A MITEL
PRODUCT
GUIDE

Unify OpenScape 4000/HiPath 4000

Debug

Debug

Servicedokumentation
06/2018

A31003-H3100-S106-6-20

Mitel

Notices

The information contained in this document is believed to be accurate in all respects but is not warranted by Mitel Europe Limited. The information is subject to change without notice and should not be construed in any way as a commitment by Mitel or any of its affiliates or subsidiaries. Mitel and its affiliates and subsidiaries assume no responsibility for any errors or omissions in this document. Revisions of this document or new editions of it may be issued to incorporate such changes. No part of this document can be reproduced or transmitted in any form or by any means - electronic or mechanical - for any purpose without written permission from Mitel Networks Corporation.

Trademarks

The trademarks, service marks, logos, and graphics (collectively “Trademarks”) appearing on Mitel’s Internet sites or in its publications are registered and unregistered trademarks of Mitel Networks Corporation (MNC) or its subsidiaries (collectively “Mitel), Unify Software and Solutions GmbH & Co. KG or its affiliates (collectively “Unify”) or others. Use of the Trademarks is prohibited without the express consent from Mitel and/or Unify. Please contact our legal department at iplegal@mitel.com for additional information. For a list of the worldwide Mitel and Unify registered trademarks, please refer to the website: http://www.mitel.com/trademarks.

Contents

1 Allgemeines über DEBUG

Der Debugger ist ein Software-Werkzeug welches mit AMO DEBUG vom Betriebssystem zur Verfügung gestellt wird. Mit dem Debugger können Anweisungen und Kommandos eingegeben werden, die dem Systemspezialisten ermöglichen gezielte Diagnose-Aktivitäten durchzuführen.

Der Debugger ermöglicht sowohl das Lesen und Schreiben als auch das Setzen von TRACE-Marken im Speicher (Datenbasis).
Mit Hilfe von DEBUG-Runfiles ist es möglich, bestimmte DEBUG-Kommandos an definierten Punkten automatisch zur Ausführung zu bringen.

Eine Unterfunktion des Debuggers sind die Trace AMOs (AMO TRACA und TRACS). Es können Meldungen zwischen SW-Komplexen des Systems nach bitgenauen Filtereigenschaften in den Tracespeicher eingelesen werden. Fehler können später anhand des dokumentierten Meldungsaustausches eingegrenzt werden.

Weitere Trace-Möglichkeiten bieten die AMOs

- PETRA (peripherer Tracer)
- DISPA und DISPS (Speicherbereiche auslesen)

IMPORTANT: Die Anwendung von DEBUG, TRACA, TRACS, DISPA, DISPS und PETRA setzt Kenntnisse der Systemsoftware und des CHILL-Sourcecodes voraus.
Fehlbedienungen können zum Ausfall des Systems und zur Zerstörung des Systemprogramms in der Datenbasis, evtl. auch auf der Hard Disk führen

---

2 Benutzungsoberfläche

2.1 Programmaufruf

Bevor an den DEBUG Eingaben gemacht werden können, muss erst ein 'Vermittler' aktiviert werden, der den Transfer der Ein- und Ausgabedaten von den Eingabegeräten an den DEBUG (und umgekehrt) übernimmt.
Es kann zwischen zwei 'Vermittlern' gewählt werden:

- dem AMO-DEBUG (AMO) (siehe Section 2.7, "Eingabe über AMO DEBUG")

und

- dem Miniterminal Handler (MTH) (siehe Section 2.8, "Eingabe über Miniterminal Handler (MTH)").

Der MTH bedient die an den DP angeschlossenen Geräte, während der AMO DEBUG die an der LCU installierten Geräte anspricht. Es wird in diesem Kapitel nicht näher darauf eingegangen, welche Endgeräte angeschlossen werden können. Die hier beschriebenen Eingaben beziehen sich auf primitive Endgeräte, die die Ein und Ausgaben transparent weiterreichen.
Die Möglichkeiten, die sich mit intelligenten Endgeräten, (PC, APOLLO) bieten, können hier nur angedeutet werden (siehe z.B. Produkt DASIST auf APOLLO).

Z.B. sind diese:

- menügesteuerte Eingabe,
- Stapelverarbeitung,
- CHILL Symbolik,
- Disassembler,
- Rückübersetzungen, usw.

Nach der Aktivierung des 'Vermittlers' kann der Dialog mit dem DEBUG geführt werden. Die Dialogschnittstelle ist so realisiert, dass der DEBUG gleichzeitig über MTH und AMO bedient werden kann. Da DEBUG und MTH in jedem Prozessor vorhanden sind, kann an die MTH Schnittstelle in jedem Prozessor ein eigenes Endgerät angeschlossen werden. Damit kann in mehreren Prozessoren gleichzeitig getestet werden. Über die AMO-Schnittstelle (LCU) kann jeweils nur ein Prozessor angesprochen werden. Es kann aber jederzeit auf einen anderen Prozessor umgeschaltet werden. Soll der Dialog mit dem DEBUG beendet werden, so kann der 'Vermittler' mit Kommando 'DEND' wieder deaktiviert werden. Sind mehrere LCU's vorhanden, so kann über die verschiedenen MMI-Sessions 'gleichzeitig' in mehreren Prozessoren der Dialog mit DEBUG geführt werden.
Das Beenden des Vermittlers hat keinen Einfluss auf den Zustand im DEBUG.

2.1.1 Der AMO DEBUG (AMO)

Die AMO Schnittstelle wird aktiviert mit:

```
EXEC-DEBUG;
```

2.1.2 Der Miniterminal Handler (MTH)

Der MTH wird aktiviert durch die Eingabe von ’CTRL D’.
Es ist darauf zu achten, dass nur Großbuchstaben eingegeben werden.

2.2 Überblick über DEBUG Kommandos

Kommando	Bedeutung	Typ
ACT	Aktivieren Testauftrag	in / op
AT	Definieren Testauftrag	in
BREAK	Prozessor anhalten	in / op
CONT	Task fortsetzen	in
D	Display (identisch mit TRACE)	in / op
DCL	Definieren DEBUG Variable	in
DEACT	Deaktivieren Testauftrag	in / op
DEF	Definieren Symbole	in
DEND	Beenden Dialog	spec
DOWHILE	Schleifenbedingung formulieren	in / op
END	Beenden Eingabe der Anweisungen	in
EXEC	Anschlussroutinen aufrufen	in / op
FILE	Zuweisen Protokoll-File	spec
GO	Fortsetzen Prozessor	in
GOFOR	Beenden Breakmodus	in
GOTIL	Setzen von Breakpunkten	in
IF	Bedingungen formulieren	in / op
IN	Anweisungen im Dialog eingeben	spec
INSP	Ausgeben von Systemobjektdaten	in / op
LIST	Ausgeben Benutzerinformation	spec
LOGOFF	Triggereintrag ausschalten	in / op
LOGON	Triggereintrag einschalten	in / op
MCALL	Macro aufrufen	in / op

Table 1 DEBUG Anweisungen (alphabetisch)

Kommando	Bedeutung	Typ
MDEF	Definieren Macro	in
PRINT	Ausdrucken der Indizien	spec
REDCL	Umdefinieren DEBUG Variable	in
REDEF	Umdefinieren DEF-Symbol	in
REMAT	Löschen eines Testpunktes	in
REMMAC	Löschen eines Macros	in
REMOVE	Löschen eines Breakpunktes	in
RESET	Rücksetzen des Tracebuffers	spec
RUN	Anweisungen von Datei einlesen	spec
RUNOFF	Eingaben nicht mehr protokollieren	in
RUNON	Eingaben in Runfile protokollieren	in
SAVEOFF	Ausschalten des Patch	spec
SAVEON	Einschalten der Patch	spec
SEL	Setzen Selektion für Triggermodus	spec
SET	Modifizieren Speicherinhalt	in / op
START	Start einer DEBUG Session	spec
STOP	Task anhalten	op
TERM	Beenden einer DEBUG Session	spec
TRACE	Tracen Speicherinhalt (Display)	in / op
TRIGGOFF	Ausschalten des Triggermodus	spec
TRIGGON	Einschalten des Triggermodus	spec
VIEW	Auflisten von Systemobjekten	in / op
WRITE	Kommentare schreiben	in / op
!	Ausführen Instruktionen	in
;	Leer-Anweisung	in / op

Table 1 DEBUG Anweisungen (alphabetisch)

Art: in Instruktion (instruction)
op Operation (operation)
spec Spezielles Kommando (special command)

Im DEBUG werden die durchzuführenden Kommandos vom Anwender eingegeben. Dabei gibt es verschiedene Kommandoebenen,

- eine Steuerebene,
- eine interaktive Ebene und
- eine Testpunktebene.

Entsprechend diesen Ebenen werden auch die Kommandos eingeteilt.

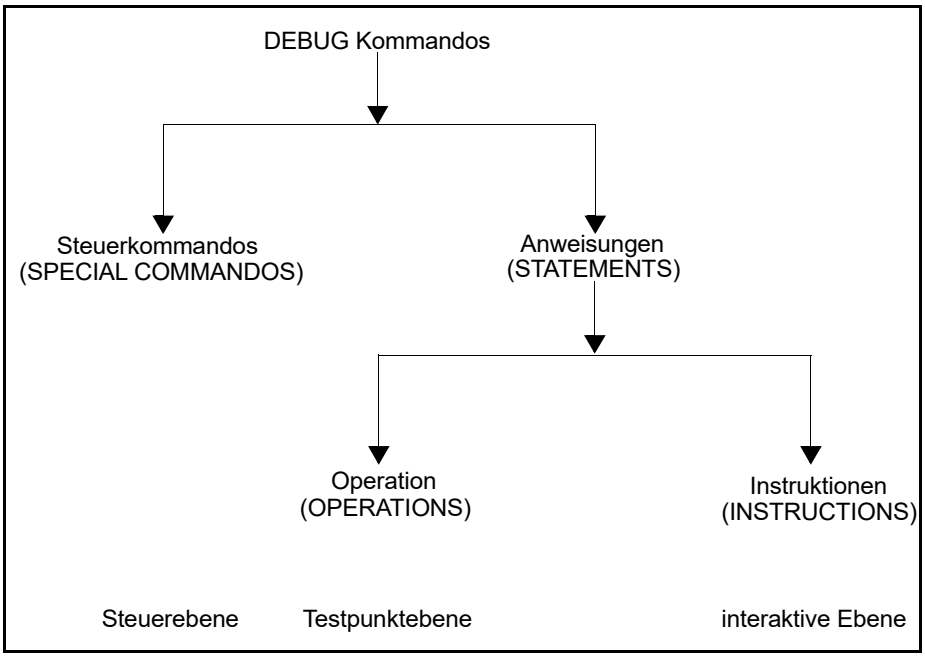

Figure 1 Überblick DEBUG-Kommandos

DEBUG Steuerkommandos (special commands)

Mit diesen Kommandos werden Funktionen durchgeführt, die nicht unmittelbar der Indiziengewinnung dienen. Insbesondere sind dies Kommandos zum Zuweisen von Protokoll-Dateien, zum Ausdrucken von Indizien und zum Starten und Beenden einer DEBUG-Session.

DEBUG Anweisungen (statements)

DEBUG Anweisungen sind Kommandos, die der Indiziengewinnung dienen. In einer CHILL-ähnlichen Sprache geschrieben, sind sie das Instrument, um dem DEBUG mitzuteilen, welche Diagnose-Aktivitäten durchzuführen sind. DEBUG Anweisungen können in Anweisungsdateien (RUN-files) enthalten sein. In Anweisungsdateien sind Steuerkommandos nicht zugelassen.
DEBUG Anweisungen werden unterschieden in Instruktionen und Operationen, je nach dem, ob die Anweisung interaktiv oder am Testpunkt ausgeführt wird. Neben den Kommandos, die nur am Testpunkt oder nur interactiv möglich sind, gibt es auch Kommandos, die in beiden Zuständen ausgeführt werden. Es gibt z.B. eine SET-Operation und eine SET-Instruktion. Syntaktisch unterscheiden sich Operationen und Instruktionen nicht.

Operationen (operations)

Operationen sind DEBUG Anweisungen, die am Testpunkt (Trigger) ausgeführt werden. Die Operationen beziehen sich auf Speicherzustände zum Zeitpunkt des Erreichens des Triggers.

Instruktionen (instructions)

Instruktionen sind DEBUG Anweisungen, die interaktiv ausgeführt werden. Sie beziehen sich auf Speicherinhalte zum Zeitpunkt der Eingabe (genauer zum Zeitpunkt der anschliessenden Ausführung)

Kommando	Langname
DO ... OD	Klammerung Operationen bei AT
FI	Beenden IF Konstruktion
MEND	Beenden Macrooperation bei MDEF
OD	Beenden Operationen in DOWHILE
THEN,ELSE	Zweige der IF Konstruktion

Table 2 Hilfsschlüsselwörter (alphabetisch)

2.3 Eingabe von Kommandos an den DEBUG

Die Eingaben werden vom DEBUG geprüft (siehe Behandlung der Syntaxfehler). Steuerkommandos können nicht über mehrere Zeilen fortgesetzt werden. Sie sind auf 160 Zeichen begrenzt. Werden die Steuerkommandos in der interaktiven Ebene eingegeben, so sind sie auf 80 Zeichen beschränkt.
Dagegen kann eine DEBUG Anweisung auf mehrere Eingaben aufgeteilt werden. Andererseits können mit einer einzigen Eingabe mehrere DEBUG Anweisungen übertragen werden. Die Länge einer Eingabe ist bei Anweisungen auf 80 Zeichen begrenzt. Kommentare sind in Steuerkommandos und Anweisungen überall dort zugelassen, wo ein ’Blank’ sein darf. Sie werden durch /*... . lang sein (auch über mehrere Eingaben hinweg).
Die Syntax der einzelnen Kommandos ist in den Abschnitten ’Beschreibung’ und ’Syntax in BNF’ erläutert.

2.4 Sonderzeichen, die in den Kommandos benutzt werden

%	Kennzeichen für DEBUG Variable und DEBUG-Mode
#	bei D R oder Kennzeichnung eines Pointerwertes #Selektor:Offset oder Kennzeichnung des Auftragskennzeichens beim D Kommando
&	Kennzeichen für DEF-Symbole
;	Ende einer DEBUG Anweisung (wird nicht sofort ausgeführt), oder Leer-Anweisung
,	zur Trennung von Operanden in DEBUG Anweisungen

:	Trennung von Labelname und AT-Schlüsselwort, oder Trennung der Bestandteile einer Adressangabe, oder Trennung der Bestandteile einer Zeitangabe
(	für Längenangabe und zur Klammerung in DEBUG Anweisungen
)	für Längenangabe und zur Klammerung in DEBUG Anweisungen
=	Zuweisung in SET, DEF und REDEF Anweisungen, oder Vergleichsoperator bei IF und DOWHILE
>	Vergleichsoperator bei IF und DOWHILE
<	Vergleichsoperator bei IF und DOWHILE
->	Zeichen für Dereferenzierung in Ausdrücken (expressions)
+	in Ausdrücken (expressions)
*	in Ausdrücken (expressions), oder in prozedurlokalen Adressen (aktuelle Prozedur), oder bei Teilqualifizierung von Testpunktnamen
-	in Ausdrücken (expressions)
/	in Ausdrücken (expressions), oder Trennung der Bestandteile der Datumsangabe
/*	Zeichen für Kommentaranfang
*/	Zeichen für Kommentarende
’	bei Wahl der numerischen Darstellung (T’,B’,H’), oder Begrenzer für HEX-Strings (X’), oder Begrenzer für CHAR-Strings (’.....’)
.	in Namen und Zahlen zur Gliederung von Zeichenfolgen
_	in Namen und Zahlen zur Gliederung von Zeichenfolgen
!	Ende einer Anweisung (sie wird sofort ausgeführt), oder Anstoß der Ausführung bisher nicht ausgeführter Anweisungen

2.5 Dialogzustände des DEBUG

Nach einem Hochlauf (Restart) außerhalb der DEBUG Session befindet sich DEBUG in der Steuerebene. In der Steuerebene ist das zweite Promptzeichen ein BLANK.

Nach einem Hochlauf (Restart) innerhalb der DEBUG Session befindet sich DEBUG in der interaktiven Ebene und fordert die Eingabe einer Instruktion an (unabhängig vom Zustand vor dem Restart). In diesem Zustand ist das zweite Promptzeichen ein ’*’.

Durch das START Kommando wird eine DEBUG Session eröffnet. Das Arbeiten mit DEBUG setzt das Eröffnen einer Session voraus. Dabei werden sämtliche Definitionen, die in der vorherigen Session gemacht wurden, gelöscht. Durch das START Kommando gelangt man sofort in die interaktive Ebene. In dieser Ebene ist das zweite Promptzeichen ungleich BLANK.

Nur mit TERM Kommando wird die DEBUG Session wieder verlassen. Beim Verlassen der DEBUG Session wird jede DEBUG Aktivität beendet. Nach dem TERM Kommando befindet man sich in der Steuerebene.

Durch das Kommando ’IN’ gelangt man von der Steuerebene in die interactive Ebene. Wird das Kommando in der interaktiven Ebene gegeben bleibt man in dieser Ebene.

Von der interactiven Ebene gelangt man mit der END-Anweisung in die Steuerebene.
In der interaktiven Ebene können alle Kommandos gegeben werden, in der Steuerebene nur die Steuerkommandos.

2.6 Prompting

Neben der Einteilung in Steuerebene und interactive Ebene haben weitere Systemzustände auf den Gültigkeitsbereich der Kommandos einen Einfluss. Sie werden deshalb dem Anwender in der Eingabeanforderung mit Hilfe der Promptzeichen angezeigt.

Das Prompting besteht immer aus drei Zeichen:

- das erste Zeichen gibt den Systemzustand an,
- das zweite Zeichen gibt den Dialogzustand im DEBUG an,
- das dritte Zeichen gibt den Dialogzustand mit dem ’Vermittler’ an.

Erstes Promptzeichen (Systemzustand)

Bedeutung		Auswirkung auf die Kommandos (- : nicht zugelassen) (+ : zusätzlich möglich)	
+	Außerhalb DEBUG Session	- - -	TERM, FILE, IN, RUN, RESET, SAVEON, SAVEOFF, TRIGGON, SEL TRIGGOFF,
*	Innerhalb DEBUG Session	-	START
#	SAVEON ist gesetzt	-	GO, GOTIL, BREAK, REMOVE, GOFOR
?	Break modus eingeschaltet	- - -	RUN, SAVEON, SAVEOFF, FILE IN: Parameter RUNFILE PRINT: Parameter INFILE
$	Prozessor am Trigger angehalten	+ + + -	Verwendung von Registernamen bei Eingabe numerischer Werte prozedurlokale Adressangaben in Instruktionen (P*, PB,..) D #R als Instruktion wie bei '?'
%	Prozessor interaktiv angehalten	-	wie bei '?'
/	Kommentarmodus	-	alles wird ignoriert außer */

Table 3 Erstes Promptzeichen

IMPORTANT: Der Kommentarmodus der Steuerebene wird nur in der Steuerebene selbst angezeigt, in der interactiven Ebene wird der Systemzustand eingeblendet.

