A MITEL
PRODUCT
GUIDE

Unify OpenScape 4000/HiPath 4000

Debug

Debug

Service Documentation
06/2018

A31003-H3100-S106-06-7620

Mitel

Contents

1 General Information on DEBUG

The Debugger is a software tool available for the operating system via the AMO DEBUG. With the Debugger, instructions and commands can be entered that allow the system specialist to execute targeted diagnostic activities.

The Debugger allows both reading and writing, as well as setting TRACE markers in memory (data basis).
With the help of DEBUG run files, it is possible to automatically execute certain DEBUG commands at defined points.

The Trace AMOs are a subfunction of the Debugger (AMO TRACA and TRACS). Messages between the system's software complexes can be read into Trace memory using precise, bit-by-bit filter properties. Later, errors can be isolated with the documented message interchange.

Further possibilities for Trace are offered by the AMOs

- PETRA (peripheral tracer)
- DISPA and DISPS (memory readout).

IMPORTANT: The DEBUG, TRACA, TRACS, DISPA, DISPS and PETRA applications assume knowledge of the system software and CHILL source code. Faulty operation can lead to system failure and destruction of the system program in the database, as well as possible destruction of the hard disk.

2 User Interface

2.1 Calling the Program

Prior to calling the program, an ’intermediary’ must be activated; it takes care of the transfer of input and output data from the input devices to DEBUG (and vice versa).
Two ’intermediaries’ can be chosen from:

- the AMO DEBUG (AMO) (see Section 2.7, “Input via AMO DEBUG”)

and

- the Miniterminal Handler (MTH) (see Section 2.8, “Input via Miniterminal Handler (MTH)”).

The MTH serves the devices connected to the DP, whereas AMO DEBUG accesses the devices installed at the LCU. This chapter contains no details as to which terminal devices may be connected. The entries described herein relate to primitive terminals that forward inputs and outputs in transparent mode.
The advantages afforded by intelligent terminals (PC, APOLLO) cannot be dealt with in detail (see for example DASIST product on APOLLO).

These are, for example:

- Menu-driven input,
- Batch processing,
- CHILL symbols,
- Disassembler,
- Retranslation, etc.

Once the ’intermediary’ has been activated, a dialog session can be started with DEBUG. The dialog interface is implemented in such a way that DEBUG can be operated simultaneously via MTH and AMO. As DEBUG and MTH are present in any processor, the MTH interface can connect a separate device in every processor. This enables tests to be performed simultaneously in several processors. The AMO interface (LCU) permits only one processor to be accessed at a time. However, a switchover to another processor can be initiated at any time.
When the dialog with DEBUG is to be terminated, the ’intermediary’ can be deactivated with the ’DEND’ command. If several LCUs are available, the dialog with DEBUG can be carried on ’simultaneously’ in several processors during the various MMI sessions.
Termination of the intermediary does not affect the state of DEBUG.

2.1.1 The AMO DEBUG (AMO)

The AMO interface is activated with:

```
EXEC-DEBUG;
```

2.1.2 The Miniterminal Handler (MTH)

The MTH is activated by entering 'CTRL D'.
All entries must be made in capital letters.

2.2 DEBUG Commands - Overview

Command	Meaning	Type
ACT	Activate test job	in / op
AT	Define test job	in
BREAK	Stop processor	in / op
CONT	Continue task	in
D	Display (identical with TRACE)	in / op
DCL	Define DEBUG variable	in
DEACT	Deactivate test job	in / op
DEF	Define symbols	in
DEND	Terminate dialog	spec
DOWHILE	Define loop condition	in / op
END	Terminate input of statements	in
EXEC	Call exit routine	in / op
FILE	Allocate log file	spec
GO	Continue processor	in
GOFOR	Terminate break mode	in
GOTIL	Set breakpoints	in
IF	Define conditions	in / op
IN	Enter statements in dialog mode	spec
INSP	Output system object data	in / op
LIST	List user information	spec
LOGOFF	Deactivate trigger entry	in / op
LOGON	Activate trigger entry	in / op
MCALL	Call macro	in / op

Table 1 DEBUG Commands (in Alphabetical Order)

