zahlen des eingegebenen Wertes werden zum Inhalt des D-Registers addiert und dieser Wert wird dann verwendet. Diese Form ist nuetzlich fuer die Anfaben von Adressen in verschieblichen Modulen. Dazu wird das D-Register (siehe unten) auf die Startadresse des Moduls gesetzt. Dann verwendet man fuer die Eingabe die Adressen aus dem Assemblerlisting, die man mit dem Zeichen "'"(Hochkomma) versieht, damit daraus die richtige absolute Adresse gebildet wird.

$<symbol> Das <symbol> wird in der RABUG-Symboltabelle aufgesucht und der entsprechende Wert wird anstelle des Symbols verwendet. Weiter unten ist beschrieben, wie der Zugriff auf die Programmsymbole erfolgt. (Wie vereinbart, wird $ bei dieser Eingabe durch die ESC-Taste erzeugt.)

'<character> Der ASCII-Wert fuer <character> wird verwendet.

$(wirklich $) Als Wert wird die Adresse der zuletzt geoeffneten Speicherzelle verwendet.

% Als Wert wird der Inhalt der zuletzt geoeffneten Speicherzelle oder des Registers oder der zuletzt durch das "="-Kommando berechnete Ausdruck verwendet.

Den Elementen kann ein einzelnes "+" oder "-"-Vorzeichen vorangestellt sein, und sie koennen miteinander durch die Operationszeichen "+", "-", "*" (multiplizieren) und "/" (dividieren) verknuepft werden. Ausdruecke werden von links nach rechts berechnet. Dabei werden alle Operationszeichen gleichwertig behandelt.

Laden der Symboltabelle:

vorausgesetzt, dass in der in 3. beschriebenen Weise eine binaere Symboltabelle angelegt wurde, koennen mit den ^E- und ^L-Kommandos die Symbole geladen werden.
Die Symboltabelle wird, an RABUG unmittelbar anschliessend, in den Speicher geladen. Es ist deshalb nicht ratsam, anschliessend an RABUG Programmkode oder Daten im Speicher zu haben, wenn man mit den Kommandos fuer die Symboltabelle arbeiten moechte. Es wird auch keine Kontrolle ausgefuehrt, die verhindert, dass die Symboltabelle ueber das physische Ende des Speichers hinausreicht.

Das ^W-Kommando gibt Auskunft ueber den von RABUG und den von der aktuellen Symboltabelle belegten Speicherbereich.
RABUG arbeitet ohne Verwendung des UDOS-Memory-Managers. Deshalb ist eine manuelle Zuweisung des fuer die Symboltabelle notwendigen Speicherplatzes ratsam (und gewoehnlich auch notwendig).

Der Name der Symboldatei wird RABUG ueber das ^E (environment)-Kommando mitgeteilt. Nach Eingabe von "^E" wird der Name der aktuellen Symbol-Datei (falls schon eine in den Speicher geladen wurde) ausgegeben. anschliessend kann der Name einer neuen Symboldatei eingegeben werden. Der Name muss o h n e den Zusatz ".SYM" eingegeben werden.

```
*^E   BASIC
*^E   BASIC   NEWPROG
*
```

In diesem Beispiel wird zuerst die Symboldatei BASIC.SYM angegeben. Dann wird die Datei NEWPROG.SYM ausgewaehlt. Nachdem dieses Kommando zur Anwendung kam, sind die globalen Symbole und Modulnamen in die RABUG-Symboltabelle geladen. Von den lokalen Symbolen sind nur die geladen, die zu dem Modul gehoeren, der den Namen der Symboldatei traegt. Gibt es keinen solchen Modul, wird ein Fragezeichen angegeben.
Die globalen Symbole und die Modulnamen werden jedoch immer geladen.

Die lokalen Symbole eines bestimmten Moduls werden geladen, indem man mit dem ^L (local)-Kommando den Namen dieses Moduls eingibt:

```
*^L   INFORM
*^L   INFORM   SCANNER
*
```

Hier wurden zuerst die lokalen Symbole des Moduls INFORM geladen und anschliessend die des Moduls SCANNER. Dabei werden die Symbole des vorhergehenden Moduls ueberschrieben. Man hat also immer nur die Symbole eines Moduls verfuegbar.

BEACHTE: RABUG benutzt die Unit 20 von UDOS, um die Symboltabelle zu laden. Deshalb sollen Anwenderprogramme diese Unit nicht verwenden.

Die Grenzen von RABUG und der aktuellen Symboltabelle koennen mit dem ^W-Kommando abgefragt werden.

```
*^W   4000   5F5F   6148
```

Hier belegt RABUG den Speicherplatz von 4000 bis 5F5F und die aktuelle Symboltabelle belegt den Bereich von 5F5F bis 6148. Man sollte sorgfaeltig darauf achten, dass die Symboltabelle nichts ueberschreibt oder ueber das physische Speicherende hinausgeht.

Reservieren von Speicherplatz fuer die Symboltabelle:

Man schaetzt die Groesse der Symboltabelle grob ab, indem man fuer 20 Seiten Quellkode etwa 1 KByte rechnet. Beim Generieren der RABUG-Versionen RABUG40 und RABUGCO wurden jeweils 4 KByte fuer den Stack vereinbart, die im Anschluss an das Programm reserviert werden. RABUG benoetigt diesen

Stack nicht, so dass der auf diese Weise reservierte Speicherplatz fuer die Symboltabelle zur Verfuegung steht. Im Bedarfsfall kann ueber eine ALLOCATE-Anweisung zusaetzlicher Speicherplatz reserviert werden.

Das Displacement-Register:

Das D-Register wird fuer zwei Zwecke genutzt:

1. Es liefert einen Ausgangswert fuer Zahlen, die als relative Adressen eingegeben werden (kenntlich gemacht durch "'" (Hochkomma) hinter der Zahl).

2. Es liefert einen Ausgangswert fuer Adressen, die von RABUG als relative Adressen ausgegeben werden sollen.

Soll eine Adresse ausgegeben werden und das D-Register ist ungleich Null, wird die Symboltabelle durchsucht, um ein Symbol zu finden, das den Wert der auszugebenden Adresse hat. Wird eines gefunden, so wird der Symbolname ausgegeben. Andernfalls wird die Adresse als 16-Bit-Hexadezimalzahl ausgegeben, falls sie kleiner ist, als der im D-Register stehende Wert. Ist sie nicht kleiner, wird nur die "Verschiebung“ zum D-Register ausgegeben, d.h., der im D-Register stehende Wert wird vorher von der Adresse abgezogen. Man erhaelt die relative Adresse. Auf diese Weise werden relative Adressen niemals als negative Zahlen ausgegeben. Setzt man das D-Register auf -1, so erreicht man, dass zwar die Symboltabelle durchsicht wird, aber es wird keine relative Adresse ausgegeben.

Zum Setzen des D-Registers wird das ^D-Kommando verwendet:

```
*^D   FFFF   CR
*^D   FFFF  $MODB    CR
*
```

Zuerst wurde das D-Register nur geoeffnet durch Eingabe des Kommandos "^D". Es wurde aber nicht modifiziert, sondern durch CR wieder geschlossen. Dann wurde es erneut geoeffnet und der Wert des Symbols MODB (hier der Name eines Moduls) wurde eingetragen. Eine vollstaendige Beschreibung ueber das Oeffnen von Registern folgt in den naechsten beiden Abschnitten.