Zweites Promptzeichen (Dialogzustand der interaktiven Ebene)

	Bedeutung
*	Anforderung einer Instruktion, alle vorherigen sind ausgeführt
	Debug in Steuerebene (siehe Hinweis)
+	Restart (siehe Hinweis)
!	Anforderung einer Instruktion, vorherige noch nicht ausgeführt
:	Anforderung innerhalb einer Testpunktdefinition
-	Anforderung innerhalb IF oder DOWHILE
/	Anforderung innerhalb eines Kommentares
.	Anforderung innerhalb einer Macrodefinition
;	Anweisung ist noch nicht vollständig eingegeben.

Table 4 Zweites Promptzeichen

IMPORTANT: Ein Leerzeichen (BLANK) signalisiert, das man sich nicht in der interaktiven Ebene sondern in der Steuerebene befindet
Das Zeichen '+' an dieser Position signalisiert, dass es sich um das erste Prompt nach einem Restart handelt. In der Regel ist dieser Prompt nur kurzeitig am Bildschirm zu sehen, da es von der MTH Meldung V120 überschrieben wird.

Drittes Promptzeichen (Dialogzustand mit dem Vermittler)

	Bedeutung
>	Innerhalb Dialog mit dem Vermittler
<	AußerhalbDialog mit dem Vermittler (nur bei MTH)

Table 5 Drittes Promptzeichen

Beispiele für Promptzeichen

+ > Debug ist außerhalb der Session (in der Steuerebene)

* > Debug ist innerhalb der Session in der Steuerebene

**> Debug ist innerhalb der Session in der interaktiven Ebene

$.> Während der Prozessor an einem Trigger angehalten wurde wird in der interaktiven Ebene ein Makro definiert

2.7 Eingabe über AMO DEBUG

Die AMO Schnittstelle wird aktiviert mit:

```
EXEC-DEBUG;
```

Darauf wird abgefragt für welchen Prozessor der Dialog eröffnet werden soll. Nach Eingabe der Prozessor Identifikation (sie kann auch beim Aufruf ’EXEC-DEBUG’ gleich mitangegeben werden) erscheint das DEBUG Prompting und anschliessend die Eingabeanforderung des AMO. Es kann nun eine Eingabe gemacht werden. Die Eingaben müssen in die MMI Syntax der AMOS eingebettet werden. Dies bedeutet, dass am Anfang und am Ende der Eingabe das Zeichen ’ " ’ stehen muss. Ist dies nicht der Fall, werden die Sonderzeichen vom MMI interpretiert und nicht an den DEBUG weitergeleitet. Insbesondere ist dies auch bei der Eingabe von ’;’ und ’!’ zu beachten. Soll ein ’;’ an den DEBUG weitergeleitet werden, so ist einzugeben: ’";"’ Der Dialog wird mit dem beim Aktivieren angegebenen Prozessor geführt. Soll der Dialog mit einem anderen Prozessor geführt werden, so ist der AMO zu deaktivieren und erneut zu aktivieren. Deaktiviert wird der AMO mit dem Kommando DEND. Erfolgt länger als 15 Minuten keine Eingabe so deaktiviert sich der AMO selbst.

IMPORTANT: Die Anweisungen BREAK, GO, GOFOR, REMOVE, GOTIL dürfen nicht über diese AMO Schnittstelle eingegeben werden. Während des Breakmodus (erkennbar an den Promptzeichen ’?’, ’%’, und ’$’) dürfen keine Eingaben erfolgen. In allen Fällen tritt Deadlock auf, wenn der Prozessor eingefroren wird, da beim Einfrieren alle Tasks (mit Ausnahme von MTH und DEBUG) suspendiert werden.
Wird das Mitprotokollieren eingeschaltet (TRIGGON), so wird nur über die MTH-Schnittstelle protokolliert (MTH braucht dazu nicht aktiviert zu werden).

Beispiel: (E): diese Zeile wurde eingegeben
(A): diese Zeile wurde ausgegeben
(AE): Zeile enthält Ein- und Ausgaben

```
(A )  EXEC-DEBUG:A1;
(A )  H500:          AMO DEBUG GESTARTET
(A )  + >
(A )  BITTE DEBUG-EINGABE FUER A1
(AE)  *START

(A )  0304I START  :DEBUG V4.0 KV01: SESSION STARTED AT:
(A )  11:08:06     21/06/      PROC-ID = 019
                   1987

(A )  **>
(A )  BITTE DEBUG-EINGABE FUER A1
(AE)  *"d 2BF8:12(6);"
```

```
(A )   *!>
(A )   BITTE DEBUG-EINGABE FUER A1
(AE)   *"dend"
(A )   AMO-DEBUG-219          VERFUEGBARMACHEN HICOM
                              DEBUG
(A )   EXEC DURCHGEFUEHRT;

(A )   EXEC-DEBUG:A1;
(A )   H500:          AMO DEBUG GESTARTET
(A )   *!>
(A )   BITTE DEBUG-EINGABE FUER A1
(AE)   *"!"

(A )   **>
(A )   BITTE DEBUG-EINGABE FUER A1
(A )   *
```

2.8 Eingabe über Miniterminal Handler (MTH)

Der MTH wird aktiviert durch die Eingabe von ’CTRL D’. Ist nicht bekannt, ob er aktiviert werden muss, bzw in welchem Zustand sich der Dialog befindet, so wird der Zustand durch Drükken der Returntaste (Endegerätabhängig!!) ausgegeben. Entweder erscheint das DEBUG Prompting oder die Meldung:

```
MTH V120:  FOLGENDE ANWENDUNGEN SIND ANGEMELDET WORDEN:
01 DEBUG-DIALOG(13)  KANN MIT ’CTRL D’ GESTARTET WERDEN!
```

Nur im zweiten Fall muss der MTH aktiviert werden. (Aus der MTH-Meldung ist ersichtlich welcher Prozessor angesprochen wird). Nach der Aktivierung des MTH erscheint das DEBUG Prompting. Es kann nun in dem Prozessor, an dem das Terminal angeschlossen ist, der Dialog mit DEBUG geführt werden. Soll der Dialog beendet werden, so ist das Kommando ’DEND’ einzugeben.

2.9 Sporadische Meldungen

Sporadische Meldungen sind solche Meldungen, die nicht unmittelbar als Antwort auf eine DEBUG Anweisung ausgegeben werden. Beispiele sporadischer Meldungen sind die Meldungen über den Wechsel der Protokolldateien und die Prozessor-Stop Meldung. Die sporadischen Meldungen werden sowohl über die AMO-Schnittstelle (sofern der Dialog aufbebaut ist) als auch über die MTH-Schnittstelle ausgegeben. Die Ausgabe der sporadischen Meldung wird zurückgehalten, wenn gerade die Bearbeitung eines Kommandos läuft.

Besonderheiten an der MTH-Schnittstelle:

Die Ausgabe der sporadischen Meldungen ist unabhängig davon ob der Dialog mit dem MTH eröffnet wurde oder nicht. Sie werden mit ETX(=H’03) abgeschlossen (für Anwendung über DASIST).

Wenn die ESC-Taste gedrückt wurde, wird die Ausgabe der sporadischen Meldungen solange zurückgehalten, bis wieder die Returntaste gedrückt wurde.

2.10 Besonderheiten bei Monoprozessor-Anlagen (Hicom 3x3)

Bei Monoprozessor-Anlagen ist kein MTH vorhanden. Die MTH-Schnittstelle wird stattdessen durch das CMS nachgebildet. Das Endgerät kann nur an der zweiten V.24 Schnittstelle der DM80 betrieben werden (siehe Kapitel Baugruppen, DM80).
Die V.24-2 Schnittstelle muss als "asynchrones Terminal" konfiguriert sein, damit man über diese Schnittstelle mit dem Debugger arbeiten kann.

Durch Eingabe von "CTRL D" kann dann eine Dialogverbindung mit dem Debugger hergestellt werden. Die Dialogverbindung wird wieder abgebaut durch Eingabe von "DEND".

Sporadische Debugger-Meldungen können nur dann ausgegeben werden, wenn eine Dialogverbindung besteht.

IMPORTANT: An der V.24-2 Schnitstelle ist es auch möglich eine "Terminal Task Session" zu eröffnen, indem man "CTRL T" eingibt.
Diese Session muss man vor Eingabe von "CTRL D" beendet haben.

---

3 Beschreibung der DEBUG-Kommandos

Die vollständige Beschreibung aller Eingabevarianten für ein Kommando ist aus den Syntax-Diagrammen ersichtlich.

Eingaben von Zahlen in DEBUG-Anweisungen sind

- hexadezimal (H'....)
- binär (B'....)

oder

- dezimal (T'....)

möglich.

Ohne Kennzeichnung werden Zahlen hexadezimal interpretiert.

3.1 START-Kommando (Start einer DEBUG Session)

Es wird die DEBUG Session eröffnet. Dies ist die Voraussetzung um mit DEBUG zu testen. Bei diesem Kommando werden die DEBUG internen Tabellen und Bereiche initialisiert. Nach der Ausführung dieses Kommandos sind keine Testpunkte, Macros oder DEBUG Variable definiert. Von den DEF-Symbolen sind nur die Standard DEF-Symbole definiert und neu initialisiert. Der Tracebuffer ist leer. In der Startquittung werden die DEBUG Version und das Datum mit Uhrzeit sowie die Prozessornummer (dezimal und hex) ausgegeben. Das START Kommando wechselt in die interaktive Ebene.

Beispiel:

```
START;
0304I START  :DEBUG V300    :STARTS AT:
             FT2
10:11:12     02/02/1994     PROC-ID      001T    DP-TYPE: DP386
APS:         S0-EF0.20.059  AMO-APS:     B0-EF0.20.059
**>
```

3.2 SAVEON-Kommando (Einschalten des Patch)

Mit diesem Kommando wird die Patchfunktion eingeschaltet, um Code oder Daten auf der Hard Disk verändern zu können. Nach SAVEON werden alle folgenden SET-Instruktionen auf der Hard Disk ausgeführt. Der Arbeitsspeicher bleibt unverändert. Es ist möglich mit den SET Kommandos (z.B in einer Runfile

oder in einem Kommandostapel) zuerst die Änderungen im Speicher durchzuführen (im Zustand SAVEOFF), die Änderungen zu testen und dann mit den identischen SET Kommandos (Runfiles ,Stapel) im Zustand SAVEON die Änderungen auf der Hard Disk nachzuziehen. Dadurch ist gesichert, dass sowohl im Speicher als auch auf der Hard Disk die Änderungen identisch sind. Das 'Patchen' kann nicht an einen Trigger gekoppelt werden. SET Anweisungen an einem Testpunkt werden auch nach SAVEON nur im Speicher ausgeführt. Nachladbare Subsysteme (AMO) können nicht mit DEBUG gepatcht werden. Im Programm nicht initialisierte Daten müssen dynamisch gepatcht werden (durch Einfügen eines entprechenden Codes in einem Patchbalkon). Die Patchfunktion im DEBUG ist nur als Notlösung zu betrachten. Sie bietet keine Unterstützung hinsichtlich Verwaltung und Aufzeichnung der durchgeführten Veränderungen.

Nach jedem RUN Kommando und nach jeder Rückkehr in die Steuerebene wird der Zustand SAVEOFF gesetzt. Der Zustand ist am Promptzeichen (Systemzustand: #) erkennbar.

IMPORTANT: Vor Ausführung eines Steuerkommandos, welches von der interaktiven Ebene aus angestoßen wird, wird der Systemzustand wieder auf SAVEOFF gesetzt.

3.3 SAVEOFF-Kommando (Ausschalten des Patch)

Mit diesem Kommando wird die Patchfunktion ausgeschaltet. SET-Instruktionen wirken nur mehr im Speicher.

3.4 TERM-Kommando (Beenden einer DEBUG Session)

Durch dieses Kommando wird die DEBUG Session beendet. Es werden alle Testaufträge deaktiviert. Es wird gewartet bis die gerade laufenden Testpunktbearbeitungen beendet sind. Falls dies nicht innerhalb von 5 Sekunden geschieht, so wird bei der Termmeldung eine Warnung mit ausgegeben. Diese Meldung wird auch ausgegeben, wenn es noch belegte Stackbereiche gibt, obwohl keine Testpunktbehandlung mehr läuft. Falls mit Protokoll-File gearbeitet wird, wird vor dem Schliessen der Datei der Tracebufffer noch vollständig gesichert.

Beispiel:

```
TERM
0325I TERM      :DEBUG TERMINATED AT:
10:11:12        04/08/1993       PROC-ID      = 001T    01H
```

3.5 RESET-Kommando (Rücksetzen Tracebuffer)

Durch dieses Kommando wird der Tracebuffer neu initialisiert, der alte Inhalt ist verloren. Das Kommando wird abgewiesen, falls die Protokollierung auf eine zugewiesene Protokolldatei noch nicht beendet ist.

3.6 IN-Kommando (Anweisungen interaktiv eingeben)

Mit diesem Kommando gelangt man von der Steuerebene in die interaktive Ebene,in der sowohl Steuerkommandos als auch Anweisungen eingegeben werden können. Diese Ebene ist am Promptzeichen (zweites Zeichen ungleich BLANK erkennbar). Ist man bereits in der interaktiven Ebene so bleibt man in dieser Ebene, es ist aber das Save-Kennzeichen auf SAVEOFF gesetzt worden.

Beim ’IN’ Kommando kann mit dem Parameter ’RUNFILE=dateiname’ eine Datei angegeben werden, in die wahlweise Eingaben mitprotokolliert werden können. Dieses Mitprotokollieren wird mit RUNON und RUNOFF gesteuert. Existiert die angegebene Datei, so wird sie fortgesetzt. (Der Dateiname der Runfile setzt sich zusammen aus ’:AMD:R/’ und dem angegebenen Dateinamen).

Beispiel:

```
IN:RUNFILE=PROT1
```

IMPORTANT: Das Kommando ist in der Regel nur nach einem Restart notwendig, da in der interaktiven Ebene alle Kommandos möglich sind, und daher diese Ebene nicht verlassen werden muss.

3.7 RUN-Kommando (Anweisungen von Datei (Runfile) einlesen)

Mit diesem Kommando werden Anweisungen von einer Datei eingelesen. Die Datei, von der gelesen werden soll, wird über das Schlüsselwort ’INFILE’ angegeben. (Der Dateiname der Runfile setzt sich zusammen aus ’:AMD:R/’ und dem angegebenen Dateinamen).
Wird eine fehlerhafte Anweisung erkannt, so wird das Einlesen abgebrochen; dies kann verhindert werden, wenn der Parameter FUNC=CHECK angegeben wird. Die Funktion FUNC=CHECK wird jedoch nicht allgemein freigegeben, da durch Fehler nicht mehr der gewünschte Ablauf erzielt wird. Z.B können aus ursprünglich an Trigger gebundenen Operationen sofort ausführbare Instruktionen werden. Fehlermeldungen werden wie bei der direkten Eingabe ausgegeben. Bei Abbruch wird zusätzlich die Zeilennummer der ersten fehlerhaften Anweisung ausgegeben. Die gültigen Anweisungen werden wie in der interaktiven Ebene ausgeführt und quittiert. Der Abbruch durch Fehler beim Dateizugriff wird gesondert gemeldet.

Mit dem wahlfreien Parameter PROT=NO kann die Quittierung der AT-Anweisung unterdrückt werden. Dies ist sinnvoll, wenn in einer Runfile sehr viele Testpunkte definiert werden.

Beispiel:

```
RUN:INFILE=RFAT,PROT=NO
```

Runfiles sind Dateien auf der Hard Disk vom Typ BYTEFILE, deren Satzlänge 80 Bytes betragen muss. (Der Dateiname der Runfile setzt sich zusammen aus ':AMD:R/' und dem angegebenen Dateinamen).

3.8 FILE-Kommando (Steuerung des Protokollierens des Tracebuffers)

Durch dieses Kommando wird dem DEBUG eine Protokolldatei auf Hard Disk zugewiesen. Damit wird die Protokollierung des Tracebufferinhalts in diese Datei ermöglicht. Durch die Schlüsselwörter 'CURFILE' bzw. 'SUCFILE' wird angegeben, in welche Datei geschrieben werden soll. Ist nur eine Datei zugewiesen, so endet das Protokollieren, wenn die Datei voll ist. Sind zwei Dateien zugewiesen, so werden die beiden Dateien wechselweise beschrieben.

Mit dem Parameter CUR=CLOSE kann die mit CURFILE zugewiesene Datei geschlossen werden. Analog dazu wird mit der Angabe des Parameters SUC=CLOSE ein Schliessen der mit SUCFILE zugewiesenen Datei erreicht. Eine geschlossene Datei wird nicht mehr zur Protokollierung verwendet. Es erfolgt ein automatisches Umschalten auf die verbleibende Datei. Eine geschlossene Datei kann mit CURFILE bzw. SUCFILE erneut zugewiesen werden. (Der Dateiname der Protokolldatei setzt sich zusammen aus ':AMD:P/' und dem angegebenen Dateinamen).

Beispiele:

```
FILE :CURFILE=PROTFILE/001,SUCFILE=PROTFILE/002

FILE :CURFILE=PROTFILE/001

FILE :SUCFILE=PROTFILE/002

FILE :CUR=CLOSE,SUC=CLOSE

FILE :SUCFILE=PROTFILE/003,CUR=CLOSE
```

IMPORTANT: Wird die einzige noch zugewiesene Datei mit SUC=CLOSE oder CUR=CLOSE geschlossen, so wird das Protokollieren sofort abgebrochen. Es sind somit unter Umständen noch nicht alle Tracebuffereinträge geschrieben. Existiert eine im File zugewiesene Datei noch nicht, so wird sie vom DEBUG angelegt (Länge von 10000 Bytes). Existiert sie bereits, so wird sie überschrieben. Diese muss allerdings vom Typ SAM sein.

3.9 PRINT-Kommando (Ausdrucken Tracebuffer und Protokolldatei)

Dieses Kommando kann auch außerhalb einer DEBUG Session angewandt werden. Es gibt zwei Anwendungen des PRINT-Kommandos:

- Ohne Parameter INFILE wird der Tracebuffer ausgegeben,
- mit dem Parameter INFILE wird die angegebene Protokolldatei ausgegeben.

Es können nur geschlossene Dateien ausgedruckt werden. (Der Dateiname der Protokolldatei setzt sich zusammen aus ':AMD:P/' und dem angegebenen Dateinamen). Eine File wird geschlossen wenn sie vollgeschrieben ist (entsprechende Meldung geht an Benutzer), durch ein FILE Kommando mit CUR=CLOSE oder SUC=CLOSE oder durch das TERM Kommando.

Der Anwender kann die Ausgabe des Tracebuffers oder der Protokolldatei durch Angabe von Selektionskriterien einschränken. Durch Angabe des Parameters INF=CUE wird die Ausgabe ganz unterdrückt, es wird nur ausgegeben wieviele Einträge mit der angegebenen Selektion vorhanden sind.

Mögliche Kriterien der Selektion beim PRINT-Kommando sind:

- die Zeit (TIMEB, TIMEE)

  Es werden nur Einträge ausgegeben, die zwischen TIMEB und TIMEE entstanden sind. Defaultwert für TIMEB ist 00:00:00, für TIMEE 23:59:59.