Command	Meaning	Type
MDEF	Define macro	in
PRINT	Print information	spec
REDCL	Redefine DEBUG variable	in
REDEF	Redefine DEF symbol	in
REMAT	Delete trigger point	in
REMMAC	Delete macro	in
REMOVE	Delete breakpoint	in
RESET	Reset trace buffer	spec
RUN	Read in statements from file	spec
RUNOFF	Stop logging of entries	in
RUNON	Log entries in runfile	in
SAVEOFF	Deactivate patch function	spec
SAVEON	Activate patch function	spec
SEL	Set trigger mode selection	spec
SET	Modify memory contents	in / op
START	Start DEBUG session	spec
STOP	Stop task	op
TERM	Terminate DEBUG session	spec
TRACE	Trace memory contents (display)	in / op
TRIGGOFF	Deactivate trigger mode	spec
TRIGGON	Activate trigger mode	spec
VIEW	List system objects	in / op
WRITE	Write comments	in / op
!	Execute instructions	in
;	Dummy statements	in / op

Table 1 DEBUG Commands (in Alphabetical Order)

Type:	in	Instruction
	op	Operation
	spec	Special command

There are different command levels:

- control level,
- interactive level and
- trigger point level.

The commands are categorized in accordance with these levels.

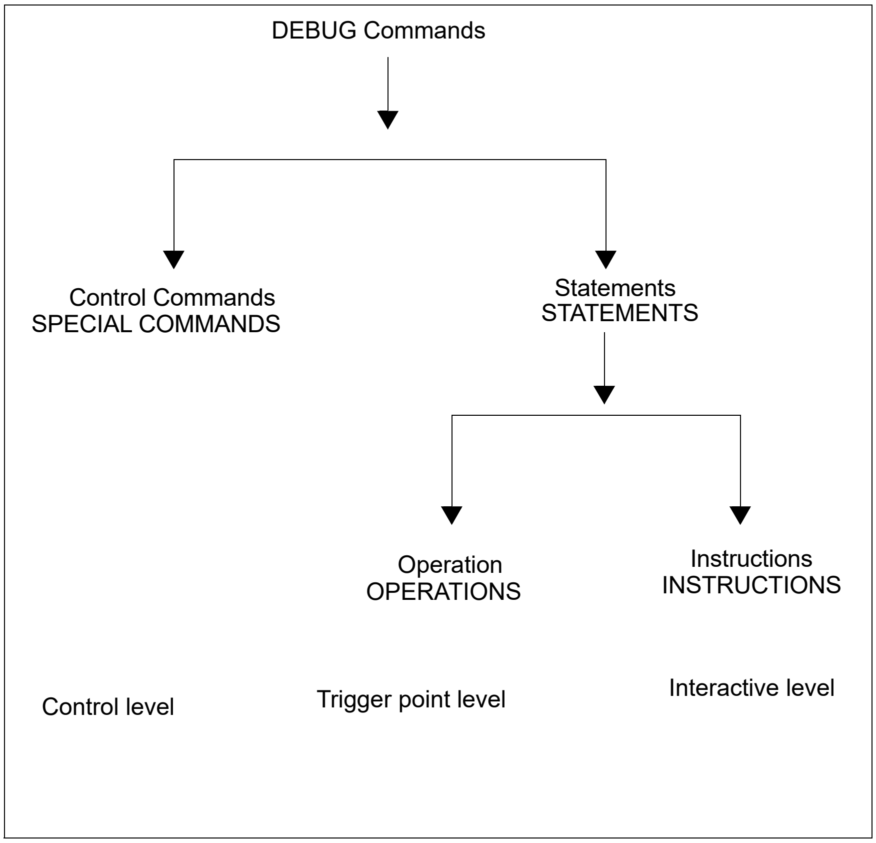


Figure 1 DEBUG Commands overview

DEBUG, commands:- special commands

The special commands serve to perform functions which are not directly used for collecting information. These commands are used in particular for allocating log files, printing out information and for starting and terminating a DEBUG session.

DEBUG Statements

DEBUG commands are used for collecting information. Written in a language that resembles CHILL, they are instrumental in informing DEBUG as to which diagnostic operations are to be performed. DEBUG instructions may be contained in runfiles. Runfiles are not permitted to contain special commands. DEBUG commands are subdivided into instructions and operations depending on whether the statement is carried out in interactive mode or at the trigger point. In addition to the functions that can be performed only at the trigger point or only in interactive mode, there are functions that can be performed in both states. There is, for instance, a SET operation and a SET instruction. There is no difference in the syntax of operations and instructions.

Operations

Operations are DEBUG commands that are carried out at the trigger point. Operations refer to memory states prevailing at the time when the trigger is reached.

Instructions

Instructions are DEBUG commands that are executed interactively. They refer to memory contents existing at the input time (or more precisely at the time they are executed).