5. Speicherkommandos

Speicher und Register koennen in verschiedenen Ausgabemodi angezeigt werden. Moeglich sind:

HEX8	8 Bit hexadezimal
HEX16	16 Bit hexadezimal
DHEX16	16 Bit hexadezimaler Offset zum D-Register, wie schon beschrieben wurde

ASCII 7 Bit ASCII-Zeichen
QUIET keine Ausgabe
INSTRUCTION Instruction-Mnemonik

RABUG befindet sich immer in einem "aktuellen" Ausgabemodus, naemlich dem, der als letzter festgelegt wurde. Der Ausgabemodus kann auch explizit durch eines der folgenden Kommandos festgelegt werden:

*. HEX8-Modus
*: HEX16-Modus
*[DHEX16-Modus
*(ASCII-Modus
*! QUIET-Modus
*; INSTRUCTION-Modus

Diese Zeichen werden auch in Verbindung mit einem oder zwei Parametern verwendet, um einen bezeichneten Speicherplatz zu oeffnen oder einen Speicherbereich auszugeben.

Das Oeffnen eines Registers ist dem Oeffnen einer Schachtel vergleichbar: man kann den Inhalt untersuchen und/oder veraendern, wenn die Schachtel offen ist. Aber es ist nicht moeglich, wenn sie verschlossen ist. Wird ein Speicherplatz geoeffnet, so wird der Inhalt in dem Modus angezeigt, der durch das Oeffnungskommando gewaehlt wurde. Oder es wird im aktuellen Modus angezeigt, falls das Kommando, das den Speicherplatz eroeffnete, keinen Modus auswaehlt. Dann kann optional ein Ausdruck eingegeben werden, der den Inhalt des Speicherplatzes ersetzen soll. diesem Ausdruck muss eines der nachstehenden Zeichen folgen:

CR schliesst den Speicherplatz (und ersetzt den Inhalt, falls ein neuer Inhalt eingegeben wurde)
LF schliesst den Speicherplatz wie CR, aber oeffnet dann den naechsten
^ schliesst den Speicherplatz wie CR, aber oeffnet dann den vorhergehenden ("^" hier nicht CRTL-Taste, sondern Taste "Pfeil nach oben"!)

. zeigt den Speicherplatz noch einmal an, aber im HEX8-Modus
: zeigt den Speicherplatz noch einmal an, aber im HEX16-Modus
(zeigt den Speicherplatz noch einmal an, aber im ASCII-Modus
[zeigt den Speicherplatz noch einmal an, aber im DHEX16-Modus
: zeigt den Speicherplatz noch einmal an, aber im INSTRUCTION-Modus

Der Inhalt eines Speicherplatzes wird nicht veraendert, wenn eine Fehlernachricht "??" ausgegeben wurde.

Speicherplaetze koennen in einem der angegebenen Modi durch folgendes Kommando eroeffnet werden:

*no

Dabei bezeichnet n den Speicherplatz (Adresse oder entsprechende Marke) und c ist eines der Zeichen:

"." , ":", "(", "!", "[" oder ";"

Werden LF oder ^ als Einzelkommandos verwendet, oeffnen sie den naechsten oder vorhergehenden Speicherplatz im aktuellen RABUG-Modus (ausgehend von dem Speicherplatz, der als letztes geoeffnet war).

Speicherbereich ausgeben:

Ein Speicherbereich kann durch eines der folgenden Kommandos ausgegeben werden:

*n,m.
*n,m(
*n,m:

"n,m." und "n,m(" erzeugen Ausgaben (dumps) in HEX8- und ASCII-Modus (kombiniert), beginnend beim Speicherplatz n bis zum Speicherplatz m.
"n,m:" erzeugt eine Ausgabe vom Speicherplatz n bis zu m im HEX16-Modus.

6. Assemblierung und Reassemblierung

Das Befehlsmnemonik-Ausgabeformat (INSTRUCTION-Modus) wird durch das Kommando: ";" ausgewaehlt. Das ";" kann benutzt werden, um einen Speicherplatz zu oeffnen oder einen Wert erneut anzuzeigen. (In der gleichen Weise, wie das auch mit ".", ":", "(" und "[" der Fall ist.) Ist der anzuzeigende Wert kein gueltiger Befehl, wird er im HEX8-Modus angezeigt.
"LF" erhoeht den Speicherplatzzaehler um die Anzahl der Bytes, die dem angezeigten Befehl entsprechen.
"^" erniedrigt ihn um 1 Byte (ungeachtet der Laenge des Befehls).
Ist ein Speicherplatz einmal geoeffnet, kann ein U880-Befehl eingegeben werden. Die Anzahl der geschriebenen Bytes, wie auch die Anzahl von Bytes, um die der Speicherplatzzaehler bei Eingabe des LF-Kommandos erhoeht wird, haengt von der Laenge des Befehls ab. In einen geoeffneten Speicherplatz koennen immer Befehle eingetragen werden, unabhaengig vom Ausgabeformat, das ausgewaehlt wurde, um ihn anzuzeigen.

Hinweis zur Befehlsassemblierung:

Es gibt einige Unterschiede zwischen dem RABUG- und dem UDOS-Assembler:

1. Sowohl Leerzeichen als auch Kommas gelten als Trennungszeichen zwischen Feldern.

2. Alle Zahlen werden als Hexadezimalzahlen betrachtet.

3. Zahlen muessen nicht mit einer Ziffer beginnen, sie werden jedoch als Registernamen interpretiert, sofern es moeglich ist. Zum Beispiel:

```
Ld  B,A   Lade Register B mit Register A
Ld  B,0A  Lade Register B mit A (hexadezimal)
```

4. IM0, IM1 und IM2 muessen ohne Leerzeichen eingegeben werden.

5. Jedes Symbol, das verwendet wird, muss mit ESC ($) beginnen, gemaess der Handhabung von Symbolen in RABUG.

7. Unterbrechungspunkte, CPU-Register und Schrittkommandos

Die Strategie von RABUG besteht darin, sich zwischen zwei Befehlen so einzuschalten, dass zwischen diesen Befehlen Register und Speicher betrachtet und/oder modifiziert werden koennen und eine Auswertung vorgenommen werden kann, ob das Programm richtig arbeitet oder nicht.

RABUG erlaubt diese Art der Fehlersuche, indem es die Moeglichkeit schafft, bis zu 8 Unterbrechungspunkte im Anwenderprogramm zu setzen, das Anwenderprogramm schrittweise abzuarbeiten, wobei ein oder mehrere Befehle auf einmal ausgefuehrt werden koennen und indem es den CPU-Status beim Eintritt in RABUG rettet und ihn beim Austritt aus RABUG wieder herstellt.

Register:

Jedesmal, wenn ein Eintritt in RABUG erfolgt, wird der Inhalt aller Register gerettet. Die Werte stehen zur Ueberpruefung und Modifizierung zur Verfuegung. Die Register koennen in aehnlicher Weise wie die Speicherplaetze angezeigt oder geoeffnet werden.

Das ^R-Kommando loest die Anzeige aller Register aus:

```
*^R
```

Einzelne Register koennen geoeffnet werden durch Angabe des Registernamens und nachfolgendem ^R-Kommando:

```
*B^R 04
```

Hier wird das Register B geoeffnet und der Wert (04) wird im HEX8-Modus angezeigt. Ist ein Register einmal geoeffnet, kann optional ein Ausdruck eingegeben werden, der den enthaltenen Wert ersetzt. Anschliessend folgt ein CR, mit dem das Register geschlossen wird.

Nur die untersten 8 Bits eines Wertes, der in ein 8-Bit-Register eingegeben wird, werden genutzt. Die Registernamen sind:

```
$A $B $C $D $H $L $F $A' $B' $C' $D' $E' $H' $L' $F'
$SP $IX $IY $PC
```

wobei $SP das Stackpoint-Register und $PC der Befehlszaehler ist.