- das Datum (DATEB, DATEE)

  Es werden nur Einträge ausgegeben, die zwischen DATEB und DATEE entstanden sind. Defaultwert für DATEB und für DATEE ist das aktuelle Tagesdatum.

- täglich (PERIOD)

  Es werden nur Einträge ausgegeben die jeweils zwischen TIMEB-Uhr und TIMEE-Uhr entstanden sind, und zwar an den Tagen zwischen DATEB und DATEE. Wird der Parameter PERIOD nicht angegeben, so werden alle Einträge ausgegeben,die zwischen dem Zeitpunkt DATEB,TIMEB und dem Zeitpunkt DATEE,TIMEE entstanden sind.

- Trigger-Eintrag (TLOG)

  Es werden nur die Trigger-Einträge ausgegeben. Damit wird die Programmverfolgung unterstützt. Wird der Parameter nicht angegeben, so werden alle Einträge ausgegeben (also auch TRACE, ACT, usw.).

- Label-Name (LABEL)

  Mit diesem Parameter werden nur die Einträge ausgegeben, die zu Testpunkten gehören, die mit dem angegebenen Namen übereinstimmen. Bei teilqualifizierter Labelangabe (z.B. LABEL=AB*) werden alle Einträge von Testpunkten ausgegeben die im angegebenen Teilnamen übereinstimmmen. Ist LABEL angegeben, so werden keine Instruktionen ausgegeben.

- die letzten Einträge (REC)

  Dies ist nur bei der Ausgabe des Tracebuffers möglich. Die Ausgabe wird auf die letzten n Einträge begrenzt, wobei von diesen Einträgen nur die ausgegeben werden, die von der Selektion betroffen sind. Ist zusätzlich der Parameter TEC angegeben, so bezeichnet REC die Anzahl der Einträge, die ausgegeben werden sollen (unter Berücksichtigung der Selektion) (mögliche Werte für n: 1 bis 255).

- ersten Eintrag definieren (TEC)

  Dies ist nur bei der Ausgabe des Tracebuffers möglich. TEC gibt die Zahl des Eintrags an, bei welchem die Ausgabe begonnen werden soll. Gezählt wird immer vom zeitlich ältesten Eintrag an. Wird TEC nicht angegeben,so wird beim zeitlich ältesten Eintrag begonnen. Wird der Trace buffer während der Ausgabe beschrieben, so wird es vorkommen, dass der auszugebende Eintrag überschrieben wird. Dies wird gemeldet und der Print wird abgebrochen. Sind im Tracebuffer nicht soviele Einträge, wie durch TEC angegeben, so wird TEC ignoriert. (mögliche Werte : 1 bis 65535)
  Jede Kombination dieser Selektionskriterien ist möglich.

Beispiele:

```
PRINT:TIMEE=14:30:00,DATEB=10/01/83,TLOG=ON
                                  Jahr
                               Monat
                            Tag
```

Aus dem Tracebuffer werden nur die Trigger-Einträge die zwischen dem 10.1.83, 0.00 Uhr und heute bis 14.30 Uhr entstanden sind.

```
PRINT:INFILE=PROTFIL/001,TIMEE=15:20:00,DATEB=17/02/84
             PERIOD=DAILY
                                    Sekunde
                                 Minute
                              Stunde
```

Aus der angegebenen Protokolldatei werden alle Einträge, die seit dem 17.02.84 bis zum heutigen Tag jeweils von 00.00 Uhr bis 15.20 Uhr entstanden sind, ausgegeben.

```
PRINT:LABEL=A*,REC=5,TLOG=ON
```

Aus dem Tracebuffer werden die letzten 5 Triggereinträge, deren Label mit ’A’ beginnt, ausgegeben.

```
PRINT:LABEL= *,INF=CUE,TLOG=ON
```

Es wird ausgegeben wieviele Triggereinträge im Tracebuffer vorhanden sind.

```
PRINT:TEC=60000,REC=1
```

Es wird der letzte Eintrag im Tracebuffer ausgegeben. (Tracebuffer fasst keine 60000 Einträge)

```
PRINT:TEC=333,REC=5
```

Beginnend bei dem zur Zeit 333ten Eintrag werden 5 Einträge ausgegeben.

3.10 TRIGGON-Kommando (Einschalten Triggermodus)

Mit diesem Kommando wird der Modus eingeschaltet in dem das Testpunktgeschehen mitprotokolliert wird.
Dieser Modus wird Triggermodus bezeichnet.
Das Mitprotokollieren erfolgt nur über die DP Schnittstelle via Miniterminal Handler (MTH), unabhängig davon, ob der Dialog über AMO DEBUG oder über Miniterminal Handler geführt wird. Bevor der Triggermodus eingeschaltet werden kann, muss die Triggerselektion eingegeben worden sein (siehe SEL Kommando). Wird das TRIGGON-Kommando dennoch vorher gegeben, so wird vom DEBUG ein SEL-Kommando mit den Default Selektionskriterien generiert. Das Triggern beginnt mit dem Eintrag, der zum Zeitpunkt des Einschaltens der letzte Eintrag im Tracebuffer ist. Erfolgt das Eintragen im Tracebuffer schneller als mitprotokolliert werden kann, so wird eine entsprechende Meldung ausgegeben. Es wird dann wieder mit dem jüngsten Eintrag fortgesetzt. Der Triggermodus ist unabhängig vom Protokollieren auf eine Hard Disk (Protokolldatei). Auch wenn Ausgaben erfolgen, können Eingaben durchgeführt werden. Dazu muss (am Qume) die ESC-Taste gedrückt werden (H’1B). Darauf wird das Triggern solange unterbrochen, bis eine Eingabe mit der RETURN-Taste abgeschlossen wird. Werden die Eingaben mit ESC-Taste abgeschlossen, so bleibt das Triggern weiterhin unterbrochen.

3.11 TRIGGOFF-Kommando (Ausschalten Triggermodus)

Mit diesem Kommando kann das Mitprotokollieren wieder abgeschaltet werden.

3.12 SEL-Kommando (Setzen der Selektion für Triggermodus)

Mit diesem Kommando können die Selektionen für den Triggermodus gesetzt werden. Dieses Kommando bewirkt noch kein Einschalten des Triggermodus. Die Kriterien bleiben so lange gesetzt, bis ein neues SEL Kommando gegeben wird. Dies kann auch erfolgen, wenn der Triggermodus bereits eingeschaltet ist. Syntaktisch ist das SEL-Kommando mit dem PRINT-Kommando identisch, d.h. es können die gleichen Parameter angegeben werden wie beim PRINT. Es werden jedoch die Parameter INFILE, TEC, REC und INF nicht ausgewertet. Ist LABEL nicht angegeben, so wird der Defaultwert LABEL=’*’ angenommen. Soll das Mitprotokollieren über Mitternacht (00.00 Uhr) hinaus erfolgen, so muss der Parameter DATEE angegeben werden, da das Tagesdatum (zum Zeitpunkt der Eingabe) Defaultwert ist.

3.13 LIST-Kommando (Ausgabe von Listen)

Mit diesem Kommando können folgende Listen ausgegeben werden:

Liste aller Trigger, an denen Tasks angehalten wurden,

- Liste aller DEF-Symbole,
- Liste der Standard DEF-Symbole,
- Liste der neu definierten DEF-Symbole,
- Liste aller DEBUG Variablen,
- Liste aller Triggerpunkte bzw. Breakpunkte
- Liste aller Macros.

Die gewünschte Liste wird durch das Schlüsselwort ’TAB’ angegeben.

Beispiele:

```
LIST:TAB=STOP      (Ausgabe der angehaltenen Tasks)
LIST:TAB=DEF       (Ausgabe der DEF-Symbole)
LIST:TAB=DEFS      (Ausgabe der Standard DEF-Symbole)
LIST:TAB=DEFN      (Ausgabe der neu definierten DEF-Symbole)
LIST:TAB=DCL       (Ausgabe der DEBUG Variable)
LIST:TAB=AT        (Ausgabe der Testaufträge)
LIST:TAB=MAC       (Ausgabe der Macros)
```

Beispiele (mit Ausgabe):

```
LIST:TAB=AT!
0440I TAB=AT  :TESTORDER TABLE :

0463I TAB=AT  :NAME      COUNTER  STATUS    KIND      ADDRESS
              19083A28   00000    ACT        BREAK    1908:3A28
              CPEVT      00002    ACT        TEST     65C8:06A4
0436I TAB=AT  :END OF TABLE

LIST:TAB=DCL!
0437I         :TABLE OF DEBUGVARIABLES:
TAB=DCL
```

```
0466I          :NAME                 MODE     VALUE     ADDRESS
TAB=DCL
               A                     BYTE     00        5990:2921
               CPB_IDX               INT      0000      5990:2940
               LODEN                 INT      0000      5990:295F
               LINE                  INT      0000      5990:297E
               L                     INT      0000      5990:299D
               K                     INT      0000      5990:29BC
               J                     INT      0000      5990:29DB
               I                     INT      0000      5990:29FA
0436I          :END OF TABLE
TAB=DCL


LIST:TAB=DEF
!
0434I          :TABLE OF DEF-SYMBOLS:
TAB=DEF


0465I          :NAME                 ADDRESS   LENGTH    TYP
TAB=DEF
               PATCH1                5948:000  00001     STD
                                     C
               PATCH2                5948:001  00001     STD
                                     E
               PATCH3                5948:003  00001     STD
                                     0
               GG_P_AR_ON            5948:004  00001     STD
               E                     2
               GG_P_AR_TW            5948:005  00001     STD
               O                     0
               LANGUAGE              59F0:001  00001     STD
                                     4
               ACTIN                 59F0:002  00001     STD
                                     E
               ACTOP                 59F0:002  00001     STD
                                     D
               DEACTIN               59F0:002  00001     STD
                                     C
               DEACTOP               59F0:002  00001     STD
                                     B
               SETIN                 59F0:002  00001     STD
                                     7
```

```
                SETOP                59F0:002  00001     STD
                                     6
                WRITE_IN             59F0:000  00001     STD
                                     E
0436I           :END OF TABLE
TAB=DEF
```

3.14 DEF-Kommando (Definieren Symbole)

Kommando DEF ordnet dem angegebenen Namen (max. 26 Zeichen) eine Adresse und eine Länge zu. Diese DEF-Symbole können z.B. der AT- und D-Anweisungen statt der numerischer Adressen verwendet werden.

Die DEF-Anweisung hat zwei Formate: entweder wird

- die Adresse, die dem Symbol zugeordnet werden soll, numerisch angegeben, oder
- durch ein bestehendes DEF-Symbol und eine Relativadresse (Offset).

Im ersten Fall muss eine Länge angegeben werden, im zweiten Fall wird dem neuen Symbol die Länge des alten zugewiesen, wenn sie nicht explizit angegeben wird.

Beispiele:

```
DEF &MARKE 1238:5678 (1);

DEF &BUFFER 4E68:10D0 (20);

DEF &HALT = &MARKE + 20;

DEF &BUFFER2 = &BUFFER + T'50 (200);
```

Erläuterung:

Der Adresse 1238:5678 ist nun in der Länge 1 der Name &MARKE zugeordnet und der Adresse 4E68:10D0 ist in der Länge H'20 der Name &BUFFER zugeordnet. (Mit Kommando 'D &BUFFER;' würde der Speicherinhalt ab dieser Adresse in der Länge H'20 ausgegeben werden). Anschliessend wird dem Symbol &HALT die Adresse 1238:5698 in der Länge 1 zugeordnet und dem Speicherplatz &BUFFER2 die Adresse von &BUFFER mit um dezimal 50 erhöhtem Offset in der hexadezimalen Länge 200, also die Adresse 4E68:1102 .

IMPORTANT: Für einige Anwendungen sind bereits Symboldefinitionen vorgeleistet. Siehe dazu die folgenden Abschnitte.

Standard DEF-Symbole : Unterdrücken von Quittungen

Zur Steuerung der Einträge von Anweisungen in den Tracebuffer und auf Terminal gibt es folgende Symbole:

```
&ACTIN,        &ACTOP         (ACT Anweisungen)
&DEACTIN,       &DEACTOP      (DEACT Anweisungen)
&SETIN,        &SETOP         (SET Anweisungen)
```

Die Voreinstellung ist TRUE d.h. die Anweisungen werden in den Tracebuffer geschrieben bzw ausgegeben.
Mit 'SET <symbol> = FALSE;' wird das Eintragen der entsprechenden Anweisung in den Tracebuffer unterdrückt und mit 'SET <symbol> = TRUE;' wieder ermöglicht.

Beispiele:

```
SET &SETOP = FALSE;
```

SET-Operationen werden nicht mehr in den Tracebuffer eingetragen.

```
SET &ACTIN = TRUE;
```

ACT-Instruktionen werden wieder in den Tracebuffer eingetragen)

Standard DEF-Symbole : Instruktionen in den Tracebuffer schreiben

```
&WRITE_IN
```

Die Voreinstellung ist FALSE d.h. die Instruktionen werden nicht in den Tracebuffer geschrieben bzw ausgegeben.
Mit 'SET &WRITE-IN = TRUE!' wird das Eintragen in den Tracebuffer eingeschaltet und mit 'SET &WRITE_IN = FALSE!' wieder ausgeschaltet.

Standard DEF-Symbole : Adressen von Anschlussroutinen

```
&PATCH1

&PATCH2

&PATCH3

&GG_P_AR_ONE

&GG_P_AR_TWO
```

Dies sind Symbole von Adressen, die vom DEBUG angesprungen werden, wenn die Operation EXEC an einem Testpunkt angegeben ist. Es stehen dann bei den ersten drei Symbolen 18 Bytes, sonst 14 Bytes zur Verfügung, in die codiert werden kann (z.B. der Absprung in einen Patchbalkon). Die Register des Testobjects stehen nicht zur Verfügung.

Standard DEF-Symbole: Sprache auswählen

Die Sprache mit der die Fehlermeldungen ausgegeben werden, kann zwischen DEUTSCH und ENGLISCH gewählt werden. Die Voreinstellung ist englisch.

Mit

```
SET &LANGUAGE = 'D'!
```

wird auf die deutsche Sprache umgeschaltet und

Mit

```
SET &LANGUAGE = 'E'!
```

wird auf auf die englische Sprache umgeschaltet.

3.15 REDEF-Kommando (Redefinieren Symbol)

REDEF weist einem bereits existierenden Symbol eine neue Adresse und/oder Länge zu. Wird keine Länge angegeben, so wird die Länge des alten Symbols beibehalten. Wie bei der DEF Anweisung kann die Adresse, die dem existierenden Symbol neu zugewiesen wird, numerisch oder mit Hilfe eines DEF-Symbols angegeben werden.

Beispiele:

```
DEF &A 1238:0 (7);
DEF &B = &A + 100 (5);
REDEF &A = &B + 10;       /* &A: Adresse 1238:110, Länge 5 */
REDEF &B 6458:21 (1A);
REDEF &A E100:123;        /* &A: Adresse E100:123, Länge 5 */
```

3.16 DCL-Kommando (Definieren DEBUG-Variable)

DCL definiert Variablen (Name max. 26 Zeichen) mit den Modes %INT, %BYT, %POI, %CHAR oder %BOOL. In Klammern können bis zu 6 Variablen des gleichen Modes definiert werden. DEBUG Variable können mit SET Anweisungen verändert werden und mit D Anweisungen ausgegeben werden. Sie können in Adressmodifikationen und in Expressions (= Ausdrücke mit den Rechenoperationen + , - , * , /) verwendet werden.

Beispiele:

```
DCL (%ABC,%XYZ) %CHAR;

DCL %ZAHL %INT;

DCL %B %BOOL;

DCL %C %POI;
```

Es existieren jetzt die Variablen %ABC und %XYZ vom Mode Character, die Pointervariable %C, die Integervariable %ZAHL und die Bool'sche Variable %B, die man z.B. mit folgenden SET Anweisungen

```
SET %C = #1238:5678;

SET %B = TRUE;

SET %ZAHL = 5;

SET %ZAHL = %ZAHL + 1;
```

verändern oder deren Inhalt man z.B. mit

```
D %ZAHL,%B,%C;
```

ausgeben kann.

3.17 REDCL-Kommando (Redefinieren DEBUG-Variable)

Mit REDCL kann eine bereits existierende Variable mit einem neuen Mode versehen werden. In Operationen, in denen die Variable bereits verwendet wurde, ist diese Umweisung nicht wirksam. Durch die REDCL Anweisung wird die Variable nicht neu initialisiert.

Beispiele:

```
DCL %B %BOOL;

REDCL %B %INT;
```

Zunächst wird eine Bool'sche Variable %B definiert. Sie wird mit REDCL in eine Integervariable umgewandelt.

3.18 AT-Kommando (Definieren Testauftrag)

Mit AT wird ein Testauftrag für die angegebene Adresse definiert. Er ist mit dem Label (Name max. 8 Zeichen) ansprechbar. Die Operationen, die an dem Testpunkt ablaufen sollen, werden durch die Schlüsselwörter 'DO' und 'OD' geklammert. Es sind auch Testaufträge ohne Operationen zugelassen. (Anwendung: wenn nur das Erreichen des Testpunktes von Interesse ist). Innerhalb eines Testauftrages wird eine Eingabe mit einem Prompting angefordert, bei dem der Dialogzustand ':' ist (z.B. mit *:>)

IMPORTANT: Testaufträge müssen zuerst definiert werden, bevor man sie aktivieren kann. Eine Definition eines Testauftrages allein bewirkt noch keine Testaktivitäten.
Ein einmal definierter Testauftrag kann beliebig oft aktiviert und deaktiviert

werden, solange er nicht mit der REMAT Anweisung gelöscht wird.
Vor der Bearbeitung des AT Kommandos werden alle bisher noch nicht ausgeführten Anweisungen zwangsweise ausgeführt.

Die an einem Testpunkt gesammelten Daten werden in den Tracebuffer geschrieben und können parallel in einer Protokolldatei gesichert werden. Auf Terminal können sie mit PRINT ausgegeben werden.

Beispiele:

```
L1AX2:AT E680:C69 DO  .........  OD;
GRISMURD:AT &MARKE DO OD;
```

An der Adresse E680:C69 ist nun der Testauftrag LABEL definiert. Er kann mit ’ACT L1AX2;’ aktiviert werden. Beim Durchlaufen eines aktiven Testpunkts werden die nach dem DO angegebenen Operationen ausgeführt. An der durch &MARKE definierten Adresse ist der Testauftrag GRISMURD definiert. Bei diesem Testauftrag werden keine Operationen ausgeführt. Es wird nur das Erreichen dieses Testpunkts protokolliert.

3.19 REMAT-Kommando (Löschen Testauftrag)

Mit REMAT ist es möglich, deaktivierte Testaufträge zu löschen. Sie können später nicht mehr aktiviert bzw. durchlaufen werden. Die Namen gelöschter Testaufträge können wieder neu verwendet werden. Sollen alle Testaufträge gelöscht werden, so ist statt Labelname ’ALL’ anzugeben. Hierbei ist zu beachten, dass nur die gerade deaktiven Testpunkte gelöscht werden. Kann ein Testpunkt nicht gelöscht werden, so erfolgt eine Meldung.

Beispiele:

```
LAB1: AT 1238:56 DO OD;
REMAT LAB1;
REMAT ALL;
```

Zunächst wird der Testauftrag LAB1 definiert. Er wird mit REMAT LAB1 wieder gelöscht. REMAT ALL löscht alle deaktiven Testaufträge.