Command	Longname
DO ... OD	Enclose operations at AT
FI	Terminate IF construction
MEND	Terminate macro operation at MDEF
OD	Terminate operations in DOWHILE
THEN,ELSE	IF construction branches

Table 2 Auxiliary Keywords (in Alphabetical Order)

2.3 Entering DEBUG Commands

Entries are checked by DEBUG (see handling of syntax errors). Special commands cannot be continued over several lines; they are limited to 160 characters. If the special commands are entered at the interactive level, they are limited to 80 characters.
A DEBUG statement can be split into several entries. On the other hand, several DEBUG statements can be transmitted with a single entry; the length of an entry being limited to 80 characters. Comments in special commands and statements are permitted wherever a BLANK (’ ’) is permitted. They are enclosed in the delimiters /*... . length (even extending over several entries).
The syntax of the individual commands is described in the sections ’Description of DEBUG commands’ and ’Syntax in BNF’.

2.4 Special Characters Used in the Commands

%	Identifiers for DEBUG variable and DEBUG mode
#	for D R or Identification of a pointer value #selector:offset or Identification of the job code for D
&	Identifier for DEF symbols

;	End of a DEBUG statement (not carried out immediately), or Dummy statement
,	for separating operands in DEBUG statements
:	Separating label name and AT keyword, or Separating the components of an address entry or Separating the components of a time entry
(	for Length entry and for parenthesis in DEBUG statements
)	for Length entry and for parenthesis in DEBUG statements
=	Allocation in SET, DEF and REDEF statements, or Relation operator for IF and DOWHILE
>	Relation operator for IF and DOWHILE
<	Relation operator for IF and DOWHILE
->	Dereferencing symbol in expressions in expressions
+	in expressions
*	in expressions or in procedure-related addresses (current procedure), or for sub-qualification of trigger point names
-	in expressions
/	in expressions or Separation of the date entry components
/*	Start-of-comment character
*/	End-of-comment character
’	for selecting the numeric representation (T’,B’,H’), or HEX-strings delimiter (X’), or CHAR-strings delimiter (’..........’)
.	in names and figures for subdividing character strings
_	in names and figures for subdividing character strings
!	End of a statement (carried out immediately), or Initiating statements which have not yet been executed

2.5 DEBUG Dialog States

After restart, outside the DEBUG session, DEBUG is at the control level. At the control level, the second prompt character is a BLANK (’ ’).

After a restart, during a DEBUG session, DEBUG is at the interactive level and requests an instruction to be entered (irrespective of the state prior to restart). In this state, the second prompt character is ’*’.

A DEBUG session is opened by a START command. Working with DEBUG requires that a session be opened beforehand. In doing so, all definitions made in the previous session are deleted. The START command gives direct access to the interactive level. At this level the second prompt character is not equal to BLANK (’ ’).

Only the TERM command can be used for quitting the DEBUG session. When the debugger quits the DEBUG session, all DEBUG activities are stopped. After the TERM command, the debugger is placed at the control level.

The ’IN’ command shifts the debugger from the control level to the interactive level. If the command is given at the interactive level, the debugger will stay at this level.

From the interactive level, the debugger is moved to the control level through an END statement.
At the interactive level, all commands can be given; at the control level, only special commands can be given.

2.6 Prompting

In addition to the subdivision into control level and interactive level, other system states have an effect on the validity of the commands. They are, therefore, indicated to the user in the input request with the aid of prompt characters.

A prompt invariably comprises three characters:

- the first character indicates the system state,
- the second character indicates the dialog state in DEBUG,
- the third character indicates the dialog state with the ’intermediary’.

First prompt character (system state)

Meaning		effect on the commands (- : invalid) (+ : possible addition)	
+	Outside the DEBUG Session	- - -	TERM, FILE, IN, RUN, RESET, SAVEON, SAVEOFF, TRIGGON, SEL TRIGGOFF,
*	During the DEBUG Session	-	START
#	SAVEON is set	-	GO, GOTIL, BREAK, REMOVE, GOFOR
?	Break mode activated	- - -	RUN, SAVEON, SAVEOFF, FILE IN: Parameter RUNFILE PRINT: Parameter INFILE
$	Processor stopped at trigger	+ + + -	Use of register names when entering numerical values Address entries relating to a procedure in instructions (P*, PB, ..) D #R as an instruction as for ’?’
%	Processor stopped interactively	-	as for ’?’
/	Comment mode	-	Everything ignored except for */

Table 3 First Prompt Character

IMPORTANT: The comment mode of the control level is indicated at the control level only, the system status is indicated at the interactive level.