Schrittbetrieb:

Ein oder mehrere Befehle (beginnend bei PC) koennen ausgefuehrt werden (wobei die Steuerung anschliessend zurueck an RABUG geht), indem das ^S-Kommando verwendet wird (STEP).

Es gibt zwei Formen:

```
*^S     Einzelschritt
*n^S    die naechsten n Befehle werden ausgefuehrt
            (n kann ein Ausdruck sein)
```

Nachdem die eingegebene Anzahl von Befehlen ausgefuehrt wurde, geht die Steuerung an RABUG zurueck. Der Inhalt der Register kann optional angezeigt werden (siehe unten).

Unterbrechungspunkte:

Unterbrechungspunkte werden auf das erste Byte eines Befehls gesetzt, der den weiter unten angegebenen Einschraenkungen genuegen muss. Wenn dieser Befehl ausgefuehrt wird, geht die Steuerung an RABUG, von wo sie wieder an das Nutzerprogramm gegeben werden kann, nachdem die Unterbrechung und die entsprechende Adresse angezeigt wurden.

Um groessere Flexibilitaet zu erreichen, wenn Unterbrechungspunkte in Programmschleifen verwendet werden, ist jedem Unterbrechungspunkt ein Durchlaufzaehler zugeordnet. Jedesmal, wenn der Unterbrechungspunkt erreicht wird, wird der Durchlaufzaehler dekrementiert. Ist der Wert ungleich Null, geht die Steuerung zurueck an das Nutzerprogramm. Erreicht der Zaehler die Null, zeigt RABUG den Unterbrechungspunkt an.

Die Adressen der Unterbrechungspunkte werden in den B-Registern gespeichert, die vorgegebenen Werte der Durchlaufzaehler in den N-Registern (Count-Register) und die heruntergezaehlten Werte in den K-Registern (Count-Down-Register).
Jeweils die gesamte Gruppe dieser Register kann angezeigt werden mit:

*^B	Anzeige der Unterbrechungspunkt-Adressen-Register
*^N	Anzeige der Durchlaufzaehler-Count-Register
*^K	Anzeige der Durchlaufzaehler-Count-Down-Register

Die Adressen der Unterbrechungspunkte werden durch die entsprechenden Symbole angezeigt, sofern welche vorhanden sind. Ansonsten erfolgt die Anzeige im HEX16-Modus.

Um einen Unterbrechungspunkt auf den Speicherplatz n zu setzen, ist das Kommando:

*n^B

erforderlich (n kann ein Symbol oder ein Ausdruck sein). Sollte dies schon der neunte, gleichzeitig eingegebene Unterbrechungspunkt sein, wuerde eine Fehlermeldung folgen. Jedem Unterbrechungspunkt ist eine Zahl zugeordnet (wird durch das ^B-Kommando auch zur Anzeige gebracht). RABUG verwendet diese Zahl, um das Auftreten eines bestimmten Unterbrechungspunktes anzuzeigen. Der Nutzer verwendet sie, um einen bestimmten Unterbrechungspunkt zu loeschen.

*^X	Loeschen aller Unterbrechungspunkte
*n^X	Loeschen des Unterbrechungspunktes n (n=0-7)

Die Unterbrechungspunkt-Durchlaufzaehler-Count- und Count-Down-Register koennen durch folgende Kommandos geoeffnet und soomit auch veraendert werden:

*n^N	Oeffnen des N-Registers fuer den Unterbrechungspunkt Nr. n
*n^K	Oeffnen des K-Registers fuer den Unterbrechungspunkt Nr. n

Steuerung der Programmabarbeitung:

Das Anwenderprogramm kann auf verschiedene Weise gestartet (oder fortgesetzt) werden:

(1) Start ab einer bestimmten Adresse:

*n^G	Die Abarbeitung beginnt bei der Adresse n (GO)

(2) Start ab aktuellem Wer des Befehlszaehlers (PC):

*^G	Die Abarbeitung beginnt beim aktuellen Befehlszaehlerstand

(3) Von einem Unterbrechungspunkt an fortsetzen:

*^P Fortsetzen (proceed)

(4) Von einem Unterbrechungspunkt an fortsetzen und gleichzeitig das Durchlaufzaehler-Count-Down-Register (K-Register) setzen:

*n^P Fortsetzen und das Count-Down-Register fuer den zuletzt erreichten Unterbrechungspunkt auf den Wert n setzen

Steuerung der Registeranzeige:

Normalerweise wird, wenn die Steuerung nach einem Programmschritt oder nach einem Unterbrechungspunkt an RABUG zurueckgeht, nur die Adresse des naechsten abzuarbeitenden Befehls angezeigt. Manchmal ist es jedoch wuenschenswert, zu diesem Zeitpunkt auch die CPU-Register angezeigt zu bekommen. Diese optionale Anzeige wird durch ein Ein-Byte-Register, $RSWITCH, gesteuert. Hat dieses Register den Wert 1, werden dir CPU-Register, jedesmal wenn RABUG die Steuerung uebernommt, angezeigt.
Hat es den Wert 0, wird die Anzeige unterdrueckt.

*$RSWITCH. 00 1 Durch Eingabe der 1 wurde die Anzeige freigegeben

Einschraenkungen:

Unterbrechungspunkte duerfen n i c h t gesetzt werden auf:

1. ein anderes als das erste Byte eines Befehls;
2. einen Befehl, der modifiziert wird;
3. einen Befehl, der auch als Data verwendet wird;
4. einen Befehl innerhalb von RABUG;
5. einen Speicherplatz, der sich in nicht modifizierbaren Speicher (PROM, RUM usw.) befindet;
6. einen Speicherplatz, der auf einen nicht modifizierbaren Speicherplatz folgt;
7. den Speicherplatz FFFFH oder
8. einen Befehl, der nicht den unten angegebenen Einschraenkungen fuer den Schrittbetrieb genuegt, da der Befehl, auf den der Unterbrechungspunkt gesetzt ist, im Schrittbetrieb abgearbeitet wird.

Ausserdem erhaelt man anormale Ergebnisse, wenn der Befehl, auf den der Unterbrechungspunkt gesetzt ist, einen Bezug zum unmittelbar vorhergehenden Speicherplatz hat. Der Grund dafuer ist, dass der dem Unterbrechungspunkt vorausgehende Befehl veraendert ist, wenn der zu untersuchende Befehl abgearbeitet wird und erst danach wieder hergestellt wird.

Der Schrittbetrieb darf n i c h t zur Anwendung kommen, wenn:

1. der Speicherplatz, der dem im Schrittbetrieb abzuarbeitenden Befehl vorausgeht, sich in nicht modifizierbarem Speicher (ROM, PROM, nicht existierender Speicher usw.) befindet;

2. der im Schrittbetrieb abzuarbeitende Befehl auf den vorhergehenden Speicherplatz als Datenwort Bezug nimmt;

3. es sich bei dem im Schrittbetrieb abzuarbeitenden Befehl um einen der folgenden Befehle handelt:

   ```
   IM 0
   IM 1
   LD I,A   mit A ungleich 0FH
   ```

   (der Grundgedanke hierfuer ist, dass am Ende des Befehls ein Interrupt auftreten wird. Sind nun die Interruptbedingungen fehlerhaft, so wird das auch mit RABUG der Fall sein.)