3.20 MDEF-Kommando (Definieren Macro)

Mit MDEF wird ein Macro definiert, der über seinen Macro-Namen (max. 8 Zeichen) angesprochen wird.
Es können 20 Macros definiert werden.
Durch die MDEF-Instruktion werden die folgenden Operationen (bis zum MEND)

zu einer Macro-Operationen zusammengefasst. Innerhalb einer Macrodefinition wird eine Eingabe mit einem Prompting angefordert, bei dem der Dialogzustand '.' ist (z.B. mit *.>)

Vor der Bearbeitung des MDEF Kommandos werden alle bisher noch nicht ausgeführten Anweisungen zwangsweise ausgeführt.

Die Macro-Operation kann durch den Aufruf des Macros (MCALL) zur Ausführung gebracht werden.

Beispiel:

```
MDEF      MACNAME1;
          D .....;
          SET .......;
          SET .......;
MEND;
```

3.21 MCALL-Kommando (Macro-Operation aufrufen)

Mit MCALL wird ein zuvor definierter Macro aufgerufen.

Beispiel:

```
LABEL:     AT         400:5 DO
                      MCALL MACNAME1;
          OD;
MDEF      MACNAME2;
          ........;
          MCALL MACNAME1;
          ........;
MEND;
```

3.22 REMMAC-Kommando (Löschen Macro)

Mit REMMAC wird ein vorher definierter Macro für ungültig erklärt. Der Macro kann durch Neudefinition (mit gleichem Namen) wieder aktiviert werden. Ein mit REMMAC gelöschter Macro wird beim MCALL ignoriert. Der MCALL wird erst wieder ausgeführt, wenn der Macro mit seinem ursprünglichen Namen definiert ist.

Mit REMMAC wird kein Speicherplatz in der Macro-Tabelle frei. Eine volle Tabelle bleibt somit voll.

Beispiel:

```
REMMAC MACNAME1;
```

3.23 RUNON-Kommando (Runfile Online: Einschalten Protokollierung)

Nach dieser Instruktion werden die nachfolgende Eingaben in die beim 'IN'-Kommando angegebene Datei (Runfile) geschrieben. Die RUNON Anweisung muss in einer eigenen Zeile stehen. Es werden auch die fehlerhaften Eingaben protokolliert. Nach dem 'IN'-Kommando ist RUNOFF eingestellt. Falls beim IN keine Datei zugewiesen wurde, ist diese Anweisung wirkungslos.

3.24 RUNOFF-Kommando (Runfile Online: Abschalten Protokollierung)

Diese Instruktion beendet die Protokollierung der Eingaben in die Datei. Es kann beliebig oft zwischen RUNON und RUNOFF gewechselt werden. Nach dem 'IN'-Kommando ist RUNOFF eingestellt.

IMPORTANT: Die erzeugten Dateien können als Runfiles in dem RUN-Kommando angegeben werden.
Die RUNOFF Anweisung soll in einer eigenen Zeile stehen, um sicherzustellen dass die Eingabe, die in der RUNOFF vorkommt geschrieben wird.

3.25 EXEC-Kommando (Anschlussroutine starten)

Mit EXEC <routine> werden Anschlussroutinen gestartet. d.h. DEBUG verzweigt in die Prozedur, die <routine> zugeordnet ist. Die Prozeduren sind vom DEBUG als Dummy Prozeduren vorgeleistet. Sie müssen erst mittels Patchtechnik mit den gewünschten Befehlen versorgt werden. In der Regel ist ein zusätzlicher Patchbalkon nötig, da die Dummies nur 14 bzw. 18 Bytes lang sind. Als Alternative können die Dummies in der DEBUG Source durch die gewünschten Prozeduren ersetzt werden. Dies setzt aber eine Neuproduktion voraus.

Beispiel:

```
EXEC GG_P_AR_ONE;
```

3.26 STOP-Kommando (Task anhalten)

Mit STOP wird jede Task, die einen Testpunkt erreicht, an dem die STOP Anweisung definiert ist, in den Wartezustand versetzt. Alle anderen Tasks laufen weiter. Das Anhalten einer Task wird am Terminal angezeigt. Zusätzlich zur STOP-Meldung werden die aktuellen Registerstände des Prozessors der angehaltenen Task ausgegeben.

3.27 CONT-Kommando (Task fortsetzen)

Mit CONT werden alle an einem Testpunkt angehaltenen Tasks wieder fortgesetzt. Mit 'CONT ALL;' werden alle Tasks fortgesetzt.
Sind an einem Testpunkt mehrere Tasks gestoppt, so können sie nur gemeinsam fortgesetzt werden.

Beispiele:

```
CONT GRISMURD;
CONT ALL;
```

3.28 BREAK-Kommando (Prozessor anhalten)

Wird diese Anweisung als Instruktion angegeben, so werden zwei Aktionen ausgeführt.

- Zum ersten wird der Breakmodus eingeschaltet, sofern er noch ausgeschaltet ist. Dies hat zur Folge, dass ab diesem Zeitpunkt die BREAK-Operationen wirksam sind.
- Zum Zweiten wird der Prozessor angehalten. Das erste Promptzeichen (Systemzustand) wird in diesem Fall auf '%' gesetzt.

Als Operation an einem Testpunkt wird der Prozessor nur dann angehalten, wenn der Breakmodus bereits eingeschaltet ist. Ansonsten wird diese Operation mit einer entsprechenden Meldung ignoriert. Wurde der Prozessor von der Break Operation angehalten, so wird eine Meldung ausgegeben. Das erste Promptzeichen (Systemzustand) wird in diesem Fall auf '&' gesetzt. Es sind die Register zur Adressierung verfügbar, D #R ist erlaubt, D prozedurlocal ist möglich, bei VIEW und INSP beziehen sich die Angaben T* und J* auf die Task die den Prozessorstop ausgelöst hat.

3.29 GO-Kommando (Prozessor fortsetzen)

Diese Anweisung besteht aus zwei Aktionen.

- Zum Ersten wird der Breakmodus eingeschaltet, sofern er noch ausgeschaltet ist. Dies hat zur Folge, dass ab diesem Zeitpunkt die BREAK-Operationen wirksam sind.
- Zum Zweiten wird der Prozessor aufgetaut (fortgesetzt), falls er eingefroren (angehalten) war. Die Aktion hat keine Wirkung auf die Definitionen von Breakpunkten. Das erste Promptzeichen (Systemzustand) wird in diesem Fall auf '?' gesetzt.

3.30 GOFOR-Kommando (Breakmodus ausschalten)

Diese Anweisung besteht aus mehrere Teilaktionen.

- Der Breakmodus wird ausgeschaltet. Dies hat zur Folge, dass ab diesem Zeitpunkt die BREAK-Operationen nicht mehr wirksam sind.
- Der Prozessor wird aufgetaut (fortgesetzt), falls er eingefroren (angehalten) war. Die Breakpunkte werden deactiviert und gelöscht. Das erste Promptzeichen (Systemzustand) wird in diesem Fall auf '*' gestezt

3.31 GOTIL-Kommando (Breakpunkte definieren)

Diese Anweisung besteht aus mehrere Teilaktionen.

- Der Breakmodus wird eingeschaltet. Dies hat zur Folge, dass ab diesem Zeitpunkt die BREAK-Operationen wirksam sind.
- Der Prozessor wird aufgetaut (fortgesetzt), falls er eingefroren (angehalten) war.

An den angegebenen Adressen werden Breakpunkte definiert und aktiviert. Der Name der Breakpunkte wird aus der angegebenen Adresse konstruiert: Die numerischen Angaben von Selektor und Offset werden mit führenden Nullen zu einem 8 Zeichen langen Namen zusammengesetzt. (Aus 4F8:120 wird der Name 04F80120). In den Debugmeldungen werden die Breakpunkte mit diesen Namen identifiziert. Falls ein Breakpunkt erreicht wird, wird der Prozessor angehalten und über Miniterminal Handler (MTH) eine Meldung ausgegeben. Das erste Promptzeichen (Systemzustand) wird in diesem Fall auf '&' gesetzt. Es sind die Register zur Adressierung verfügbar, D #R ist erlaubt, D prozedurlocal ist möglich, bei VIEW und INSP beziehen sich die Angaben T* und J* auf die Task die den Prozessorstop ausgelöst hat. Es sind aber weiterhin nur Instruktionen und keine Operationen (also kein IF, DOWHILE, MCALL,....) zugelassen..

Beispiel:

```
GOTIL 2BF8:2356;
GOTIL 2BF8:1456,2BF8:134,2E50:345;
```

IMPORTANT: Liegt die Adresse im Miniterminal Handler (MTH) oder DEBUG so wird der Prozessor zwar angehalten, es sind aber keine Register verfügbar. Das erste Promptzeichen ist in diesem Sonderfall '%' ;

3.32 REMOVE-Kommando (Breakpunkt löschen)

Die mit GOTIL definierten Breakpunkte werden mit diesem Kommando deaktiviert und gelöscht . Die Breakpunkte werden nicht mit dem generierten Namen sondern mit ihrer Adresse angesprochen, also genau so wie beim GOTIL Kommando.

3.33 DOWHILE-Kommando (Schleife definieren)

Mit DOWHILE werden Schleifenoperationen definiert. Die Schleifendefinition wird mit 'OD' beendet. Eine Schleife muss mindestens eine Operation beinhalten. Innerhalb DOWHILE wird eine Eingabe mit einem Prompting angefordert, bei dem der Dialogzustand '-' ist (z.B. mit *->). Die Ausführung der Operationen ist abhängig von der angegebenen Bedingung. Die Bedingung ist eine beliebige logische und arithmetische Verknüpfung von DEBUG Variablen, Konstanten oder Adressen (Länge 1,2 oder 4).
DOWHILE kann 7-fach geschachtelt werden.

Um bei der Bearbeitung eines Testpunkes Endlosschleifen zu vermeiden, wird die Summe aller Schleifendurchläufe eines Testpunkts auf 255 Durchläufe beschränkt.

Beispiele:

```
       DCL %B %BOOL;
       DCL %I %INT;
       SET %B = TRUE;
       SET %I =1;

A1:AT  &MARKE DO
       DOWHILE %B;
                                IF      1250:AB = %I
                                THEN    SET %B = FALSE;
```

```
                                FI;
                                SET %I = %I + 1;
        OD;
OD;

MDEF    MACRO1;
        DOWHILE                 %I < 4;
                                IF %B THEN SET %I = %I + 1; FI;
                                WRITE 'SCHLEIFENDURCHLAUF';
        OD;
MEND;
```

3.34 ACT-Kommando (Aktivieren Testauftrag)

ACT aktiviert einen bzw. mehrere vorher mit AT definierte Testaufträge. Der Testauftrag wird mit seinem Label angesprochen. Sollen alle Testaufträge aktiviert werden, so ist als Labelname 'ALL' anzugeben. Der Interrupt INT3 wird beim ersten ACT nach dem Restart zum DEBUG umgeleitet.

Beispiele:

```
ABC:AT H'1238:H'17 DO ACT LABEL;OD;
ACT ABC,XYZ;
ACT ALL;
```

Durch den Aufruf 'ACT ABC,XYZ;' werden die Testaufträge ABC und XYZ aktiviert. Wird der Testpunkt ABC durchlaufen, so wird zu diesem Zeitpunkt auch der Testauftrag LABEL aktiviert. Mit dem Aufruf 'ACT ALL;' werden alle deaktiven Testaufträge aktiviert.

3.35 DEACT-Kommando (Deaktivieren Testauftrag)

DEACT deaktiviert Testaufträge. Der Testauftrag wird mit seinem Label angesprochen. Sollen alle Testaufträge deaktiviert werden, so ist als Labelname 'ALL' anzugeben.

Beispiele:

Testauftrag LABEL deaktivieren

```
DEACT LABEL;
```

GRISMURD und LABEL deaktivieren

```
DEACT GRISMURD,LABEL;
```

Alle aktiven Testaufträge deaktivieren

```
DEACT ALL;
```

Beim Erreichen des Testpunkts XYZ alle Testaufträge deaktivieren einschließlich des Testauftrages XYZ selbst

```
XYZ:AT 2220:1111 DO DEACT ALL;OD;
```

3.36 LOGON-Kommando (Triggereintrag einschalten)

Nach LOGON wird beim Erreichen eines Triggers eine Triggermeldung in den Tracebuffer geschrieben. Nach dem START-Kommando ist LOGON eingestellt.

3.37 LOGOFF-Kommando (Triggereintrag ausschalten)

Nach LOGOFF wird beim Erreichen eines Triggers keine Triggermeldung in den Tracebuffer geschrieben. Dies gilt nicht für Triggerpunkte bei denen Operations protokolliert werden müssen. Für diese wird auch bei LOGOFF eine Triggermeldung abgesetzt.

Beispiel:

```
A:   AT     &MARKE DO
            IF &           SPEICHER(1) = 1
                           THEN D &SPEICHER (50);
            FI;
     OD;
```

Bei LOGOFF wird nur dann ein Eintrag des Triggers gemacht, wenn auch das D Kommando ausgeführt wird. In allen anderen Fällen erfolgt kein Eintrag in den Tracebuffer.

3.38 D-Kommando (Display Speicherinhalt)

Mit D kann man Speicherinhalte an den angegebenen Code- oder Datenadressen in beliebiger Länge, DEBUG Variable, sowie den Stackinhalt und die Registerinhalte ausgeben. Weist der Bereich, der ausgegeben werden soll über den definierten Bereich hinaus, so wird das Display bis zum Ende des durch den Selektor definierten Bereiches ausgegeben. Es erfolgt eine Meldung, dass das Display gekürzt wurde. Die Darstellung des Inhaltes wird durch die Angabe eines ’display pictures’ beeinflusst. Er kann hexadezimal, binär, als

Charakterstring oder im Dumpformat aufbereitet werden. Wird nichts angegeben, erfolgt die Ausgabe im Dumpformat.
Es ist wahlweise die Angabe eines Auftragskennzeichens möglich, das bei der Displayquittung mit ausgegeben wird. Damit hat der Anwender die Möglichkeit ein Display nicht nur durch die Adresse zu identifizieren, sondern durch eine selbst vergebene Kennung. Diese Auftragskennung wird vor den Nutzdaten (Adresse = Inhalt) mit ausgegeben. Ist kein Auftragskennzeichen angegeben, so wird 0000 ausgegeben. Wird als Auftragskennzeichen #FFFF angegeben, so werden nur die Nutzdaten ausgegeben. Zum Aufbau der Ausgabe siehe Kapitel ’Meldungen’ Abschnitt ’Zusatzinformation’.

Beispiele:

```
(1) D 1238:5678 (1);
(2) D 1238:5678 TO 1238:5721;
(3) D ABC0:EF (T’70) %BIN;
(4) D &MARKE %CHAR #1717;
(5) D %B;
(6) D %STACK(20);
(7) D #R;
(8) D P*:A->(5);
(9) D (&MARKE,1238:5678(1))%char;
```

Erläuterungen:

(1) Es wird ein Byte der angegebenen Adresse hexadezimal (Defaultwert) ausgegeben.

(2) Es werden die Bytes zwischen Offset H’5678 und H’5721 des Selektors H’1238 ausgegeben (= 170 Bytes) .

(3) Ab der Adresse ABC0:EF werden 70 Bytes binär ausgegeben.

(4) Es werden die Bytes ab der durch &MARKE definerten Adresse in der implizit definierten Länge als Charakterstring ausgegeben. Unabdruckbare Zeichen (< H’20 bzw > H’7E) werden durch ’.’ ersetzt. Vor den Displaydaten (Adresse = Inhalt) wird das Auftragskennzeichen 1717 mit ausgegeben.

(5) Ausgabe des Wertes der DEBUG Variablen %B

(6) Ausgabe von 32 Bytes des Stackinhalts

(7) Ausgabe der Registerbelegung

(8) Ausgabe eines Aufrufparameters in der Länge 5. Es wurde angenommen, dass der Parameter als LOC Parameter definiert ist, deshalb wird dereferenziert.

(9) Es werden zwei Speicherbereiche ausgegeben, jedoch in der umgekehrten Reihenfolge: der zuletzt angegebene Bereich als erstes, der erste Bereich zum Schluss.Dies ist bei allen DS der Fall bei denen mehrere Bereiche angegeben wurden.

Die Beispiele (6), (7) und (8) sind nur am Testpunkt möglich oder interaktiv, wenn der Prozessor am Tigger angehalten wurde.

Beispiele (mit Ausgabe):

```
D H'65C8:H'10 (H'50)!
0614T DISPLAY :DISPLAY EXECUTED
0000
65C8:0010 = 1C 19 67 15 1C 19 4D 18 1C 19 1C 19 1C 19 1C 19
..g...M.........
65C8:0020 = 97 14 1C 19 1C 19 1C 19 1C 19 DE 12 37 16 37 16
............7.7.
65C8:0030 = 6E 10 DE 12 08 08 D8 08 DF 09 08 08 B2 0A 85 0B
n...............
65C8:0040 = 55 0C 25 0D 25 0D 25 0D 25 0D 25 0D F8 0D CB 0E
U.%.%.%.%.%.....
65C8:0050 = 9E 0F 1C 19 C7 13 1C 19 1C 19 1C 19 1C 19 1C 19
................

D #R!
0629T DISPLAY :DISPLAY REGISTER
CS:  IP:  DS:  SS:  SP:  BP:  AX:  BX:  CX:  DX:  SI:  DI:
ES:  FL:
65C8 06A4 65D0 B850 07E4 07E8 6200 1790 000B B85C 002F 01DE
0970 0216
```

3.39 TRACE-Kommando (Tracen Speicherinhalt)

Diese Anweisung ist identisch mit dem Kommando D (siehe Section 3.38, “D-Kommando (Display Speicherinhalt)”).

3.40 SET-Kommando (Modifizieren Speicherinhalt)

SET verändert Speicherinhalte und DEBUG Variable. Die Speicheradresse kann numerisch oder mit Hilfe von DEF-Symbolen und DEBUG Variablen angegeben werden. Pro SET-Anweisung können maximal 240 Bytes verändert werden. Es ist wahlweise die Angabe eines Auftragskennzeichens möglich, das bei der Setquittung mit ausgegeben wird. Damit hat der Anwender die Möglichkeit einen SET nicht nur durch die Adresse zu identifizieren, sondern durch eine selbst vergebene Kennung. Diese Auftragskennung wird vor den Nutzdaten (Adresse = Inhalt) mit ausgegeben. Ist kein Auftragskennzeichen angegeben, so wird 0000 ausgegeben. Wird als Auftragskennzeichen #FFFF angegeben, so werden nur die Nutzdaten ausgegeben.

Beispiele:

```
(1)   SET %B = TRUE;
(2)   SET %ABC = 'A';
(3)   SET %ZAHL = T'16+T'5 * T'17 #ABCD;
(4)   SET &ANFANG TO &ENDE = &BEREICH;
(5)   SET 18:34 (1) = B'00000101;
(6)   SET 20:H'45 (T'11) = H'20:H'88-> + T'4;
```

```
(7)   SET 988:65 TO 988:6A = 988:6B;
(8)   SET &ADDR(2) = H'12FC;
(9)   SET AB8:CD(3) = X'123856;
(10)  SET FE0:C(9) = X'90;
(11)  SET &ACTOP = FALSE;
(12)  SET #P1404 = 0;
```

Erläuterungen:

(1) Der DEBUG Variablen %B wird der Wert TRUE zugewiesen,

(2) der Variablen %ABC wird das abdruckbare Zeichen 'A' zugewiesen (Auftragskennzeichen ist ABCD) und

(3) der Variablen %ZAHL wird der Wert 101 zugewiesen.