Second prompt character (dialog state of the interactive level)

	Meaning
*	Requesting an instruction, all previous ones having been executed
	Debug at control level (see note)
+	Restart (see note)
!	Requesting an instruction, previous one not having been executed
:	Request inside a trigger point definition
-	Request within IF or DOWHILE
/	Request within a comment
.	Request within a macro definition
;	Statement not yet completely entered.

Table 4 Second Prompt Character

IMPORTANT: A BLANK (’ ’) indicates that the system is not at the interactive level but at the control level
The ’+’ character in this position indicates that this is the first prompt following a restart. As a rule, this prompt appears on the screen only for a short time, because it is overwritten by the MTH message V120.

Third prompt character (state: dialog with the intermediary)

	Meaning
>	Within the dialog with the intermediary
<	Outside the dialog with the intermediary (for MTH only)

Table 5 Third Prompt Character

Typical prompt characters

+ >	Debug is outside the session (at the control level)
* >	Debug is within the session at the control level
**>	Debug is within the session at the interactive level
$.>	While the processor is stopped at a trigger, a Macro is defined at the interactive level.

2.7 Input via AMO DEBUG

The AMO interface is activated with:

```
EXEC-DEBUG;
```

Followed by the question which processor the dialog should be opened for. Once the processor identification has been input (it may also be entered together with the ’EXEC-DEBUG’ call) the DEBUG prompt appears followed by the AMO input request. Now an entry may be made. The entries should be embedded in the MMI syntax of the AMOS. This means that the ’ " ’ character must appear at beginning and end of an entry. If this is not the case, the special characters are interpreted by the MMI and not forwarded to DEBUG. This should be observed in particular when entering ’;’ and ’!’. If a ’;’ character is to be forwarded to DEBUG, ’";"’ should be entered: The dialog is carried on with the processor specified upon activation. If a dialog is to be carried on with another processor, the AMO must be deactivated and then reactivated again. The AMO is deactivated with the DEND command. If there is no entry for more than 15 minutes, the AMO will deactivate itself.

IMPORTANT: The BREAK, GO, GOFOR, REMOVE, GOTIL statements must not be entered via this AMO interface. During break mode, (distinguishable by prompt characters ’?’, ’%’, and ’$’) no entries must be made. Deadlock occurs in any case when the processor is frozen up, since all tasks (with the exception of MTH and DEBUG) are suspended at freeze-up.
When TRIGGON is activated, logging takes place via the MTH interface only (MTH does not have to be activated for this purpose).

Example: (E): this line was input
(A): this line was output
(AE): this line contains inputs and outputs

```
(A )   EXEC-DEBUG:A1;
(A )   H500:          AMO DEBUG STARTED
(A )   + >
(A )   PLEASE ENTER DEBUG FOR 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 )   PLEASE ENTER DEBUG FOR A1
(AE)   *"d 2BF8:12(6);"

(A )   *!>
```

```
(A )   PLEASE ENTER DEBUG FOR A1
(AE)   *"dend"
(A )   AMO-DEBUG-219          MAKING HICOM DEBUG AVAILABLE
(A )   EXEC EXECUTED;

(A )   EXEC-DEBUG:A1;
(A )   H500:          AMO DEBUG STARTED
(A )   *!>
(A )   PLEASE ENTER DEBUG FOR A1
(AE)   *"!"

(A )   **>
(A )   PLEASE ENTER DEBUG FOR A1
(A )   *
```

2.8 Input via Miniterminal Handler (MTH)

The MTH is activated by entering ’CTRL D’. If it is not known whether MTH must be activated, or what status the dialog is in, the status is displayed by depressing the return key (in dependence on the terminal device!!). Either DEBUG prompting or the following message will appear:

```
MTH V120: THE FOLLOWING APPLICATIONS HAVE BEEN DEFINED:
01 DEBUG DIALOG(13) CAN BE STARTED WITH ’CTRL D’!
```

The MTH must be activated only in the second case (the MTH message tells which processor is being accessed). When the MTH has been activated, the DEBUG prompt appears. The processor to which the terminal is connected can now carry on the dialog with DEBUG. When the dialog is to be terminated, the ’DEND’ command must be entered.

2.9 Sporadic Messages

Sporadic messages are messages which are not immediately output in answer to a DEBUG instruction. Typical sporadic messages are messages concerning the change of protocol files and the processor-stop message. Sporadic messages are output both via the the AMO interface (provided a dialog has been set up) and via the MTH interface. Output of the sporadic messages is stopped while a command is being processed.