Wird ein DI-Befehl im Schrittbetrieb abgearbeitet, geht die Steuerung so lange nicht an RABUG zurueck, bis ein Befehl der einem EI-Befehl folgt, abgearbeitet wird.
Soll ein EI-Befehl abgearbeitet werden, wird auch noch der dem EI folgende Befehl ausgefuehrt, bevor die Steuerung an RABUG zurueckgeht.

8. Durchsuchen und Fuellen des Speichers

Suchen:

RABUG bietet die Moeglichkeit, den Speicher nach speziellen Bitmustern abzusuchen. Werte bis zur Groesse von 4 Bytes koennen gesucht werden. Der Suchprozess laeuft folgendermassen ab:
Jede Speicherzelle in dem bezeichneten Bereich wird getestat, indem 4 Bytes, beginnend bei der betrachteten Speicherzelle, geladen werden. Diese 4 Bytes werden mit den 4 Bytes des Maskregisters "und"-verknuepft und dann mit den 4 Bytes des Wordregisters verglichen. Bei Uebereinstimmung werden die Speicheradresse und der Inhalt im aktuellen Ausgabemodus ausgegeben. Dann wird der Prozess wiederholt, indem, von der naechstfolgenden Speicherzelle ausgehend, wieder 4 Bytes betrachtet werden.
Auf diese Weise kann z.B. nach Ein-, Zwei-, Drei- oder Vier-Byte-Befehlen oder auch nach Zwei-Byte-Adressen gesucht werden.
Das Mask- und das Wordregister koennen wie eine Speicherzelle gesetzt werden. Ueber die Symbole "$MASK" und "$WORD" werden sie geoeffnet. Die Eingabe erfolgt wie fuer eine normale Speicherzelle. Es ist darauf zu achten, dass jeweils nur 4 Bytes, beginnend bei den genannten Symbolen,

veraendert werden, da die Speicherzellen innerhalb von RABUG liegen.
Die Suche wird mit dem Kommando:

*n,m^S<

ausgeloest. Der Speicherbereich von der Adresse n bis zur Adresse m wird in der beschriebenen Weise durchsucht.

Als Beispiel soll nach einem HALT-Befehl (76H), dem eine 1 folgt, gesucht werden:

```
*$MASK!   -1  LF
XXXX      -1  LF
XXXX       0  LF
XXXX       0  CR
*$WORD!   76  LF
XXXX       1  LF
XXXX       0  LF
XXXX       0  CR
*4000,5000^S        Es wird im Speicherbereich 4000 -
                    5000 gesucht
```

Fuellen des Speichers:

Eine Anzahl aufeinander folgender Speicherplaetze kann in folgender weise auf einen Wer gesetzt werden:

*n,m^Z	Setzt die Speicherplaetze von n bis m auf Null
*n,m,k^Z	Fuellt die Speicherplaetze von n bis m mit dem Wert k

Ein Ueberschreiben von RABUG durch diese Kommandos ist nicht moeglich

Beispiel:

*4000,5000,'X^F	Fuellt die Speicherplaetze von 4000 bis 5000 mit dem ASCII-Zeichen "X"

9. Einschraenkungen und ihre Konsequenzen

Bei der Anwendung von RABUG koennen ein paar Besonderheiten auftreten, die man vielleicht auch als Fehler ansehen koennte. Nachfolgend geben wir eine Liste von ihnen an, auch, wie man dem begegnen kann.

1. Die Verwendung von Modulnamen, die das Zeichen "." enthalten, verursacht Probleme, da "." ein RABUG-Kommando ist. Deshalb darf solch ein Name in keinem anderen Zusammenhang als in Bezug auf das ^E- oder das ^L-Kommando verwendet werden.

Loesung:

1. Man verwende keinen "." in Modulnamen.
2. Man verwende in solch einem Modul ein zusaetzliches globales Symbol, dessen Wert der Adresse des 1. Bytes des Moduls entspricht und verwende es anstelle des Modulnamens.

2. Fuer eine erfolgreiche Suche in der Symboltabelle muss eine Uebereinstimmung mit allen eingegebenen Zeichen bestehen. Es muss dabei bedacht werden, dass der Assembler Namen auf 6 Zeichen verkuerzt. Deshalb muss man sich z.B. auf das Symbol

```
SEARCHTBL:  ... ,
```

das in einem Programm verwendet wurde, nur mit $SEARCH beziehen.
Modulnamen werden nicht verkuerzt.

3. Dem im Zusammenhang mit dem ^E- und ^L-Kommando anzugebenden Modulnamen wird kein ESC ($) vorangestellt.

4. Es werden einige unzulaessige Befehle assembliert und reassembliert, ohne dass ein Fehler angezeigt wird. Dazu gehoeren insbesondere:

Assemblierung:

- Die Verwendung von IX/IY in 2 Feldern (z.B. LD (IX) (IY))
- Die Verwendung von IX/IY in einigen Befehlen, fuer die das nicht zulaessig ist.
- Der eingegebene Befehl OUT (C),A wird faelschlich als OUT (0C),A notiert (wird wie OUT (n),A betrachtet).

Reassemblierung:

- Befehle, die wie IX/IY-Befehle beginnen, aber die IX oder IY nicht verwenden (beginnend mit DD oder FD).

Anhang: RABUG - Kommandouebersicht

Vereinbarungen:

^<chr>	Bedeutet Zeichen gemeinsam mit CONTROL-Taste
m.m.k	Sind Zahlenwerte (s. unten).
$	Ist ESC-Taste, wenn nicht anders angegeben

Fehlernachrichten:

OVF ?	Bedeutet, dass ein Wert zu gross fuer den Inhalt ist
UND ?	Bedeutet ein undefiniertes Symbol
??	Bedeutet: "Nicht ausfuehrbar!"

Kommandos ohne Argument:

^B	Liste die Breakpoints auf
^D	Oeffne das Displacement-Register (s. Abschn. 4)
^E	Oeffne das Symboldatei-Namensregister
^G	Gehe zur Adresse im PC (geht aus RABUG in User-Programm)
^K	Liste die Breakpoint-Count-Down-Register auf (in den Count-Down-Registern werden die in den Count-Registern vorgegebenen Werte heruntergezaehlt)
^L	Oeffne das Modulnamensregister fuer lokale Symbole
^N	Liste die Breakpoint-Count-Register auf (ein Count-Register gibt an, wie oft ein Breakpoint durchlaufen werden soll, bevor er angezeigt wird)
^P	Setze das Programm ab Breakpoint fort
^Q	"QUIT", Rueckkehr zu UDOS
^R	Liste die Register auf
^S	Schritt, fuehre einen Befehl aus
^W	Wo ? Liste die Start und Endadresse von RABUG und die Endadresse der Symboltabelle aus
^X	Loesche alle Breakpoints
^Z	Wie ^P
!	Setze QUIET-Ausgabemodus (keine Ausgabe)
.	Setze HEX8-Ausgabemodus
:	Setze HEX16-Ausgabemodus
(	Setze ASCII-Ausgabemodus
[	Setze DHEX16-Ausgabemodus (16 Bit Hexadezimal-Offset zum D-Register fuer die Ausgabe von relativen Adressen, s. Pkt. 4)
;	Setze INSTRUCTION-Ausgabemodus
LF	Oeffne den naechsten Speicherplatz
^	Oeffne den vorhergehenden Speicherplatz