(4) An die durch &ANFANG adressierte Stelle wird der durch &BEREICH adressierte Bereich in der Länge (&ENDE-&ANFANG+1) übertragen.

(5) An die Stelle H'18:H'34 wird eine 5 geschrieben.

(6) An die Speicherstelle H'20:H'45 wird in der Länge 11 der durch die rechte Seite adressierte Bereich übertragen. Der Inhalt von H'20:H'88 wird als Pointer interpretiert, um 4 erhöht und stellt dann die Adresse des zu übertragenden Bereichs dar.

(7) Dieses Beispiel ist identisch mit Beispiel (4) nur werden hier die Adressen numerisch angegeben.

(8) Der Adresse &ADDR wird der Wert H'FC und der Adresse &ADDR+1 der Wert H'12 zugewiesen; übertragen wird byteweise vertauscht (gilt nur für die Länge 2).

(9) Der Adresse H'AB0:H'CD wird der Wert H'12, der Adresse H'AB0:H'CD+1 der Wert H'34 und der Adresse AB0:CD+2 der Wert H'56 zugewiesen. Die Übertragung erfolgt also byteweise.

(10) An die Adresse FE0:C wird in der Länge 9 der Wert H'90 eingetragen. Falls auf der rechten Seite nur ein Byte steht wird die Länge als Wiederholungsfaktor interpretiert.

(11) Die Protokollierung von ACT-Operationen wird ausgeschaltet.

(12) Port 1404 wird auf 0 gesetzt (Watchdog wird ausgeschaltet).

3.41 VIEW-Kommando (Ausgeben von Objektinformationen)

Mit dem VIEW-Kommando werden Informationen über Objekte und über die Kataloge aufgelistet.

Objekte	Kataloge
JOB	MODCATI
TASK	MODCATA
SEGMENT	SYSCATI

Objekte	Kataloge
SEMAPHOR	SYSCATA
MAILBOX	
QUEUE	
POOL	
REGION	

Die Menge der betroffenen Objekte kann eingeschränkt werden auf diejenigen Objekte , die zu einem angegebenen Objekt gehören (z.B. alle Tasks, die zum einem Job gehören).

Beispiele:

```
VIEW MODCATI 5640:1300;
VIEW MAILBOX #4890:1302;
VIEW SEGMENT  2458:143;
VIEW TASK J*;
VIEW JOB;
```

3.42 INSP-Kommando (Ausgeben JOB und TASK Informationen)

Mit dem INSP-Kommando erhält man umfangreichere Angaben über ein Objekt. Für die eigene Task und den eigenen Job gibt es eine vereinfachte Syntax. Ansonsten muss das Token des Objektes angegeben werden. Dies kann direkt erfolgen (mit #) oder indirekt durch eine Adressangabe, an der das Token zu finden ist. Es werden dann der Typ des Objekts und die für dieses Objekt spezifischen Daten ausgegeben.
Als Objekte können auftreten: JOB, TASK, MAILBOX, POOL, BUFFER, SEGMENT, QUEUE, REGION UND SEMAPHPORE.

Beispiele:

```
INSP T*;
INSP J*;
INSP #1238:1300;
INSP  2458:135->+5;
```

3.43 IF-Kommando (Bedingungen formulieren)

IF macht die Ausführung von Operationen abhängig von den angegebenen Bedingungen. In diesen Bedingungen sind beliebige arithmetische und logische Verknüpfungen von DEBUG Variablen, Konstanten und Speicherinhalten (Länge

1, 2 oder 4) zugelassen. Das erste Promptzeichen (Systemzustand) wird innerhalb IF auf '-' gesetzt. IF kann 7-fach geschachtelt werden. Die Syntax (IF, THEN, ELSE, FI) ist identisch mit der von CHILL. Beispiel:

```
A1:AT          H'1238:H'5678 DO
               IF      %ABC > H'AB30:H'12(2)
               THEN    D   %B;
               IF          %A = &B(4)AND
                           %C < %A OR
                           %B = 1
               THEN SET %A =2;
               FI;
               ELSE SET %ABC = %XYZ;
               FI;
OD;
```

Leere ELSE-Zweige können auch weggelassen werden.

Mögliche Vergleichsoperatoren bei der Formulierung von Bedingungen sind :

EQ	gleich	(=)
NE	ungleich	(/=)
GT	größer	(>)
GE	größer	oder gleich (>=)
LT	kleiner	(<)
LE	kleiner oder gleich	(<=)
AND	und	
OR	oder	
NOT	nicht	
XOR	exklusives ODER	

3.44 Das Kommando ! (Anweisungen ausführen)

Durch Angabe eines Ausrufungszeichens werden alle Anweisungen die vorher nur mit ';' abgeschlossen wurden ausgeführt.
Wird eine Anweisung mit '!' abgeschlossen statt mit ';' , so wird sie sofort ausgeführt.
Bei Anweisungen innerhalb einer Testpunktdefinition oder Macrodefinition wird

diese Anweisung ignoriert. Wird die Tabelle mit den gespeicherten Instruktionen voll, so werden nach einer entsprechenden Meldung die Instruktionen automatisch ausgeführt.

Die Anzahl der Anweisungen, die gekettet werden können hängt von der Anzahl der gespeicherten Operationen ab. Es ist jedoch sichergestellt, dass zu jeder Zeit Intruktionen eingebeben werden können. Die Kettung wird abgebrochen (d.h. die Instruktionen werden ausgeführt), wenn eine AT oder MDEF Instruktion angegeben wird. Beispiel:

```
D 18:24 (2);
   .
   .
D 18:36 (2);
   !
```

oder zum Unterschied:

```
D 18:24 (2)!
   .
   .
D 18:36 (2)!
```

Im ersten Fall werden die D Anweisungen gemeinsam ausgeführt, im zweiten Fall sofort nach jeder Eingabe.

3.45 WRITE-Kommando (Kommentare schreiben)

Mit WRITE können Kommentare in den Tracebuffer in der Länge 1-80 eingetragen werden, um Einträge zu dokumentieren.

Beispiel:

```
WRITE 'Dies ist ein Kommentar';
```

3.46 END-Kommando (Beenden interaktive Ebene)

Mit diesem Kommando erfolgt der Übergang in die Steuerebene. Beim Übergang in die Steuerebene wird der Zustand SAVEON auf SAVEOFF gesetzt.Dieses Kommando wird nur noch benötigt, wenn ein PRINT Kommando eingegeben werden soll,welches länger als 79 Zeichen ist.

4 Adressierung

4.1 Adressierung numerisch und mit Hilfe von DEF-Symbolen

Im Prozessor-Baustein 80286 setzt sich eine Adresse zusammen aus

- dem Selektor

und

- einem Offset.

Eine numerische Adresse kann einem Symbol zugeordnet werden. Dieses Symbol kann in allen DEBUG Anweisungen statt der numerischen Adresse verwendet werden.

4.2 Adressierung von CHILL Variablen

4.2.1 Grund-Datentypen

Die Grund-Datentypen setzten sich zusammen aus:

```
addr        numerische Adresse oder DEF-Symbol
distance,
length,
index       Konstante, DEBUG-Variable oder arithmetischer
            Ausdruck
```

addressiertes Objekt	CHILL Syntax	DEBUG Syntax
Variable	VAR	addr
Pointer	POI ->	addr ->
Strukturkomponente	STR.FIELD	addr+distance
Array-Element	ARY(INDEX)	addr+index*length
gekettete Pointer	POI -> -> -> ->	addr -> -> -> ->
Feld eines Elementes eines Arrays von Strukturen	ARY(INDEX).FIELD	addr+index*length + distance
Element eines Arrays	STR.ARY(INDEX)	addr+distance +
einer Struktur		index * length

Table 6 Übersicht Grund-Datentypen

4.2.2 Kombinationen

Grund-Datentypen, durch Pointer adressiert

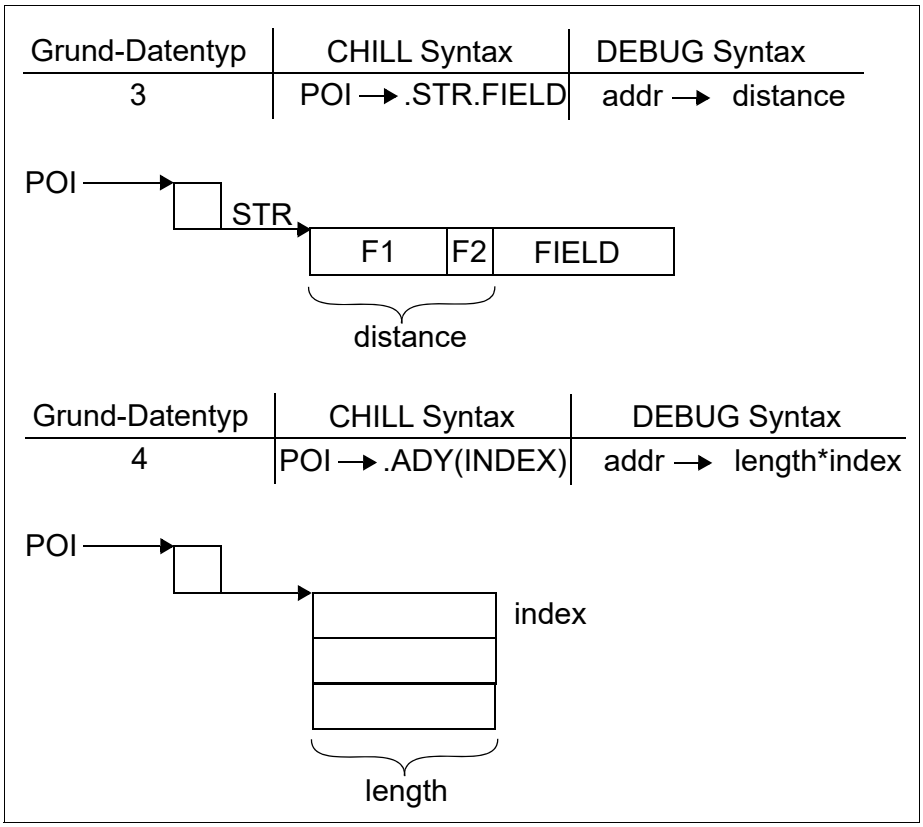

Figure 2 Grund-Datentyp 3 und 4 durch Pointer adressiert

Dereferenzierung von Grund-Datentypen

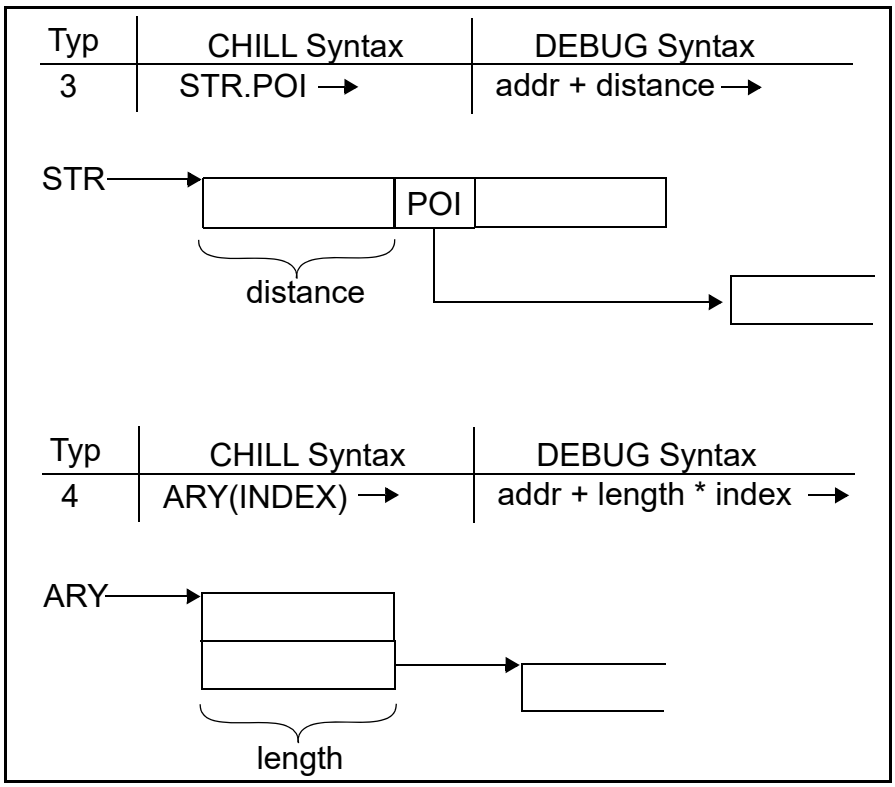

Figure 3 Grund-Datentyp 3 und 4 Dereferenzierung

IMPORTANT: Der Ausdruck, mit dem eine Adresse im DEBUG modifiziert ist, wird von rechts nach links ausgewertet. Ein Dereferenzierungspfeil (’->’) wirkt dabei als Klammerung.

5 Spezielle Anwendungen

5.1 Testen mit prozedurlokalen Variablen

Lokale Daten sind am Testpunkt mit DEBUG-Anweisungen adressierbar. Es sind die lokalen Variablen der Prozedur, in der der Testpunkt liegt und aller statisch umgebenden Prozeduren, sowie deren Parameter adressierbar. Statt Selektor und Offset werden in der Adresse Schachtelungstiefen (aus Chillcompilerlisting) angegeben. P* bezeichnet die aktuelle Prozedur, Pn (n = B bis Z) die statische Tiefe, wobei B die äußerste Ebene darstellt.

Beispiel:

```
PROC     :   AT     FF0:26 DO
                    D P*:2(4);
                    SET PC:F(2) = X'ABCD;
             OD;
```

5.2 Testen mit Prozessorstop

Hier sind die Unterschiede zu den entsprechenden ICE Anweisungen dargestellt.

GOTIL wirkt additiv, die mit den bisherigen GOTIL Anweisungen Statements bleiben als Breakpunkte erhalten. Gelöscht werden können sie mit der REMOVE Anweisung.

DEBUG hält vor der Ausführung des Statements an. Die Adresse des nächsten Befehls ist nicht verfügbar.

Der Prozessor wird dadurch angehalten, dass jede Task suspendiert wird. Die nicht als Task realisierten Abläufe laufen weiter. Ebenso werden die Tasks des DEBUG und des Miniterminal Handler (MTH) nicht suspendiert.

5.3 Globales Beispiel

Meldungen, die vom DEBUG ausgegeben werden, sind in diesem Beispiel nicht berücksichtigt. Jedoch ist das Prompting mit dem die nächste Eingabe angefordert der Eingabe vorangestellt. Die Nummer dient als Verweis zu den Erläuterungen.

```
1)    + >START
```

```
2)   **>FILE : CURFILE=PROTFILE/001
3)   **>DCL       %COUNTER          %INT;
     **>DCL       %DEACT            %BOOL;
     **>SET       %COUNTER          = 1;
     *!>OPL   :    AT   E320:       4C5          DO
     *:>               IF           %COUNTER  LT T'13
     *->          THEN SET %COUNTER = %COUNTER + 1;
     *->          ELSE DEACT OPL; SET %DEACT = TRUE;
     *->          FI;
     *:>          OD;
     *!>          ACT OPL!

                   <Quittungen>

     **>DEF       &BUFFER 4E60:H'10D0 (T'16);
     *!>D &BUFFER;

4)   *!>PRINT

                   <Quittungen für  Trace>
                   <Tracebufferausgabe>

5)   **>DEF &MESGEN E680:C69 (1);
     *!>AAA   :  AT    &MESGEN      DO
     *:>               D &BUFFER;
     *:>          OD;
     *!>ACT AAA!
                   <Quittungen>

     **>D &MESGEN %BIN!

                   <Quittung>

     **>BBB   :  AT    F618:4EF     DO
     *:>               DEACT AAA;
```

```
*:>          OD;
*!>ACT BBB;
*!>ACT AAA!

              <Quittungen>
**>D 8820:17C4 (7) %CHAR;
*!>DEACT AAA,BBB;
*!>END
              <Quittungen>

6)   * >PRINT :TLOG=ON
       <Tracebufferausgabe>

7)   * >IN
     **>DEF       &ONE H'EBD0:H'165C (1);
     *!>DEF       &TWO H'EBD0:H'166B (1);
     *!>DEF       &THREE = &ONE + 2;
     *!>SET       &ONE TO &TWO = H'90;
     *!>SET       &THREE = T'144;
     *!>D %COUNTER;
     *!>D %DEACT;

 8)  *!>      RUN : INFILE=RUNFILE1/003
              <Quittungen der Instruktionen>
              <Quittungen der Runfile>

9)   **>PRINT

10)  **>TERM

11)  + >PRINT : INFILE=PROTFILE/001
```

Erläuterungen zum globalen Beispiel:

ad 1) Die DEBUG Session wird gestartet. Zugleich Übergang in die interaktive Ebene.

ad 2) Die Datei PROTFILE/001 wird als Protokolldatei zugewiesen.

ad 3) Eingabe von Anweisungen:

Es werden die DEBUG Variablen %COUNTER vom Mode INTEGER und %DEACT vom Mode BOOL definiert.

Der Testpunkt OPL wird definiert.
Ist dieser Testpunkt aktiviert, so wird bei jedem Durchlauf der %COUNTER um 1 erhöht, solange er kleiner als 13 ist. Andernfalls wird der Testpunkt OPL deaktiviert und die Variable %DEACT auf TRUE gesetzt.

der Testpunkt OPL wird aktiviert.

Das Symbol &BUFFER wird definiert mit der Anfangsadresse des Buffers und der Länge 16.

Ausgabe des Inhaltes des durch das DEF-Symbol angegebenen Speichers.

ad 4) Der Tracebuffer wird ausgedruckt. Vorher werden die noch nicht ausgeführten Anweisungen abgearbeitet.

ad 5) Eingabe von Anweisungen:

Das Symbol &MESGEN wird definiert.

Der Testpunkt AAA wird definiert. Beim Durchlauf des aktiven Testpunkts AAA wird der durch &BUFFER adressierte Bereich in den Tracebuffer geschrieben in der durch &BUFFER implizit definierten Länge 1.

Der Testpunkt AAA wird aktiviert.

Der durch &MESGEN adressierte Speicherbereich wird in binärer Darstellung ausgegeben. (Länge ist explizit mit 3 angegeben)

Der Testpunkt BBB, bei dessen Durchlauf AAA deaktiviert werden soll, wird definiert.

Der Testpunkt AAA wird aktiviert. Falls er noch aktiv sein sollte, wird diese ACT Anweisung zurückgewiesen.

Der Testpunkt BBB wird aktiviert.

Ausgabe von 7 Bytes in Charakter-Darstellung

Erneute Aktivierung von AAA

Deaktivieren der Testpunkte AAA und BBB.

ad 6) Aus den Einträgen im Tracebuffer werden nur die Triggerinformationen ausgegeben.

ad 7) Eingabe von Anweisungen:

Definieren der Symbole &ONE und &TWO.

Das Symbol &THREE wird relativ zu &TWO (Offset+2) definiert, die Länge von &ONE wird übernommen.

Der Adressbereich von &ONE bis &TWO wird auf hexadezimal 90 gesetzt (16 Bytes).

In den mit dem Symbol &THREE adressierten Speicher wird der Wert 144 geschrieben.

Der Inhalt der Variablen %COUNTER und %DEACT wird ausgegeben.

ad 8) Aus der Kommandodatei RUNFILE1/003 werden Anweisungen eingelesen.

ad 9) Der gesamte Tracebuffer wird ausgedruckt.

ad 10) Die DEBUG Session wird beendet.

ad 11) Die gesamte Protokolldatei wird ausgedruckt.

6 DEBUG Meldungen

6.1 DEBUG-Meldungsklassen

Während der Anwendung des DEBUG kommt es zu vielen Situationen, die dem Benutzer mitgeteilt werden müssen. Entsprechend der Art des Ereignisses klassifiziert DEBUG die Meldungen in sechs Gruppen.

Fehlermeldungen: (E)

Mit Hilfe dieser Klasse meldet DEBUG Fehler, die bei internen Prüfungen entdeckt werden, insbesondere das Erkennen inkonsistenter Tabelleninhalte. Bei solchen Meldungen hilft in den meisten Fällen ein neu Aufsetzen der DEBUG Session (also TERM und START).

Syntaxmeldungen: (S)

Mit dieser Klasse werden alle formalen Fehler (Syntaxfehler) während der Kommandoeingabe gemeldet. Dazu gehören auch die Fehlermeldungen für Fehler in Anweisungsdateien.

Testauftragmeldungen: (T)

Mit dieser Klasse meldet DEBUG alle Ereignisse während der Ausführung von Anweisungen. Die Nutzdaten sind binär oder als CHarakter dargestellt.

Testauftragmeldungen: (H)

Identisch mit Klasse T. Die Nutzdaten sind hexadezimal aufbereitet.

Informationsmeldungen: (N)

Mit dieser Klasse sind die Meldungsnummern versehen, denen kein Langtext zugeordnet ist.

Informationsmeldungen: (I)

Diese Klasse enthält alle übrigen Meldungen (Informationen).

Meldungen des DEBUG können im allgemeinen in den Tracebuffer geschrieben werden und/oder auf das Terminal ausgegeben werden. Die folgende Tabelle enthält einen Überblick wo die Meldungen abgesetzt werden. Bei den Klassen E und I kann es in Einzelfällen Abweichungen geben.

Meldungsklasse	Ausgabe in Tracebuffer	Ausgabe auf DEBUG Terminal
Fehler (E)	ja	ja
Syntax (S)	nein	ja
Testauftrag (T)	ja	nein
Testauftrag (H)	ja	nein
Information (I)	nein	ja

Table 7 Meldungsklassen

Ausgabe auf Terminal nur für Instruktionen (jedoch nicht für Operationen).

Die Ausgabe der Instruktions in der Tracebuffer wird gesteuert durch das Standard DEF-Symbol &WRITE_IN (siehe DEF-Kommando).

Die Ausgabe der Operations in der Tracebuffer wird gesteuert durch einige Standard DEF-Symbole (siehe DEF-Kommando).

6.2 Aufbau der DEBUG-Meldungen

Eine DEBUG Meldung besteht aus einem festen und einem variablen Teil. Der variable Teil enfällt bei einer Reihe von Meldungen. Der feste Teil der Meldung entfällt nur bei D und SET mit dem Auftragskennzeichen #FFFF und bei WRITE. Fester und variabler Teil beginnen jeweils in einer neuen Zeile.

6.3 Aufbau fester Teil der Meldung

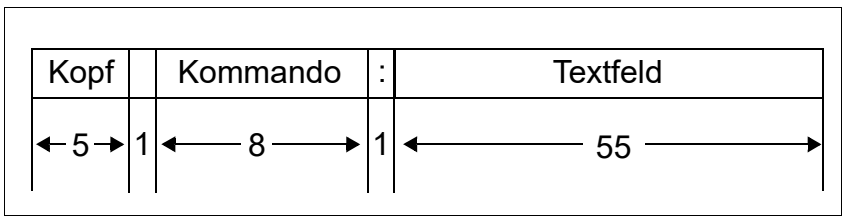

Figure 4 Fester Teil der Meldung

Aufbau des Kopfes

Der Kopf kann als Kurzform der Meldung betrachtet werden.

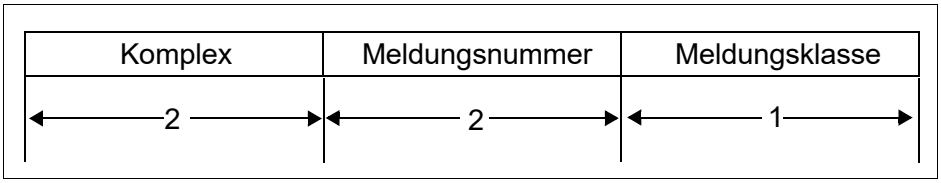

Figure 5 Kopf der Meldung

Erläuterung:

Komplex

Der Komplex dient zur Lokalisierung des DEBUG Komplexes welcher die Meldung veranlasst und zur Erweiterung des Meldungsnummernspektrums.

Mögliche Werte sind:

00 syntax analyzer

01 syntax analyzer

03 command analyzer

04 special commands

05 interpreter

06 execution routines

08 system object inspection

Meldungsnummer

Die Meldungsnummer ist eine pro Komplex vergebene eindeutige Nummer zwischen 1 und 99.

Meldungsklassen

Es werden folgende Meldungsklassen unterschieden:

E Fehlermeldung (Software Error)

S Syntaxmeldung

T Testauftragmeldung

I Informationsmeldung

N Meldungstext nicht vorhanden

H Wie Klasse T , wenn Aufbereitung hexadezimal

Kommando

Bei einigen Meldungen wird das Kommandoschlüsselwort wiederholt.

Textfeld

Erläuternder Text der Meldung. Dieser wird immer mit ausgegeben. In dieser Bedienungsanleitung sind alle Meldungen mit dem Textfeld aufgelistet. Es werden keine zusätzlichen Erläuterungen gegeben.

Zusatzinformation (variabler Teil der Meldung)

7 Syntax-Fehler

7.1 Syntax Fehlerbehandlung bei Steuerkommandos

Folgende Prüfungen werden vorgenommen:

- Es wird geprüft, ob das Kommando existiert und zu diesem Zeitpunkt zugelassen ist.
- Außerdem wird geprüft, ob die Parameter existieren, ob alle Pflichtparameter angegebenen sind und ob die Parameterwerte im zulässigen Bereich liegen.

Im Fehlerfall wird eine Meldung ausgegeben. Es wird jedoch nicht angegeben bei welchem Parameter ein Fehler vorliegt.

7.2 Syntax Fehler bei der Eingabe von Anweisungen

Die Behandlung der Syntaxfehler geht von folgendem Prinzip aus: Um eine fehlerhafte Eingabe leicht korrigieren zu können, ist die Syntaxprüfung zeilenorientiert und anweisungsbezogen. Diese Methode verhindert, daß die gesamte Eingabe von zusammengesetzten Anweisungen wiederholt werden muß.

Aus Sicht des Benutzers stellt sich die Fehlerbehebung wie folgt dar:

Nach der Eingabe einer Zeile bekommt der Benutzer eine Mitteilung, ob diese Zeile korrekt war oder nicht. Bei richtiger Eingabe erscheint nur das Prompting z.B. '**>'. Wird ein Fehler gemeldet muß der Anwender nur die fehlerhafte Anweisung wiederholen. Stehen in derselben Zeile hinter der fehlerhaften Anweisung noch weitere Anweisungen, so müssen auch diese neu eingegeben werden.

Beispiel:

**>	LABEL:AT &MARKE DO	
*:>	D %COUNTER;IF %B = %C THEN D %X;FI;D	
*:>	1238:J'745(4); SET %B=1; OD;	
0062S	:ILLEGAL CHARACTER	
1238:		
*:>		

In der letzen Zeile wird dem Benutzer ein Fehler gemeldet. Er muß in diesem Beispiel folgendes korrigieren und wiederholen:

```
D 1238:H'745(4);SET %B=1;OD;
```

Hinweis: Die Anweisungen ’AT’ und ’IF’ sind Ausnahmen, da diese Anweisungen eine oder mehrere Anweisungen enthalten können. Außerdem enthalten die ’THEN’-Anweisung und die ’ELSE’-Anweisung selbst wieder Anweisungen. Zur Syntaxprüfung wird diese Schachtelung in eine lineare Anweisungsfolge übergeführt.

Zum Beispiel ist die AT-Anweisung:

```
'LIN: AT xy DO D z;D v;OD;'
```

zusammengesetzt aus vier Anweisungen:

```
'LIN: AT xy DO',
'D z;',
'D v;' UND
'OD;'.
```

Unter diesem Gesichtspunkt ist das Prinzip der Syntaxfehlerbehandlung auf jede einzelne Anweisung anwendbar.

7.3 Syntaxfehlerbehandlung in Runfiles

Die Anweisungen in der Runfile werden genauso geprüft wie bei der interactiven Eingabe. Die erste fehlerhafte Zeile beendet das Einlesen der Anweisungsdatei. Dem Benutzer wird gemeldet welcher Fehler und in welcher Zeile er aufgetreten ist.

Im Gegensatz zur Fehlerbehandlung bei der Eingabe über Terminal ist es nicht möglich, die Anweisungen sofort zu korrigieren. Fehlerhafte geschachtelte Anweisungen werden durch DEBUG selbst mit der nötigen Anzahl von ’FI’ und ’OD’ abgeschlossen. Die Anweisungen werden nicht auf Terminal protokolliert.

Falls bei Fehlern in Runfiles nicht abgebrochen werden soll, so kann beim RUN-Kommando der Parameter FUNC=CHECK angegeben werden. Dieser Parameter darf nur angewandt werden, wenn sichergestellt ist, daß durch den Fehler keine ungewollten Ablaufe entstehen. Falls z.B. in der Testpunktdefinition ein Fehler ist, so werden die Anweisungen für diesen Testpunkt nicht mehr als Operations interpretiert, sondern als Instruktionen angesehen und sofort ausgeführt.

8 Syntax in BNF

Sei	`@`	das leere Symbol,
	`A`	ein Nicht-Terminal-Symbol,
	`X,Y,Z`	Kombinationen von Terminal-,Nicht-Terminal-Symbolen und einigen anderen Zeichen (, die im folgenden erklärt werden).

Die folgende Aufzählung erklärt die Bedeutung der verwendeten Symbole, welche überwiegend der BNF-Notation (Bacchus-Naur-Form) entsprechen:

`A ::= X`	weist dem Nicht-Terminal-Symbol A eine Zeichenfolge zu.
`"a"`	a ist ein Terminal-Symbol.
`X | Y`	erlaubt die Wahl zwischen X und Y. A ::= X | Y ist die Kurzform für A ::= X oder A ::= Y .
`()`	faßt eine Menge von Symbolen zusammen oder klärt die Zugehörigkeit bei möglichen Mißverständnissen.
`(X | Y) Z`	ist die Abkürzung für X Z | Y Z .
`[X]`	ist definiert als @ | X .
`{X}`	ist definiert als @ | X | X X | X X X | ...

und

`--comment`	Hier stehen wichtige Hinweise.

Nun zur Syntax:

-- zuerst die Kommando- (oberste) Ebene:

```
AMO_debug_cmd  ::=  """ debug_cmd """ [;]
debug_cmd      ::=  (  |  act_cmd       --  instruction/operation
                       |  at_cmd        --  instruction
                       |  break_cmd     --  instruction/operation
                       |  cont_cmd      --  instruction
                       |  dcl_cmd       --  instruction
                       |  deact_cmd     --  instruction/operation
                       |  def_cmd       --  instruction
                       |  dend_cmd      --  special command
                       |  display_cmd   --  instruction/operation
                       |  dowhile_cmd   --  instruction/operation
                       |  end_cmd       --  instruction
                       |  exec_cmd      --  instruction/operation
                       |  file_cmd      --  special command
```

```
                | go_cmd        --  instruction
                | gofor_cmd     --  instruction
                | gotil_cmd     --  instruction
                | if_cmd        --  instruction/operation
                | in_cmd        --  special command
                | insp_cmd      --  instruction/operation
                | list_cmd      --  special command
                | logoff_cmd    --  instruction/operation
                | logon_cmd     --  instruction/operation
                | mcall_cmd     --  instruction/operation
                | mdef_cmd      --  instruction
                | print_cmd     --  special command
                | redcl_cmd     --  instruction
                | redef_cmd     --  instruction
                | remat_cmd     --  instruction
                | remmac_cmd    --  instruction
                | remove_cmd    --  instruction
                | reset_cmd     --  special command
                | run_cmd       --  special command
                | runoff_cmd    --  instruction
                | runon_cmd     --  instruction
                | saveoff_cmd   --  special command/instruction
                | saveon_cmd    --  special command/instruction
                | sel_cmd       --  special command
                | set_cmd       --  instruction/operation
                | start_cmd     --  special command
                | stop_cmd      --  /operation
                | term_cmd      --  special command
                | trace_cmd     --  instruction/operation
                | triggoff_cmd  --  special command
                | triggon_cmd   --  special command
                | view_cmd      --  instruction/operation
                | write_cmd     --  instruction/operation
        )       (";" | "!")
    | (";" | "!")
        --  max. 1 special command und
        --  max. 160 Zeichen pro Zeile,
        --  bei special commands sind ";" und "!" optiona
        --  saveoff_cmd und saveon_cmd sind sowohl als
```

```
                    -- special command als auch als instruction/operation
                    -- erlaubt.
act_cmd       ::=   "ACT" ("ALL" | at_label {"," at_label})
at_cmd        ::=   at_label ":"          "AT" trigger
                                          "DO" {debug_cmd}   -- operation
                                          "OD"
break_cmd     ::=   "BREAK"
cont_cmd      ::=   "CONT" ("ALL" | at_label {"," at_label})
                    -- max. 6 Labels
dcl_cmd       ::=   "DCL" (debug_var | "(" debug_var {"," debug_var} ")")
                          debug_mode
                    {","  (debug_var | "(" debug_var {"," debug_var} ")")
                          debug_mode
deact_cmd     ::=   "DEACT" ("ALL" | at_label {"," at_label})
def_cmd       ::=   "DEF"
                     def_symbol
                    (      num_addr_abs length
                    |      "=" def_symbol [("+" | "-") offset] [length]
                    )
                    {      num_addr_abs length
                    |      "=" def_symbol [("+" | "-") offset] [length]
                    }
dend_cmd      ::=   "DEND"
display_cmd   ::=   ("D" | "DISPLAY" | "TRACE")
                    display_def {"," display_def}
display_def   ::=   (  display_item
                    |  "(" display_item {"," display_item} ")"
                    )  [disp_pict] ["#" number]
display_item  ::=     num_addr_rng
                    |  symb_addr_rng
                    |  "%STACK" length
                    |  "#R"
                    |  debug_var
                    |  io_port
dowhile_cmd   ::=   "DOWHILE"       condition ";"
                                    (debug_cmd)       -- operation
                    "OD"            {debug_cmd}       -- operation
end_cmd       ::=   "END"
exec_cmd      ::=   "EXEC" name
```

```
file_cmd        ::=   "FILE" ":"
                      (    "CURFILE" "=" filename ["," SUCFILE" "=" filename]
                      |    "SUCFILE" "=" filename
                      |    "CUR" "=" "CLOSE"
                           [["SUC" "=" "CLOSE"] | "," "SUCFILE" "=" filename]
                      |  "SUC" "=" "CLOSE" ["," CURFILE" "=" filename]
                      )
go_cmd          ::=   "GO"
gofor_cmd       ::=   "GOFOR"
gotil_cmd       ::=   "GOTIL" trigger {"," trigger}
if_cmd          ::=   "IF" condition
                        "THEN" (debug_cmd) {debug_cmd}      -- operations
                       ["ELSE" (debug_cmd) {debug_cmd}]     -- operations
                      "FI"
in_cmd          ::=   "IN" [":" "RUNFILE" "=" filename]
insp_cmd        ::=   "INSP" ("T*" | token_id)
list_cmd        ::=   "LIST" ":" "TAB" "="          (  "AT"
                                                     | "DCL"
                                                     | "DEF"
                                                     | "DEFN"
                                                     | "DEFS"
                                                     | "MAC"
                                                     | "STOP"
                                                     )
logoff_cmd      ::=   "LOGOFF"
logon_cmd       ::=   "LOGON"
mcall_cmd       ::=   "MCALL" macname
mdef_cmd        ::=   "MDEF" macname ";" {debug_cmd}        -- operation
                      "MEND"
print_cmd       ::=   "PRINT" [":" print_param]
redcl_cmd       ::=   "REDCL" debug_var debug_mode
redef_cmd       ::=   "REDEF"               -- DEF-Symbol bereits definiert
                      def_symbol
                      (  num_addr_abs [length]
                      |  "=" def_symbol [("+" | "-") offset] [length]
                      )
remat_cmd       ::=   "REMAT" ("ALL" | at_label {"," at_label})
remmac_cmd      ::=   "REMMAC" macname
remove_cmd      ::=   "REMOVE" trigger {"," trigger}
```

```
reset_cmd       ::=   "RESET"
run_cmd         ::=   "RUN" ":" "INFILE" "=" filename
                      ["," "PROT" "=" "NO"]
                      ["," "FUNC" "=" "CHECK"]
                      -- ["," "FUNC" "=" "CHECK"] entfällt  bei DASIST
runoff_cmd      ::=   "RUNOFF"
runon_cmd       ::=   "RUNON"
saveoff_cmd     ::=   "SAVEOFF"
saveon_cmd      ::=   "SAVEON"
sel_cmd         ::=   "SEL" [":" print_param]
set_cmd         ::=   "SET"              (     debug_var
                                         |     num_addr_rng
                                         |     io_port
                                         |     symb_addr_rng
                                         )
                      "="   (            arith_expr
                            |            bool_expr          -- nicht mit
                                                              io_port
                            |            bool_const         -- nicht mit
                                                              io_port
                            |            "'" char "'"       -- nicht mit
                                                              io_port
                            |            hex_string
                            |            num_addr
                            |            symb_addr
                            |            token
                            |            local_proc
                            )
                      ["#" number]
hex_string      ::=   "X'" hex_digit hex_digit {["_"] hex_digit hex_digit}
start_cmd       ::=   "START"
stop_cmd        ::=   "STOP"
term_cmd        ::=   "TERM"
trace_cmd       ::=   display_cmd
triggoff_cmd    ::=   "TRIGGOFF"
triggon_cmd     ::=   "TRIGGON"
view_cmd        ::=   "VIEW" view_object
view_object     ::=   (     "JOB" | "TASK" | "MAILBOX" | "SEMAPHOR"
                      |     "QUEUE" | "SEGMENT" | "REGION" | "POOL"
```

```
                    )     [token_id]
                  |  "MODCATA" [token_id | catname | "T*"]
                   |  "SYSCATA" [token_id | catname]
                   |  "MODCATI" [token_id | index | "T*"]
                   |  "SYSCATI" [token_id | index]
catname        ::=     string                -- max. 7 Zeichen
index          ::=     number
write_cmd      ::=     string                -- 1 bis 80 Zeichen
```