Kommandos mit einem Argument:

n^B	Setze Breakpoint auf die Speicheradresse n
n^G	Beginne die Programmabarbeitung bei der Speicheradresse n
n^K	Oeffne das Breakpoint-Count-Down-Register n (n=0-7)
n^N	Oeffne das Breakpoint-Count-Register n (n=0-7)
n^P	Setze das Programm ab Breakpoint fort und setze das Breakpoint-Count-Down-Register auf n
n^R	Oeffne das Register n (n=0-20 oder $A, $B, ..., $a', ..., $F', $SP, $IX, $IY, $SP, $I)
n^S	Fuehre die naechsten n Befehle aus
n^X	Loesche den Breakpoint Nr. n (n=0-7)
n!	Oeffne die Speicherzelle n, ohne Ausgabe
n.	Oeffne die Speicherzelle n, im HEX8-Ausgabemodus
n:	Oeffne die Speicherzelle n, im HEX16-Ausgabemodus
n(	Oeffne die Speicherzelle n, im ASCII-Ausgabemodus
n[	Oeffne die Speicherzelle n, im DHEX16-Ausgabemodus (16 Bit hexadezimaler Offset zum D-Register
n;	Oeffne die Speicherzelle n im INSTRUCTION-Ausgabemodus
n=	Schreibe n im HEX16-Ausgabemodus (berechnet Ausdruecke)

Kommandos mit zwei Argumenten:

n,m^S	Durchsuche den Speicher von n bis m
n,m^Z	Fuelle den Speicher von n bis m (einschliesslich) mit Null
n,m.	Drucke "DUMP" des Speichers von n bis m im HEX8- und ASCII-Ausgabemodus
n,m(	wie n,m.
n,m:	Drucke "DUMP" des Speichers von n bis m im HEX16- und ASCII-Ausgabemodus

Kommandos mit drei Argumenten:

n,m,k^F	Fuelle den Speicher von n bis m mit k

Werte fuer die Eingabe:

Eingabewerte werden von links nach rechts berechnet und koennen die Operationen +, -, * und / enthalten. An jedem Element kann sich ein fuehrendes + oder - Vorzeichen befinden. In einem Ausdruck koennen folgende Elemente verwendet werden:

<hex number>	Die letzten vier Zahlen gelten
<hex number>'	Der Zahlen-Offset zum D-Register
'<character>	Der Wert des angegebenen 7-Bit-ASCII-Zeichens
$<symbol> ($ ist ESC)	Der Wert, der fuer das Symbol in der Symboltabelle eingetragen ist, wird verwendet

$	Der Wert ist die Adresse der zuletzt betrachteten Speicherzelle ($ ist $, nicht ESC)
%	Der Wert ist der Inhalt der zuletzt geoeffneten Speicherzelle oder Registers oder der zuletzt durch "=" berechnete Ausdruck

Wenn eine Speicherzelle geoeffnet ist, kann ein Wert eingegeben werden, der den Inhalt der Speicherzelle ersetzt. Folgt dem Wert (falls einer eingegeben wurde) ein CR, schliesst es die Speicherzelle. folgt ein LF, schliesst es die Speicherzelle und oeffnet die folgende. Ein "^" (Pfeil nach oben) schliesst die Speicherzelle und oeffnet die vorhergehende.
Wird kein Wert eingegeben, kann der angezeigte Wert zusaetzlich in einem anderen Ausgabemodus zur Anzeige gebracht werden, indem eines der oben beschriebenen Kommandos ohne Argument eingegeben wird.

RABUG-Register:

Wurden sie durch ein Kommando eroeffnet, kann ein neuer Wert mit abschliessendem CR eingegeben werden, der den alten Inhalt ersetzt. Das Register kann aber auch unveraendert wieder geschlossen werden, indem nur ein CR eingegeben wird.

$MASK	Ist das erste Byte des 4-Byte-Maskregisters
$WORD	Ist das erste Byte des 4-Byte-Wordregisters
$RSWITCH	Ist ein 1-Byte-Register, dessen Wert bestimmt, ob im Schrittbetrieb nach jedem Schritt und beim Erreichen des Breakpoints die Registerinhalte angezeigt werden sollen oder nicht. ($RSWITCH muss 1 oder 0 sein)

Teil 6

D I S K T E S T

Diskettentestprogramm

Inhaltsverzeichnis Seite

1. Einfuehrung

Das Programm DISKTEST (Version 2) dient dazu, Disketten auf ihre Guete zu testen. Sollte die getestete Diskette fehlerhafte Sektoren enthalten, so koennen diese Sektoren im Diskettenbelegungsplan (disk allocation map) als belegt gekennzeichnet werden, und die fehlerhafte Diskette kann wieder ohne Einschraenkungen genutzt werden. ansonsten waere eine Diskette mit fehlerhaften Sektoren unbrauchbar.
Weiterhin koennen fuer spezielle Anwendungen Manipulationen mit dem Diskettenbelegungsplan ausgefuehrt werden.
Das Programm ist menuegesteuert, d.h., alle notwendigen Informationen zur Programmbedienung erscheinen auf dem Bildschirm oder koennen ggf. auf einem Drucker ausgegeben werden.

Der Aufruf erfolgt durch:

```
%DISKTEST [laufwerksnummer]
```

2. Kommandouebersicht

Nach Aufruf von DISKTEST erscheint eine Meldung und das PROMPT-Zeichen "#", das zur Kommandoeingabe auffordert.

Folgende Kommandos sind moeglich:
(xx - Spurnummer, yy - Sektornummer, d - Laufwerksnummer)

T	Diskettentest (mit Unterkommandos TCRC, TERR, ACRC, AERR, Q)
A xxyy	belegt Setzen eines Sektors
F xxyy	frei Setzen eines Sektors
S xxyy	Status eines Sektors (belegt oder frei)
R	Diskettenbelegungsplan in Speicher lesen
W	Diskettenbelegungsplan aus Speicher schreiben
L d	Laufwerkauswahl
DRUCK	Ausgabe der DISKTEST-Beschreibung auf UNIT 3
D	Sprung in den U880-Softwaremonitor
Q	Rueckkehr ins Betriebssystem UDOS

Die Kommandos IT, IA, IF, IS, IR und IW geben Informationen zu den Kommandos T, A, F, S, R bzw. W auf Bildschirm aus.

3. Kommandos

3.1. Kommando "T"

Syntax: T

Das Kommando dient zur Durchfuehrung eines Diskettentestlaufes. anschliessend besteht die Moeglichkeit, fehlerhafte Sektoren oder Sektoren mit CRC-Lesewiederholung im Diskettenbelegungsplan belegt zu setzen, so dass diese Sektoren fuer weitere Systemzugriffe gesperrt sind.

Der Inhalt der zu testenden Diskette wird nicht zerstoert.

Nach Beendigung des Testlaufes erscheint das PROMT-Zeichen ":". Jetzt koennen folgende Unterkommandos eingegeben werden: TCRc, TERR, ACRC, AERR, Q.

Moegliche Fehlerausschriften waherend des Testlaufes:
- "DRIVE d NOT READY"
- "Spur xx, Sektro yy: ERROR zz"

Anmerkung:
Jede UDOS-Diskette enthaelt einen Diskettenbelegungsplan auf Spur 17H ab Sektor 00 (Laenge = 180H Byte). Aus dem Diskettenbelegungsplan kann man ablesen, welche Sektoren der Diskette belegt (entsprechendes Bit = 1) und welche Sektoren frei (entsprechendes Bit = 0) sind.
Nach dem Formatieren einer Diskette sind einige Sektoren bereits belegt (Sektoren fuer das Directory, fuer den Diskettenbelegungsplan und ggf. fuer Systemprogramme).
Sind beim Testen einer neu formatierten Diskette Sektoren fehlerhaft, die bereist belegt sind, so ist diese Diskette nicht mehr verwendbar.
Sind beim Testen einer bereist mit Dateien belegten Diskette Sektoren fehlerhaft, die bereits belegt sind, so ist diese Diskette neu zu formatieren, ein Testlauf durchzufuehren und die fehlerhaften Sektoren "belegt" zu setzen (Unterkommandos ACRC, AERR). anschliessend kann die Diskette wieder mit Dateien bespielt werden.
Diskettenfehler koennen auch laufwerkabhaengig sein (schlecht justiertes oder verschmutztes Laufwerk). Deshalb sollte der Testlauf beim Auftreten von Diskettenfehlern auf einem anderen Laufwerk wiederholt werden, bevor die fehlerhaften Sektoren "belegt" gesetzt werden.