-- die nächste Ebene enthält Symbole, die öfter als einmal auftreten:

```
at_label       ::=     name       -- max. 8 alphanumerische Zeichen
condition      ::= general_cond | pointer_cond
general_cond   ::= bool_expr {("AND" | "OR" | "XOR") bool_expr}
bool_expr      ::= ["NOT"]
                   (   factor rel factor
                   |   bool_var
                   |   condition
                   |   "[" bool_expr "]"
                   )
factor         ::=     arith_expr | num_addr_rng | symb_addr
io_port        ::=     "#P" hex_number
pointer_cond   ::=     pointer_rel {("AND" | "OR" | "XOR") pointer_rel}
pointer_rel    ::=     pointer_var rel          (        pointer_var
                                                |        symb_addr_rng
                                                |        num_addr_rng
                                                |        "#" num_addr_abs
pointer_var    ::=     "%" name
print_param    ::= [   ["INFILE" "=" filename] ["," "INF" "=" "CUE"]
                   |   "REC" "=" dec_digit {dec_digit}
                       ["," "TEC" "=" dec_digit {dec_digit}]
                   ]
                       ["," "LABEL" "=" label] ["," "TLOG" "=" "ON"]
                       ["," "TIMEB" "=" time] ["," "TIMEE" "=" time]
                       ["," "DATEB" "=" date] ["," "DATEE" "=" date]
                       ["," "PERIOD" "=" "DAILY"]
                       -- "," wird weggelassen, wenn der zugehörige
                       -- Parameter der erste ist
label          ::=     "*" | letter {letter | dec_digit} ["*"]
                       -- max. 8 Zeichen
```

```
time            ::=   hour ":" minute ":" second
hour            ::=   00 | 01 | ... | 23
minute          ::=   second
second          ::=   00 | 01 | ... | 59
date            ::=   day "/" month "/" year
day             ::=   00 | 01 | ... | 31
month           ::=   01 | 02 | ... | 12
year            ::=   00 | 01 | ... | 99
token_id        ::=             "J*"
                    |           def_symbol
                    |           num_addr
                    |           num_addr_proc
                    |           token
token           ::=   "#" seg_addr [":" offset]
                ::=   (def_symbol | num_addr_abs) [("+" | "-") offset]
```

-- nun die Hilfsebene (diese Symbole kommen in den vorherigen Ebenen vor):

```
num_addr        ::=  num_addr_abs [addr_mode]
num_addr_rng    ::=    num_addr_abs ("TO" num_addr_abs | [addr_mode] length)
                     | num_addr_proc length
num_addr_abs    ::=  seg_addr ":" offset
num_addr_proc   ::=  local_proc ":" ["-"] offset [addr_mode]
string          ::=  "'" char { char} "'"
                     -- ' im string ist zu verdoppeln
symb_addr       ::=  def_symbol [addr_mode]
symb_addr_rng   ::=  def_symbol ["TO" def_symbol | [addr_mode] length]
```

-- zum Schluss das Fundament (Grundsymbole, die häufig benutzt werden):

```
addr_mode   ::=   ("->" [("+" | "-") arith_expr] | ("+" | "-") arith_expr)
                  {"->" [("+" | "-") arith_expr]}
arith_expr  ::=   operand {("+" | "-" | "*" | "/") operand}
operand     ::=     number
                  | debug_var
                  | arith_expr
                  | "[" arith_expr "]"
```

-- Namen ...

```
bool_var    ::=  name
debug_var   ::=  "%" name
def_symbol  ::=  "&" name
```

```
filename    ::=  name        -- das erste Zeichen muß  ein Buchstabe sein
                             -- max. 27 alphanumerische Zeichen
macname     ::=  name        -- max. 8 alphanumerische Zeichen
name        ::=  letter {letter | dec_digit | "_" | "."}

-- Zahlen ...
length          ::=  "(" (number | register) ")"
                     -- in Bedingungen sind nur 1, 2 oder 4 Bytes erlaubt
number          ::=  bin_number | dec_number | hex_number
bin_number      ::=  "B'" ("0" | "1") {["_"] ("0" | "1")}
dec_number      ::=  "T'" dec_digit {["_"] dec_digit}
hex_number      ::=  (["H'"] hex_digit | dec_digit) {["_"] hex_digit}
hex_digit       ::=  dec_digit | "A" | "B" | "C" | "D" | "E" | "F"
offset          ::=  number | register
seg_addr        ::=  number | register

-- druckbare Zeichen ...
bool_const  ::=  "TRUE" | "FALSE"
char        ::=      letter | dec_digit
                |    "," | "." | "/" | "<" | ">" | "?" | "+" | ":" | "*"
                |    "]" | "}" | "@" | "[" | "{" | "#" | "|" | "%" | " "
                |    "&" | "'" | "(" | ")" | "_" | "=" | "-" | " " | "—"
                |    "|" | "\" | "!" | ";"
debug_mode  ::=  "%" ("CHAR"| "INT" | "BOOL" | "POI" | "BYT")
dec_digit   ::=      "0" | "1" | "2" | "3" | "4" | "5"
                |    "6" | "7" | "8" | "9"
disp_pict   ::=  "%" ("CHAR"| "BIN" | "HEX" | "DMP" | "SYMB" | "ASM")
                     -- "SYMB" and "ASM" only in DASIST
letter      ::=      "A" | ... | "Z" | "a" | ... | "z"
local_proc  ::=  "P*" | "PB" | "PC" | "PD" | ... | "PZ"
register    ::=      "CS" | "IP" | "AX" | "BX" | "CX" | "DX" | "ES"
                |    "SS" | "SP" | "BP" | "DI" | "SI" | "DS" | "FL"
rel         ::=  "EQ" | "NE" | "GT" | "GE" | "LE" | "LT"
                 "="  | "/=" | ">"  | ">=" | "<=" | "<"
```

9 Benutzerhinweise

9.1 Allgemeines

Überall, wo mindestens ein Blank stehen kann, können beliebig viele Blanks und/oder Kommentare geschrieben werden. Ein Kommentar kann über beliebig viele Zeilen gehen und wird durch '/*....

9.2 Generierungsgrößen

Im folgenden sind die maximalen Werte angegeben, die bei der Verwendung von DEBUG bei der Eingabe von Kommandos eine Rolle spielen. Die Größen können nur durch eine Neuübersetzung geändert werden.

Bedeutung	Wert
Anzahl der Testpunkte	170
Anzahl der Macros	34
Anzahl der Standard DEF-Symbole	12
Anzahl der definierbaren DEF-Symbole	244
Anzahl der DEBUG Variablen	100
Anzahl der Operationen (gesamt)	750
Anzahl Expressions bei Operationen	510
Anzahl klammerbarer Instruktionen	34
Anzahl Expressions bei Instruktionen	68
Länge des Namens der DEBUG Variable	26
Länge des Namens der DEF-Symbole	26
Länge des Namens von Testpunkten	8
Länge des Namens von Makros	8
Länge von Dateinamen	32
Größe des Tracebuffers (Byte)	17000
Größe der Protokolldatei	10000
Charakteranzahl bei Instruktionen	260
Charakteranzahl bei Operationen	1000
Hexa Zeichen bei Instruktionen	400
Hexa Zeichen bei Operationen	1700

Table 8 Generierungsgrößen

10 Liste der DEBUG-Meldungen

Die hier aufgeführten Meldungen stellen alle Meldungen an den Benutzer dar. Sie sind nach Komplexen und Meldungsnummern geordnet.
Die Zusatzinformationen sind nach Meldungsnummern geordnet im nächsten Abschnitt beschrieben.
Alle Meldungen mit den Nummern 00xxS und 01xxS haben als Zusatzinformation die letzte Eingabezeile bis zum fehlerhaften Zeichenstring.
Es wird zuerst der Text in deutsch angegeben, darunter die englische Version. Ausgegeben werden die Meldungen jeweils nur in einer Sprache.

/***** COMPLEX : SYNTAX ANALYZER (00/01) *****/

0001S	:FALSCHER BEGINN EINER ANWEISUNG :ILLEGAL BEGINNING OF INSTRUCTION
0002S	:"AT" FEHLT :"AT" MISSING
0003S	:":" FEHLT :":" MISSING
0004S	:TESTPUNKTNAME WURDE ERWARTET :TESTORDERLABEL EXPECTED
0005S	:DEF-SYMBOL FALSCH ODER FEHLT :MISSING OR ILLEGAL DEF-SYMBOL
0006S	:ENDADRESSE NICHT WIE ANFANGSADRESSE NUMERISCH :ENDADDRESS NOT NUMERICAL LIKE STARTADDRESS
0007S	:SELEKTOR:OFFSET MUSS INTEGER SEIN :SELECTOR:OFFSET MUST BE INTEGER
0008S	:ENDADRESSE KLEINER ALS ANFANGSADRESSE :ENDADDRESS LOWER THAN STARTADDRESS
0009S	:DEF-SYMBOL TABELLE VOLL :NO MORE DEF-SYMBOL ACCEPTED
0010S	:DEF-SYMBOL IST SCHON DEFINIERT

	:DEF-SYMBOL ALREADY DEFINED
0011S	:WARNUNG: ES WERDEN NUR 6 VARIABLE BERUECKSICHTIGT :WARNING: ONLY 6 VARIABLES ACCEPTED
0012S	:ADRESSE UNGUELTIG :ADDRESS INVALID
0013S	:ADRESSMODIFIKATION NICHT ERLAUBT :ADDRESS MODIFICATION NOT ALLOWED
0014S	:"," ODER ";" oder "!" WURDE ERWARTET : "," OR ";" OR "!" EXPECTED
0015S	:LAENGE FEHLT ODER UNZULAESSIGE MODIFIKATION :LENGTH MISSING OR WRONG MODIFICATION
0016S	:NUR INNERHALB AT BZW MDEF ZUGELASSEN :ONLY ADMITTED WITHIN AT- OR MDEF-INSTRUCTIONS
0017S	:RECHTE KLAMMER FEHLT :RIGHT PARENTHESIS MISSING
0018S	:"FI" WURDE ERWARTET : "FI" EXPECTED
0019I	:KEINE OPERATION MEHR MOEGLICH, AT BZW MDEF AUSGEFUEHRT :NO MORE OPERATION POSSIBLE, AT OR MDEF ACCEPTED
0020S	:SELEKTOR HAT UNGUELTIGEN WERT :SELECTOR INVALID
0021S	:DCL IGNORIERT, MAXIMUM AN VARIABLEN ERREICHT :NO MORE DEBUG VARIABLES POSSIBLE
0022S	:TESTPUNKT AKTIV ODER BELEGT, REMAT/REMOVE IGNORIERT :TESTORDER ACTIVE OR IN EXECUTION, REMAT/REMOVE REJECTED
0023S	:DEBUG VARIABLE FALSCH ODER FEHLT

:MISSING OR ILLEGAL DEBUG VARIABLE

0024S :DEBUG VARIABLE IST SCHON DEFINIERT
:DEBUG VARIABLE ALREADY DECLARED

0025S :"," FEHLT
:"," MISSING

0026S :ADRESSMODIFIKATION KEIN ARITHMETISCHER AUSDRUCK
:ADDRESSMODIFICATION MUST BE AN ARITHMETIC EXPRESSION

0027S :ZU VIELE OPERANDEN
:TOO MANY OPERANDS

0028S :UNVERTRAEGLICHE OPERANDEN
:INCOMPATIBLEOPERANDS

0029S :DEBUG VARIABLE IST NICHT DEFINIERT
:DEBUG VARIABLE NOT DECLARED

0030S :ALS LAENGE HIER NUR 1,2 ODER 4 ERLAUBT
:ONLY LENGTH OF 1,2 OR 4 BYTES ALLOWED

0031S :BITTE ";" ODER "!" ODER SETAUFTRAGSKENNZEICHEN EINGEBEN
:PLEASE ENTER ";" OR "!" OR THE SET-ORDERNUMBER

0032S :OPERATION FALSCH ODER FEHLT
:ILLEGAL OR MISSING OPERATION

0033S :TESTPUNKTNAME EXISITIERT BEREITS
:TESTORDER ALREADY EXISTS

0034S :"DO" FEHLT ODER UNZULAESSIGE MODIFIKATION
:"DO" MISSING OR ILLEGAL MODIFICATION

0035S :TESTPUNKTNAME FALSCH ODER FEHLT
:WRONG OR MISSING LABEL

0036S :TESTPUNKTNAME EXISTIERT NICHT

:LABEL DOES NOT EXIST

0037S :REMAT NUR NACH "!" ZUGELASSEN
:REMAT ONLY ALLOWED IF ALL INSTRUCTIONS EXECUTED

0038S :UNZULAESSIGER LAENGENWERT
:ILLEGAL LENGTH VALUE

0039S :DEF-SYMBOL IST NICHT DEFINIERT
:DEF-SYMBOL NOT DEFINED

0040S :ANGABE %HEX,%BIN,%CHAR FALSCH ODER UNZULAESSIG
:DISPLAY PICTURE %HEX,%BIN,%CHAR WRONG OR ILLEGAL

0041S :WARNUNG: NUR 12 DISPLAYS ANGENOMMEN
:WARNING : ONLY 12 DISPLAYS ACCEPTED

0042S :OBJEKT NAME ZU LANG
:OBJECT NAME TOO LONG

0043S :ZU VIELE OPERATOREN
:TOO MANY OPERATORS

0044S :"THEN" FEHLT
:"THEN" MISSING

0045S :ZU GROSSE SCHACHTELUNGTIEFE
:MAXIMAL NESTING EXCEEDED

0046S :OPERATION WIRD ERWARTET
:OPERATION KEYWORD EXPECTED

0047S :DEBUG MODE FALSCH ODER FEHLT
:MISSING OR ILLEGAL DEBUG MODE

0048S :OPERANDEN HABEN UNZULAESSIGEN MODE
:ILLEGAL MODE OF OPERANDS

0049S :NAME DER ANSCHLUSSROUTINE FALSCH ODER FEHLT
:WRONG OR MISSING NAME OF ADDITIONAL ROUTINE

0050S :ZUVIELE OPERATIONEN, AT BZW MDEF WIRD IGNORIERT
:NO MORE OPERATION POSSIBLE, AT OR MDEF IGNORED

0051S :SCHLUESSELWORT FALSCH
:ILLEGAL INSTRUCTION KEYWORD

0052S :ZUVIELE NICHT AUSGEFUEHRTE ANWEISUNGEN
:NO MORE INSTRUCTION POSSIBLE, PLEASE ENTER "!"

0053S :TESTPUNKTTABELLE VOLL
:NO MORE TESTORDER DEFINITION POSSIBLE

0054S :OPERANDEN FEHLEN
:OPERANDS MISSING

0055S :EXPRESSIONTABELLE VOLL
:NO MORE EXPRESSION POSSIBLE

0056I :INSTRUKTIONEN WERDEN AUSGEFUEHRT, SPEICHER IST VOLL
:INSTRUCTIONS WILL BE EXECUTED, NO MORE ROOM AVAILABLE

0057S :TESTPUNKT AN DATENADRESSE NICHT ERLAUBT
:TRIGGER ON DATA ADDRESS NOT ALLOWED

0058S :LAENGENANGABE NICHT NUMERISCH
:LENGTH NOT NUMERICAL

0060S :UNVERTRAEGLICHE ADRESSANGABE
:IN COMPATIBLE ADDRESSES

0061I :RUNON IGNORIERT: ES IST KEINE DATEI ZUGEWIESEN
:RUNON IGNORED: NO FILE ASSIGNED

0062S :UNZULAESSIGES ZEICHEN
:UNEXPECTED CHARACTER

0063S :ANWEISUNG IM BREAKMODUS NICHT ERLAUBT
:INSTRUCTION NOT ALLOWED IN BREAKMODUS

0064S :ZEICHENFOLGE ZU LANG

:STRING TOO LONG

0065S :TESTPUNKTNAME ZU LANG
:LABEL NAME TOO LONG

0066E :TESTPUNKTTABELLE ZERSTOERT
:TESTORDER TABLE DESTROYED

0067S :WERT ZU GROSS
:NUMBER TOO LARGE

0068S :SCHLUESSELWORT ZU LANG
:IDENTIFIER TOO LONG

0069S :TESTPUNKTADRESSE FEHLT
:MISSING OR ILLEGAL TRIGGER

0070S :MINDESTENS EIN ZEICHEN NOTWENDIG
:AT LEAST ONE CHARACTER NECESSARY

0071S :FEHLER IN EXPRESSION
:ERROR IN EXPRESSION

0072S :SET-OPERAND FEHLT ODER FALSCH
:SET-OPERANDS MISSING OR WRONG

0073S :"=" FEHLT
:"=" MISSING

0074S :TRACEABLE ITEM FALSCH ODER FEHLT
:TRACEABLE ITEM MISSING OR ILLEGAL

0075S :KEINE STRINGEINGABEN MEHR MOEGLICH
:NO MORE CHARACTER OR HEXADECIMAL STRING POSSIBLE

0076S :ADRESSBEREICHSANGABE BEI LOKALEN DATEN NICHT ERLAUBT
:ADDRESS RANGE NOT ALLOWED FOR LOCAL ADDRESSES

0077S :MODIFIKATION NUR DURCH DIREKTE WERTANGABE
:MODIFICATION ONLY BY VALUE POSSIBLE

0078S :CHARAKTERSTRING FEHLT
:CHARACTER STRING MISSING

0079S :INDEX ZU GROSS
:INDEX TOO LARGE

0080S :INDEX NICHT ERLAUBT
:INDEX NOT ALLOWED

0081S :EIGENE TASK NICHT ERLAUBT
:OWN TASK NOT ALLOWED

0082S :OBJECT ODER KATALOG FEHLT ODER FALSCH
:OBJECT OR CATALOG MISSING OR WRONG

0083S :TOKEN MUSS INTEGER SEIN
:TOKEN MUST BE INTEGER

0084S :OBJECT NAME NUR FUER NAMENKATALOG
:OBJECT NAME ONLY FOR NAME CATALOG

0085S :OBJECT TOKEN FEHLT ODER FALSCH
:OBJECT TOKEN MISSING OR WRONG

0086S :PROCEDUREBENE "A" NICHT ERLAUBT
:PROCEDURE LEVEL "A" NOT ALLOWED

0087S :FALSCHE LAENGENWERTE
:INCOMPATIBLE LENGTH VALUES

0088S :TRIGGERADRESSEN DUERFEN NICHT DEREFERENZIERT WERDEN
:DEREFERENCING OF TRIGGERS NOT ALLOWED

0089S :DEBUG VARIABLE ODER DEBUG MODE FALSCH
:WRONG DEBUG VARIABLE OR WRONG DEBUG MODE

0090E :MAKROTABELLE ZERSTOERT
:MACRO TABLE DESTROYED

0091S	:MAKRONAME ZU LANG :MACRO NAME IS TOO LONG
0092S	:MAKRONAME EXISTIERT BEREITS :MACRO NAME ALREADY EXISTS
0093S	:MAKRONAME EXISTIERT NICHT :MACRO NAME NOT FOUND
0094S	:KEINE WEITEREN MAKRODEFINITIONEN MOEGLICH :NO MORE MACRO DEFINITION POSSIBLE
0095S	:NUR ZUM BEENDEN EINER MAKRODEFINITION ERLAUBT :ONLY ADMITTED TO CLOSE A MACRO DEFINITON
0096S	:MAKRONAME FALSCH ODER FEHLT :MACRO NAME MISSING OR ILLEGAL
0097S	:REGISTERNAMEN NUR ERLAUBT WENN BREAKPUNKT ERREICHT IST :REGISTERNAME ONLY ALLOWED IF BREAKPOINT REACHED
0098S	:ES SIND ZU VIELE ADRESSEN ANGEGEBEN :TOO MANY ADDRESSES
0099I	:MAKRO BEREITS GELOESCHT :MACRO ALREADY REMOVED
0101S	:DEREFERENZIEREN NICHT ERLAUBT :DEREFERENCING NOT ALLOWED
0102S	:NUR ZUM BEENDEN VON AT BZW DOWHILE :ONLY ADMITTED TO CLOSE AN AT OR DOWHILE STATEMENT
0103S	:ADRESSE ODER "=" FALSCH ODER FEHLT :ADDRESS OR "=" MISSING OR WRONG
0104I	:TESTPUNKT BEREITS GELOESCHT :TESTORDER ALREADY REMOVED
0105S	:APOSTROPH FEHLT