3.1.1. Unterkommando "TCRC"

Ausgabe der "Tabelle der Anzahl der Lesewiederholungen eines Sektors bei CRC-Fehler" auf Konsole

Diese Tabelle wird waehrend des Testlaufes sektorweise ausgebaut.
Jeder Sektor wird beim Auftreten eines CRC-Fehlers intern mehrmals (i.allg. 10mal) gelesen, bevor die Meldung "ERROR C6" erfolgt. Wurde ein Sektor gleich beim erstenmal richtig gelesen, so wird er in der CRC-Tabelle mit "." gekennzeichnet. ansonsten erscheinen in der Tabelle die Anzahl der Leseweiderholungen (hexadezimal 1-E) des entsprechenden Sektors.
Fehlerhafte Sektoren (ERROR C4,C5,C6) werden in dieser Tabelle mit "F" gekennzeichnet.

Die Ausgabe der Tabelle auf Konsole erfolgt spurweise (1 Zeile = 1 Spur). Die Spurnummernangabe erfolgt dabei hexadezimal.
Bei einer einwandfreien Diskette besteht die CRC-Tabelle nur aus ".".

3.1.2. Unterkommando "TERR"

Ausgabe der "Tabelle der Sektor-Lesefehler" auf Konsole

Diese Tabelle wird waehrend des Testlaufes sektorweise aufgebaut. Dabei wird jeder fehlerhafte Sektor (ERROR C4,C5,C6) entsprechend durch "4", "5" bzw. "6" gekennzeichnet.

Die Ausgabe der Tabelle auf Konsole erfolgt spurweise (1 Zeile = 1 Spur). Die Spurnummernangabe erfolgt dabei hexadezimal.

3.1.3. Unterkommando "ACRC"

Es besteht die Moeglichkeit, Sektoren mit CRC-Lesewiederholung (siehe Unterkommando "TCRC") im Diskettenbelegungsplan "belegt" zu setzen.

Sektoren mit CRC-Lesewiederholung wird man i.allg. nur "belegt" setzen, wenn der Wiederholungsfaktor sehr gross ist (z.B. groesser als 5).

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Keine Sektoren mit CRC-Lesewiederholung vorhanden!"
- "Spur xx, Sektor yy, Anzahl der CRC-Lesewdhl.: n"
- "Spur xx, Sektor yy aber bereits belegt"
- "Spur xx, Sektor yy belegt setzen?(Y,N,A,Q):"
  (A --> alle weiteren Sektoren mit CRC-Lesewiederholung werden ohne Abfrage "belegt" gesetzt)
- "Spur xx, Sektor yy belegt gesetzt!"
- "Spur xx, Sektor yy nicht belegt gesetzt!"

3.1.4. Unterkommando "AERR"

Es besteht die Moeglichkeit, fehlerhafte Sektoren im Diskettenbelegungsplan "belegt" zu setzen.

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Keine fehlerhaften Sektoren vorhanden!"
- "Spur xx, Sektor yy fehlerhaft, aber bereits belegt!"
- "Spur xx, Sektor yy belegt setzen?(Y,N,A,Q):"
  (A --> alle weiteren fehlerhaften Sektoren werden ohne Abfrage "belegt" gesetzt)
- "Spur xx, Sektor yy belegt gesetzt!"
- "Spur xx, Sektor yy nicht belegt gesetzt!"

3.2. Kommando "A"

Syntax: A xxyy
(xx-Spurnummer, yy-Sektornummer (hexadezimal))

Das Kommando dient dazu, Sektoren (von denen man z.B. weiss, dass sie fehlerhaft sind) im Diskettenbelegungsplan "belegt" zu setzen, ohne vorher einen Diskettentestlauf durchfuehren zu muessen.

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Spur xx, Sektor yy belegt gesetzt!"
- "Spur xx, Sektor yy bereits belegt!"

Moegliche Ausschriften bei fehlerhafter Parametereingabe:
- "MISSING OR INVALID OPERAND"
  (d.h., keine Hexadezimalzahl eingegeben)
- "BAD DISK ADDRESS"
  (d.h., keine gueltige Diskadresse eingegeben)

3.3. Kommando "F"

Syntax: F xx
(xx-Spurnummer, yy-Sektornummer (hexadezimal))

Das Kommando dient dazu, Sektoren (die z.B. irrtuemlich "belegt" gesetzt wurden) im Diskettenbelegungsplan wieder "frei" zu melden.

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Spur xx, Sektor yy frei gesetzt!"
- "Spur xx, Sektor yy bereits frei!"

Moegliche Ausschriften bei fehlerhafter Parametereingabe:
- "MISSING OR INVALID OPERAND"
  (d.h., keine Hexadezimalzahl eingegeben)
- "BAD DISK ADDRESS"
  (d.h., keine gueltige Diskadresse eingegeben)

3.4. Kommando "S"

Syntax: S xxy
(xx-Spurnummer, yy-Sektornummer (hexadezimal))

Das Kommando dient zum Feststellen des Status eines Sektors im Diskettenbelegungsplan (Sektor belegt bzw. frei).

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Spur xx, Sektor yy frei!"
- "Spur xx, Sektor yy belegt!"

Moegliche Ausschriften bei fehlerhafter Parametereingabe:
- "MISSING OR INVALID OPERAND"
  (d.h., keine Hexadezimalzahl eingegeben)
- "BAD DISK ADDRESS"
  (d.h., keine gueltige Diskadresse eingegeben)

3.5. Kommando "R"

Syntax: R

Das Kommando dient zum Einlesen des Diskettenbelegungsplans in den Speicher ab Adresse 7C00H (180H Byte lang).
Im Monitormode (Kommando "D") kann man sich dann den Diskettenbelegungsplan im Speicher ansehen und ggf. eigene Manipulationen vornehmen. Danach kann der Disketten-belegungsplan mit dem Kommando "W" auf Diskette geschrieben werden.

3.6. Kommando "W"

Syntax: W

Das Kommando dient zum Schreiben des Speicherinhaltes 7C00H-7D7FH auf Diskette als Diskettenbelegungsplan.
Der Speicher 7C00H-7D7FH muss einen gueltigen Disketten-belegungsplan enthalten (z.B. durch vorherigen Aufruf des Kommandos "R")

Moegliche Ausschriften bei ordnungsgemaesser Abarbeitung:
- "Disk-Allocation-Mapp schreiben?(Y):"
- "Disk-Allocation-Mapp geschrieben!"
- "Disk-Allocation-Mapp nicht geschrieben!"

3.7. Kommando "L"

Syntax: L d (d - Laufwerksnummer)

Das Kommando dient zur Aenderung der ausgewaehlten Lauf-werksnummer

3.8. Kommando "DRUCK"

Syntax: DRUCK

Das Kommando dient zum Ausdrucken der DISKTEST-Beschreibung (bestehend aus DISKTEST-Meldung und Inhalt aller I*-Kommandos).
Die DISKTEST-Beschreibung wird dabei auf UNIT 3 ausgegeben. Deshalb ist vor Aufruf von DISKTEST der Druckertreiber auf UNIT 3 zu aktivieren (%ACTIVATE $LP; DEFINE 3 $LP).

3.9. Kommando "D"

Syntax: D

Das Kommando bewirkt einen Sprung in den U880-Softwaremonitor. Dadurch ist es moeglich, die implementierten Monitorkommandos (siehe Band "Einfuehrung in das Geraetesystem P8000" der P8000-Dokumentation) zu nutzen (z.B. Speicher lesen und schreiben).
Der Ruecksprung in das Programm DISKTEST erfolgt mit dem Monitorkommando "Q".

3.10. Kommando "Q"

Syntax: Q

Das Kommando bewirkt das Verlassen von DISKTEST und eine Rueckkehr ins Betriebssystem.
Vorher wird noch ein I-Request an das Dateisystem NDOS realisiert, so dass gewaehrleistet ist, das dann UDOS mit dem ggf. geaenderten Diskettenbelegungsplan weiter arbeitet und nicht mit dem, der vor Aufruf von DISKTEST vorhanden war.

Teil 7

S I

Treiber zur seriellen Datenuebertragung

Inhaltsverzeichnis Seite

1. Einfuehrung

$SI ist ein UDOS-Geraetetreiber zur Eumlation eines Terminals fuer ein externes System. Er ueberwacht die asynchrone Kommunikation zwischen UDOS und dem externen System (z.B. einem "Remote-System" unter WEGA mittels des UDOS-Programms "remote") ueber einen seriellen Kanal des 8-Bit-Teils des P8000.
Die Eingabe vom asynchronen seriellen Kanal ist interruptgesteuert, und das empfangene Zeichen wird in einem 256 Byte langen Puffer abgespeichert.
Die Reuqests zum SI-Treiber sind analog zu den Requests anderer UDOS-Geraetetreiber (siehe z.B. NDOS) anzuwenden. Dazu ist ein Aufruf des UDOS-Operation-Systems (OS) durchzufuehren, wobei das IY-Register auf einen Standard-UDOS-Parameter-Vektor zeigt. Dieser Parameter-Vektor enthaelt folgende Informationen:

Byte	Inhalt
0	logische Geraetenummer (unit
1	Request-Kode
2-3	Datenanfangsadresse
4-5	Datenanzahl (Bytezahl)
6-7	Ruecksprungadresse bei Request-Fertigstellung (wird nur verwendet, wenn Bit 0 des Reuqest-Kodes =1 ist)
8-9	Ruecksprungadresse bei Fehler bzw. 0000H, wenn Ruecksorung bei Fehler mit RET erfolgen soll
10	Fertigstellungskode
11-12	Adresse des zusaetzlichen Parametervektors bzw. 0000H, wenn kein zusaetzlicher Parametervektor angegeben werden soll (fuer ASSIGN und OPEN)

Der zusaetzliche Parametervektor hat folgenden Aufbau:

Byte	Inhalt
0	-
1	Nummer des seriellen Kanals (0-3) (fuer ASSIGN und OPEN mit ASSIGN) (falls zusaetzlicher Parametervektor bzw. Nummer des seriellen Kanals nicht angegeben wurde, wird Kanal 0 verwendet (TTY0))
2	FFH, wenn bei OPEN kein ASSIGN ausgefuehrt werden soll (falls zusaetzlicher Parametervektor nicht angegeben wurde, wird OPEN ohne ASSIGN ausgefuehrt)

Wie bei anderen UDOS-Geraetetreibern kann man mit $SI erst nach dessen Aktivierung arbeiten. Dazu ist das UDOS-Kommando "ACTIVATE $SI" zu benutzen.

2. Request-Kodes

$SI verarbeitet folgende E/A-Requests:

Funktion	Beschreibung
INITIALIZE [00H]	Eroeffnung des Eingabepuffers und Setzen impliziter Parameter
ASSIGN [02H]	Programmierung des SIO und des dazugehoerigen CTC-Kanals zur Baudratengenerierung
OPEN [04H]	Die zurueckgegebenen Daten (falls gefordert, d.h., wenn Datenanzahl > 0 angegeben wurde) stellen den Deskriptor eines ASCII-Geraetes (z.B. einer Datei) mit 100H Byte langen Records und 100H Byte im letzten Record dar. Das zurueckgegebene Erstellungsdatum ist das aktuelle Datum
CLOSE [06H]	Es wird keine Operation ausgefuehrt, aber es ist ein zulaessiger Request.
READ BINARY [0AH]	Empfaengt die angegebene Anzahl von Zeichen aus dem Eingabepuffer. Die Eingabe wird gestoppt, wenn das Zeichen FFH gelesen wird. In diesem Fall wird der "END OF FILE"-Fehlerkode zurueckgegeben. Falls nicht genug Zeichen im Eingabepuffer sind, um den Request zu erfuellen, wartet der Treiber, bis die erforderliche Anzahl von Zeichen empfangen wurde.
WRITE BINARY [0EH]	Sendet die angegebene Anzahl von Zeichen; stoppt aber, wenn das Zeichen FFH auftritt. Das Zeichen FFH wird mit gesendet.
READ STATUS [40H]	Die Bytes des $SI-Status-Feldes werden in dem Speicherbereich beginnend bei der angegebenen Datenanfangsadresse abgespeichert. Die $SI-Statusflags sind in Abschn. 3 definiert.
WRITE STATUS [42H]	Erlaubt das Setzen bestimmter Status-Informationen fuer die Kommunikation. Es ist zu beachten, dass bestimmte Kommunikationsparameter ,(z.B. die Baudrate) waehrend einer Datenuebertragung nicht geaendert werden sollten. Die Statusflags sind in Abschn. 3 definiert.
DEACTIVATE [44H]	Sperrt den Interrupt fuer den Treiber, so dass der Treiber im Speicher geloescht werden kann.
READ ABSOLUTE [46H]	Analog zu READ BINARY, ausser, dass das Zeichen FFH keine spezielle Bedeutung hat.

WRITE ABSOLUTE [48H] Analog zu WRITE BINARY, ausser dass das Zeichen FFH keine spezielle Bedeutung hat.

3. Datenformat fuer READ/WRITE-Status-Request-Kodes

Byte	Bit	Bedeutung
0	0	Die Empfaenger-Interrupt-Routinen testen die empfangenen Zeichen auf Spezialzeicehn, die die Zeichenausgabe steuern. Dabei stoppt das "XOFF"-Zeichen die Ausgabe bis das "XON"-Zeichen empfangen wird. Falls dieses Bit=0 ist, wird dieser Test nicht durchgefuehrt. (initialisiert auf 1)
	1	Beim Fuellen und Leeren des Eingabepuffers senden die Empfaenger-Interrupt-Routinen automatisch die Spezialzeichen "XON" und "XOFF", um den Sender am anderen Leitungsende zu steuern. Das "XOFF"-Zeichen wird gesendet, wenn der Eingabepuffer 3/4 voll ist und das "XON"-Zeichen wird gesendet, wenn der Eingabepuffer wieder 1/4 voll ist. (initialisiert auf 1)
	2	(Lesebit) Die Zeichenausgabe ist durch den Empfang eines "XOFF"-Zeichens unterbrochen.
	3	(Lesebit) Die Eingabe-Interrupt-Routine hat ein "XOFF"-Zeichen gesendet, um die Zeicheneingabe zu unterbrechen.
	6	Der Eingabepuffer ist ueberfuellt. ein Zeichen wurde empfangen, waehrend der Eingabepuffer voll war. Dieses Bit bleibt gesetzt, bis ein "WRITE STATUS"-Kommando es loescht.
	7	(Lesebit) Dieses Bit ist 0, wenn der Eingabepuffer leer ist und 1, wenn im Eingabepuffer Zeichen sind.