:APOSTROPHE MISSING

0106S :NEGATIVER OFFSET
:NEGATIVE OFFSET

0107S :FALSCHE EINGABE BEI POINTER VARIABLEN
:WRONG INPUT FOR POINTER VARIABLES

0108S :ZUM LOESCHEN VON TESTPUNKTEN BITTE REMAT BENUTZEN
:PLEASE USE REMAT TO DELETE TESTPOINTS

0109S :ZUM LOESCHEN VON BREAKPUNKTEN BITTE REMOVE BENUTZEN
:PLEASE USE REMOVE TO DELETE BREAKPOINTS

0110S :REMAT IGNORIERT FUER AKTIVE UND LAUFENDE TESTPUNKTE
:REMAT IGNORED FOR ACTIVE OR RUNNING TESTPOINTS

/***** COMPLEX : COMMAND ANALYZER (03) *****/

0301S :UNGUELTIGES ZEICHEN IM STEUERKOMMANDO
:SPECIAL COMMAND : UNEXPECTED SIGN

0302S :KOMMANDONAME ZU LANG ODER ":" FEHLT
:COMMAND NAME TOO LONG OR ':' MISSING

0303S :KOMMANDO NICHT GEFUNDEN
:COMMAND NOT FOUND

0304I :DEBUG V4.2 KV..: GESTARTET UM :
:DEBUG V4.2 KV..: STARTS AT:

0305S :UNGUELTIGE ZEICHEN IM PARAMETERNAMEN
:UNEXPECTED CHARACTER IN PARAMETER NAME

0306S :PARAMETER ZU LANG ODER "=" FEHLT
:PARAMETER TOO LONG OR '=' MISSING

0307S :TRIGGER-MODUS EINGESCHALTET
:TRIGGER MODE ON

0308S :TRIGGER-MODUS AUSGESCHALTET
:TRIGGER MODE OFF

0309S :UNGUELTIGER PARAMETERWERT
:ILLEGAL PARAMETER VALUE

0310S :KOMMANDO ZUR ZEIT NICHT ERLAUBT
:COMMAND NOT ADMITTED AT THE MOMENT

0312S :PARAMETER FEHLT
:PARAMETER MISSING

0313S :UNGUELTIGER PARAMETERNAME
:ILLEGAL PARAMETER NAME

0314S :UNGUELTIGES ZEICHEN IM PARAMETERWERT
:UNEXPECTED SIGN IN PARAMETER VALUE

0315S :PARAMETERWERT ZU LANG ODER "," FEHLT
:PARAMETER VALUE TOO LONG OR "," MISSING

0316S :PARAMETERWERT FEHLT
:PARAMETER VALUE MISSING

0317S :TRIGGON OHNE VORHERIGES SEL (SEL WIRD GENERIERT)
:TRIGGON WITHOUT SEL BEFORE (STANDART SEL IS GENERATED)

0319S :UNGUELTIGE GROESSE DES PARAMETERWERTS
:PARAMETER VALUE EXCEEDS LIMITS

0321S :UNGUELTIGES ZEICHEN IM STEUERKOMMANDO
:UNEXPECTED SIGN IN THE SPECIAL COMMAND

0322S :UNGUELTIGES SCHLUESSELWORT
:ILLEGAL KEYWORD

0323S :NICHT ERLAUBTER PARAMETER
:FORBIDDEN PARAMETER

0325I :DEBUG BEENDET UM:
:DEBUG TERMINATED AT:

0326I :WARNUNG: NICHT ALLE STACKS FREI - DEBUG BEENDET UM:
:WARNING: OCCUPIED STACKS CANCELED,DEBUG TERMINATED AT:

0327I :TRACEBUFFER ZURUECKGESETZT
:TRACEBUFFER RESET

0328I :RESET ABGEWIESEN, DA NOCH PROTOKOLLIERT WIRD
:RESET REJECTED, PROTOCOLLING IS STIL ACTIVE

/***** COMPLEX : SPECIAL COMMANDS (04) *****/

0401I :ANWEISUNGSDATEI GEPRUEFT
:RUN FILE CHECKED

0402I :ERSTER FEHLER IN ZEILE:
:FIRST ERROR DETECTED IN LINE :

0404I :OEFFNEN DER ANWEISUNGSDATEI NICHT MOEGLICH
:OPEN OF RUNFILE NOT POSSIBLE

0405I :SCHLIESSEN DER ANWEISUNGSDATEI NICHT MOEGLICH
:CLOSE OF RUNFILE NOT POSSIBLE

0406I :SELEKTION FUER TRIGGER-MODUS GESETZT
:SELECTION FOR TRIGGER MODE SET

0407I :EINGABE IGNORIERT, AUSGABE WIRD FORTGESETZT
:INPUT IGNORED, OUTPUT CONTINUED

0408I :TRACEBUFFER IST LEER
:TRACE BUFFER IS EMPTY

0409E :TESTPUNKTENDE FEHLT, TESTPUNKT NICHT KOMPLETT
:TRIGGER END MISSING, TESTORDER INCOMPLETE

0410I :ALLE ANWEISUNGEN AUSGEFUEHRT

:ALL INSTRUCTIONS ARE EXECUTED

0412I :PRINT BZW LIST ABGEBROCHEN
:PRINT OR LIST CANCELLED

0413I :PRINT BEENDET
:PRINT TERMINATED

0414I :DMS FEHLER BEIM LESEN EINES SATZES
:DMS ERROR WHEN READING NEXT RECORD

0415I :ENDE DER PROTOKOLLDATEI ERREICHT
:END OF PROTOCOL FILE REACHED

0417I :SCHLIESSEN DER PROTOKOLLDATEI NICHT MOEGLICH
:CLOSE OF PROTOCOL FILE NOT POSSIBLE

0418I :OEFFNEN DER PROTOKOLLDATEI NICHT MOEGLICH
:OPEN OF PROTOCOL FILE NOT POSSIBLE

0419I :EINGABE ZURUECKGEWIESEN, DA DIALOG MIT AMO-DEBUG LAUEFT
:INPUT REJECTED,SESSION RUNNING ON AM-TERMINAL

0420I :CUR-DATEI BEREITS ZUGEWIESEN
:CURRENT FILE ALREADY ASSIGNED

0421I :OEFFNEN DER CUR-DATEI NICHT MOEGLICH
:OPEN OF CURFILE NOT POSSIBLE

0422I :SCHLIESSEN DER CUR-DATEI NICHT MOEGLICH
:CLOSE OF CURFILE NOT POSSIBLE

0423I :CUR-DATEI ZUGEWIESEN
:CURFILE ASSIGNED

0424I :SUC-DATEI BEREITS ZUGEWIESEN
:SUCFILE ALREADY ASSIGNED

0425I :OEFFNEN DER SUC-DATEI NICHT MOEGLICH
:OPEN OF SUCFILE NOT POSSIBLE

0426I	:SCHLIESSEN DER SUC-DATEI NICHT MOEGLICH :CLOSE OF SUCFILE NOT POSSIBLE
0427I	:SUC-DATEI ZUGEWIESEN :SUCFILE ASSIGNED
0428I	:EINGABE ZURUECKGEWIESEN, DA DIALOG MIT MTH LAEUFT :INPUT REJECTED, DIALOG WITH MTH AT THIS MOMENT
0429I	:TABELLE DER GESTOPPTEN TASKS : :TABLE OF STOPPED TASKS:
0431I	:SPEICHERGERAET NICHT VORHANDEN :STORAGE DEVICE NOT READY
0432I	:UNGUELTIGER DATEINAME :ILLEGAL FILENAME
0433I	:KEIN SPEICHER VORHANDEN ODER UNGUELTIGER DATEINAME :NO MEMORY AVAILABLE OR WRONG FILENAME
0434I	:TABELLE DER DEF-SYMBOLE: :TABLE OF DEF-SYMBOLS:
0436I TAB=....	:ENDE DER TABELLE :END OF TABLE
0437I TAB=DCL	:TABELLE DER DEBUG VARIABLEN: :TABLE OF DEBUGVARIABLES:
0440I TAB=AT	:TESTPUNKTTABELLE: :TESTORDER TABLE :
0444I TAB=....	:KEIN TABELLENEINTRAG :NO ENTRY IN TABLE
0445I	:LIST ABGEBROCHEN :LIST CANCELLED

0447I :CUR-DATEI NICHT ZUGEWIESEN
:CURFILE NOT ASSIGNED

0448I :CUR-DATEI GESCHLOSSEN
:CURFILE CLOSED

0449I :SUC-DATEI NICHT ZUGEWIESEN
:SUCFILE NOT ASSIGNED

0450I :SUC-DATEI GESCHLOSSEN
:SUCFILE CLOSED

0451I :KEIN GEFUNDENER EINTRAG
:NO CORRESPONDING ENTRY

0454I :FEHLER WAEHREND SCHREIBEN
:ERROR DURING WRITE

0457I :MAKROTABELLE:
TAB=MAC
:TABLE OF DEBUG MACROS:

0459I :ENDE DER MAKROTABELLE
:END OF MACRO TABLE

0460I :ANZAHL DER EINTRAEGE:
:NUMBER OF ENTRIES:

0461I :PARAMETER ODER DATEINAME NICHT GEFUNDEN
:PARAMETER OR FILENAME NOT FOUND

0462I :SCHREIBEN IN PROTOKOLLDATEI NICHT MOEGLICH
:WRITING INTO PROTOCOL FILE NOT POSSIBLE

0463I	TAB=AT	:NAME	ZAEHLER	ZUSTAND	ART	ADRESSE
		:NAME	COUNTER	STATUS	KIND	ADDRESS

0464I	TAB=MAC	:NAME	ZUSTAND	
		:NAME	STATUS	

0465I	TAB=DEF	:NAME		ADRESSE	LAENGE TYP	
		:NAME		ADDRESS	LENGTH TYP	
0466I	TAB=DCL	:NAME		MODE	WERT	ADRESSE
		:NAME		MODE	VALUE	ADDRESS
0467I	TAB=STOP	:NAME	STOPZEIT		ART	
		:NAME	STOPTIME		KIND	

/***** COMPLEX : INTERPRETER (05) *****/

0501I :KEINE GESTOPPTE TASK AN DIESEM TESTPUNKT
:NO TASK STOPPED AT THIS LABEL

0502I :ADRESSBEREICH ZU GROSS, DISPLAY WURDE GEKUERZT
:ADDRESSRANGE TOO LARGE, DISPLAY WAS SHORTENED

0503I :ANGEGEBENE LOGISCHE ADRESSE EXISTIERT NICHT
:DESIRED LOGICAL ADDRESS DOES NOT EXIST

0504I :DIVISION NICHT ERLAUBT
:DIVISION NOT ALLOWED

0505I :GO, GOTIL, BREAK :BREAKMODUS ZUR ZEIT NICHT ERLAUBT
:GO, GOTIL, BREAK: BREAKMODUS NOT ALLOWED AT THIS MOMENT

0506I :PROZESSOR IST BEREITS ANGEHALTEN, BREAK IGNORIERT
:PROCESSOR ALREADY STOPPED, BREAK IGNORED

0507I :BREAK IGNORIERT, NICHT IM BREAKMODUS ODER DIALOG MIT AMO
:BREAK IGNORED, NO BREAKMODUS OR DIALOG VIA AMO DEBUG

/***** COMPLEX : EXECUTION ROUTINES (06) *****/

0601T :TESTPUNKT AKTIVIERT
:TESTORDER ACTIVATED

0602T :TESTPUNKT BEREITS AKTIVIERT
:TESTORDER ALREADY ACTIVATED

0603T :AKTIVIERUNG NICHT DURCHFUEHRBAR
:ACTIVATION REJECTED

0604T :TESTPUNKT DEAKTIVIERT
D :TESTORDER DEACTIVATE

0605T :TESTPUNKT BEREITS DEAKTIVIERT
:TESTORDER ALREADY DEACTIVATED

0606T :DEAKTIVIEREN ABGEWIESEN, KONFLIKT MIT AKTIVIEREN
:DEAKTIVIVATION REJECTED, CONFLICT WITH ACTIVATION

0607T :DEAKTIVIEREN ABGEWIESEN
:DEACTIVATION REJECTED

0608T :ANSCHLUSSROUTINE WURDE AUFGERUFEN
:ADDITIONAL ROUTINE EXECUTED

0609E :IN DER OPERATIONTABELLE FEHLT ENDEKENNUNG
:OD MISSING IN OPERATION TABLE

0611T :REMAP-FEHLER
:REMAP-ERROR

0612T :AUSZUGEBENDER TRACEBUFFEREINTRAG WURDE UEBERSCHRIEBEN
:TRACEBUFFER ENTRY IS OVERWRITTEN

0613T :BEGINN DER TESTPUNKTBEHANDLUNG
:START OF TESTORDER EXECUTION

0614T :DISPLAY AUSGEFUEHRT
:DISPLAY EXECUTED

0616T :TESTPUNKT DEFINIERT
:AT INSTRUCTION EXECUTED

0617T :AT NICHT AUSGEFUEHRT

:TESTORDER NOT DEFINED

0618T :DATEI KONNTE NICHT GEOEFFNET WERDEN
:OPEN OF FILE NOT POSSIBLE

0619T :DATEI KONNTE NICHT GESCHLOSSEN WERDEN
:CLOSE OF FILE NOT POSSIBLE

0620T :BEIM SCHREIBEN TRAT EIN FEHLER AUF
:ERROR DURING PUT

0621T :PROTOKOLLIERUNG ABGEBROCHEN, DA DIE DATEI VOLL IST
:PROTOCOLLING STOPPED , PROTOCOL FILE FULL

0622T :DATENVERLUST, LETZTER SATZ EVENTUELL UNGUELTIG
:DATA LOST, LAST ENTRY PROBABLY DESTROYED

0623T :DATENVERLUST, PROBLEM MIT TRACEBUFFER
:DATA LOST ,PROBLEM WITH TRACEBUFFER

0624T :PROTOKOLLDATEI GEWECHSELT, ZUR ZEIT GUELTIG:
:PROTOCOL FILE CHANGED, AT THIS TIME VALID:

0625T :KEIN SPEICHER VORHANDEN
:NO MEMORY AVAILABLE

0626I :STOPTABELLE IST VOLL
:NO FREE ENTRY IN STOPTABLE

0627T :TASK ANGEHALTEN (TESTPUNKTNAME,STOPZEIT)
:TASK STOPPED (LABEL,STOPTIME)

0628T :TASK FORTGESETZT (TESTPUNKTNAME,STOPZEIT,STARTZEIT)
:TASK CONTINUED (LABEL,STOPTIME,CONTTIME)

0629T :INHALT DER REGISTER
:DISPLAY REGISTER

0630T :SET AUSGEFUEHRT
:SET EXECUTED

0632T :******** PROZESSOR ANGEHALTEN ********
:******** PROCESSOR STOPPED ********

0634T :AN DIESER ADRESSE IST SCHON EIN ANDERER TESTPUNKT AKTIV
:ALREADY AN ACTIVATED TRIGGER AT THIS ADDRESS

0635T :PATCH WURDE AUSGEFUEHRT,RETURNCODE 8800 ODER 8803
:PATCH EXECUTED WITH WARNINGS 8800 OR 8803

0636T :PATCH WURDE AUSGEFUEHRT
:PATCH EXECUTED

0637T :PATCH WURDE NICHT AUSGEFUEHRT
:PATCH NOT EXECUTED

0638T :MAKRO WURDE AUFGERUFEN
:MACRO EXECUTED

0639T :MAKRO GELOESCHT, AUFRUF WURDE IGNORIERT
:MACRO REMOVED, EXECUTION IGNORED

0640T :BREAK IGNORIERT,DA MTH/DEBUG JOB NOCH NICHT CREIERT IST
:BREAK IGNORED BECAUSE MTH/DEBUG JOB ARE NOT CREATED YET

0641T :DOWHILE NACH 255 DURCHLAEUFEN ABGEBROCHENN
:DOWHILE ABORTED AFTER THE MAXIMUM OF 255 LOOPINGS

0641T :ANWEISUNG ZUR ZEIT NICHT AUSFUEHRBAR
:BINSTRUCTION NOT EXECUTABLE AT THEN MOMENT

/***** COMPLEX : SYSTEM OBJECTS (08) *****/

0801I :START DER JOB-DATEN
:START OF JOB-DATA

0802I :START DER TASK-DATEN
:START OF TASK-DATA

0809I :START DER MODCATI-DATEN
:START OF MODCATI-DATA

0810I :START DER SYSCATI-DATEN
:START OF SYSCATI_DATA

0811I :START DER MODCATA-DATEN
:START OF MODCATA-DATA

0812I :START DER SYSCATA-DATEN
:START OF SYSCATA_DATA

0813I :ENDE DER DATEN
:END OF DATA

0814I :UNBEKANNTER OBJEKT-TYP IN OS-INSP-RUECKMELDUNG
:UNKNOWN OBJECT-TYPE DETECTED IN OS-INSP-RESULT

0815I :RMX-TOKEN IST NICHT ERLAUBT
:RMX-TOKEN NOT ALLOWED

0820I :JOB-DATEN
:JOB-DATA

0821I :TASK-DATEN
:TASK-DATA

0822E :MANGELNDE BETRIEBSMITTEL
:LACK OF RESOURCES

0823I :PARAMETER FEHLER
:PARAMETER ERROR

0824I :KEIN ENTSPRECHENDER EINTRAG GEFUNDEN
:NO CORRESPONDING ENTRY FOUND

0825I :MAILBOX-DATEN
:MAILBOX-DATA

0826I :POOL-DATEN
:POOL-DATA

0827I :SEMAPHORE-DATEN

:SEMAPHORE-DATA

0828I :BUFFER-DATEN
:BUFFER-DATA

0829I :REGION-DATEN
:REGION-DATA

0830I :QUEUE-DATEN
:QUEUE-DATA

Bilder

Tabellen

Index

E

F

V

W