---

1 Ein Lesen dieses Bytes zeigt die aktuellen Uebertragungsparameter an.
Ein Schreiben in Verbindung mit Byte 2 kann diese Parameter setzen (SIO-Schreibregister 4,5).

1,0 Paritaet
00 - keine Paritaet (initialisierter Wert)
01 - ungerade Paritaet
10 - keine Paritaet
11 - gerade Paritaet

3,2 Anzahl der Stop-Bits
00 - undefiniert
01 - 1 Stop-Bit
10 - 1 1/2 Stop-Bits
11 - 2 Stop-Bits (initialisierter Wert)

4,5 Anzahl der Bits je Zeichen
00 - 5 Bits
01 - 6 Bits
10 - 7 Bits
11 - 8 Bits (initialisierter Wert)

7,6 SIO-Clock-Rate
00 - x 1
01 - x 16 (initialisierter Wert)
10 - x 32
11 - x 64

---

2 Maske zum Setzen der Uebertragungsparameter in Byte 1.
Nur die Bits in Byte 1 werden bei einem "WRITE STATUS"-Request geaendert, dessen entsprechende Bits in Byte 2 den Wert 1 haben (z.B. Byte 2 = 03H, d.h., nur die Paritaet wird geaendert).
Ein "READ STATUS" Request liefert bei diesem Byte immer den Wert 0

---

3 weitere Uebertragungsparameter
(SIO Schreibregister 5)

1 REQUEST TO SEND (RTS) (initialisiert auf 1)

2 Ruecksetzen des Fehlerflags
Dieses Flag setzt den Paritaets- und Ueberlauffehler in Byte 6 zurueck.

4 SEND BREAK (initialisiert auf 0)

7 DATA TERMINAL READY (DTR) (initialisiert auf 1)

---

4 Maske zum Setzen der Uebertragungsparameter in Byte 3.
Nur die Bits in Byte 3 werden bei einem "WRITE STATUS"-Request geaendert, deren entsprechende Bits in Byte 4 den Wert 1 haben.
Zusaetzlich wird, wenn Bit 0 = 1 ist, die Uebertragunsgrate entsprechend Byte 5 gesetzt.

---

5 Zeitkonstante fuer die Baudratengenerierung durch einen CTC im Zaehlermode
(initialisiert auf 4 - entspricht 9600 Baud)

---

6 Die Werte der aktuellen SIO-Status-Bits werden hier zurueckgegeben (aus SIO-Leseregister 0, 1).

0 Paritaetsfehler

1 Ueberlauffehler

2 FRAME ERROR

3 DATA CARRIER DETECT (DCD)

4 SYNC

5 CLEAR TO SEND (CTS)

7 BREAK wurde erkannt

---

7 "XOFF"-Zeichen, das empfangen wird, um die Zeichenausgabe zu unterbrechen
(initialisiert auf 13H)

---

8 "XON"-Zeichen, das empfangen wird, um die unterbrochene Zeichenausgabe wieder fortzusetzen
(initialisiert auf 11H)

---

9 "XOFF"-Zeichen, das zu senden ist, wenn der Eingabepuffer 3/4 voll ist
(initialisiert auf 13H)

---

10 "XON"-Zeichen, das zu senden ist, wenn der Eingabepuffer nur noch 1/4 voll ist (initialisiert auf 11H)

---

11 Maske, die auf jedes empfangene Zeichen angewendet wird, bevor es auf das "XON"- bzw. "XOFF"-Zeichen getestet wird (initialisiert auf 7FH)

---

4. SI-Konfiguration

Der $SI-Treiber benutzt die folgende Hardwarekonfiguration des 8-Bit-Mikrorechnerteils des P8000:

E/A-Adressen:

Nummer des seriellen Kanals	Geraet	Port-Adr.
0 (TTY0/XB6)	SIO 0 - PORT A	24H, 25H (impliziter Kanal)
1 (TTY1/XB7)	SIO 0 - PORT B	26H, 27H (Systemterminal!)
2 (TTY2/XB7)	SIO 1 - PORT a	28H, 29H
3 (TTY3/XB7)	SIO 1 - PORT B	2AH, 2BH
	CTC 1 - Kanal 0	2CH (Baudrate fuer SIO0_A)
	CTC 1 - Kanal 1	2DH (Baudrate fuer SIO0_B)
	CTC 1 - Kanal 2	2EH (Baudrate fuer SIO1_A)
	CTC 0 - Kanal 0	08H (Baudrate fuer SIO1_B)

Baudratentabelle:

Baudrate	Zeitkonstante fuer CTC
38400	1
19200	2
9600	4 (impliziter Wert)
4800	8
2400	16
1200	32
600	64
300	128

5. Beispiel

Beispiel einer Datenuebertragung zwischen zwei P8000 unter Betriebssystem UDOS:

Die Datei "datei1" des Systems A ist zur Datei "datei2" des Systems B zu uebertragen.

Dazu ist zuerst der Treiber $SI auf beiden Systemen zu aktivieren:

```
System A                    System B
%ACTIVATE $SI               %ACTIVATE $SI
```

Danach ist der Treiber $SI auf dem System B zu initialisieren um zu gewaehrleisten, dass der Eingabepuffer neu initialisiert ist, d.h., dass sich keine ungueltigen Zeichen (z.B. durch die Aktivierung) in ihm befinden.
Dazu ist folgendes einzugeben:

```
                            System B
                            %I $SI
```

Zur eigentlichen Datenuebertragung ist dann einzugeben:

```
System A                    System B
%COPY datei1 $SI            %COPY $SI datei2
```

Wenn auf dem System B das PROMPT-Zeichen "%" erscheint, ist die Uebertragung abgeschlossen.

Hinweis:
Soll zur Datenuebertragung ein anderer als der serielle Kanal 0 (TTY0/Buchse XB6) benutzt werden, so ist beim Kopieren anstelle von $SI zu schreiben $SI:x (x=0,1,2,3), wobei x die Nummer des seriellen Kanals angibt.

Hinweise des Lesers zu diesem Dokumentationsband

Wir sind staendig bemueht, unsere Unterlagen auf einem qualitativ hochwertigen Stand zu halten. Sollten Sie deshalb Hinweise zur Verbesserung dieses Dokumentationsbandes bzw. zur Beseitigung von Fehlern haben, so bitten wir Sie, diesen Fragebogen auszufuellen und an uns zurueckzusenden.

Titel des Dokumentationsbandes:

Ihr Name / Tel.-Nr.:

Name und Anschrift des Betriebes:

Genuegt diese Dokumentation Ihren Anspruechen? ja / nein
Falls nein, warum nicht?

Was wuerde diese Dokumentation verbessern?

Sonstige Hinweise:

Fehler innerhalb dieser Dokumentation:

Unsere Anschrift: Kombinat VEB ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW "FRIEDRICH EBERT"
Abteilung Basissoftware
Hoffmannstrasse 15-26
BERLIN
1193

POWERS OF 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$

POWERS OF 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

Notizen des Anwenders

Kombinat VEB
ELEKTRO-APPARATE-WERKE
BERLIN-TREPTOW >FRIEDRICH EBERT<
Hoffmannstraße 15-26, Berlin, DDR-1193
011 2263 eaw 011 2264 eaw

Die Angaben über technische Daten entsprechen dem bei Redaktionsschluß vorliegenden Stand. Änderungen im Sinne der technischen Weiterentwicklung behalten wir uns vor.