Characteristics of the MTH interface:

The sporadic messages are output irrespective of whether or not the dialog with the MTH has been opened. They are concluded with ETX(=H’03) (for application via DASIST).

When the ESC key is pressed, the output of sporadic messages is stopped until the return key is pressed again.

2.10 Characteristics of Monoprocessor Systems (Hicom 3x3)

Monoprocessor systems do not have a mini-terminal handler. Instead, the MTH interface is emulated by the CMS. The service terminal can only be plugged into the second V.24 interface of the DM80 in the 3rd row of the CC80 shelf (see DM80 description in "Modules" chapter of the Hicom 300 Service Manual).
This V.24 interface must be configured as an "asynchronous terminal" interface, in order to be able to use the debugger.

Press the keys "CTRL" and "D" simultaneously to set up a dialog connection to the debugger. The dialog connection is terminated by entering "DEND".

Sporadic debugger-messages can only be output during a dialog session.

IMPORTANT: It is, of course, also possible to open a "Terminal Task Session" at the V.24 interface. Enter "CTRL T"
Always terminate a terminal task session before starting a debugger dialog session with "CTRL D".

---

3 Function Description of DEBUG Special Commands

complete description of all input variants of a command can be seen from the syntax diagrams.

The following number entries in DEBUG statements are possible:

- hexadecimal (H'....),
- binary (B'....)

or

- decimal (T'....).

Unless specified, hexadecimal numbers are assumed.

3.1 START Command (Start of a DEBUG Session)

The DEBUG session is opened. This is a requirement for testing with DEBUG. This command serves to initialize the internal DEBUG tables and regions. When this command has been executed, no test orders, macros or DEBUG variables are defined. As to the DEF symbols, only the standard DEF symbols are defined and re-initialized. The trace buffer is empty. The start acknowledgement contains the DEBUG version, the date and time of day as well as the processor number (decimal und hex.). The START command changes to the interactive level.

Example:

```
START;
0304I START  :DEBUG V300 FT2 :STARTS AT:
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 Command (Activating the Patch Function)

This command is used to activate the patch function in order to change code or data on the hard disk. Following SAVEON all subsequent SET instructions are executed on hard disk. The main memory remains unchanged. It is possible to use the SET commands (e.g. in a runfile or in a command stack) to make the changes initially in the memory (in the SAVEOFF state), then to test the changes, and subsequently to carry out the changes on hard disk using the identical SET commands (runfiles, stacks) in the SAVEON state. This ensures that the same changes are made in the memory and on hard disk. ’Patching’ cannot be linked to a trigger. At a trigger, SET instructions are executed in the memory only, even after SAVEON. Reloadable subsystemes (AMO) cannot be patched with DEBUG. Data not initialized in the program must be patched dynamically (by inserting an appropriate code in a patch area). The patch function in DEBUG should be considered only an expedient. It offers no support to administration and recording of the changes performed.

After each RUN command and each return to the control level, the SAVEOFF state is set. This state can be recognized by the prompt character (system state: #).

IMPORTANT: Before executing a special command, which is initiated from the interactive level, the system state is set back to SAVEOFF.

3.3 SAVEOFF Command (Deactivating the Patch Function)

This command is used to deactivate the patch function. SET instructions are now only effective in the memory.

3.4 TERM Command (Terminating a DEBUG Session)

This command terminates the DEBUG session. All test jobs are deactivated. The system waits for any running test order processing operation to be terminated. If it is not terminated within 5 seconds, a warning is output together with the Term message. This message is also output when some stack areas are still assigned though test order processing has ceased. In case a log file is used, the entire trace buffer is saved before the file is closed.

Example:

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

3.5 RESET Command (Resetting the Trace Buffer)

This command serves to re-initialize the trace buffer, the old contents being lost. The command is rejected in case logging to an assigned log file is still in progress.

3.6 IN Command (Entering Instructions Interactively)

This command transfers the system (data) from the control level to the interactive level, where special commands as well as instructions can be entered. This level can be recognized at the prompt character (2nd character not equal to BLANK). Once the system is at the interactive level, it will stay there; however, the save identifier is set to SAVEOFF.

With the ’IN’ command, a file can be entered with parameter ’RUNFILE=file name’; in this file entries can be logged simultaneously at option. The simultaneous logging procedure is controlled with RUNON and RUNOFF. If the specified file exists, it will be continued. (The name of the runfile is composed of ’:AMD:R/’ and the specified file name).

Example:

```
IN:RUNFILE=PROT1
```

IMPORTANT: The command is, as a rule, required only after a restart, since all commands are possible at the interactive level and therefore it is not necessary to quit this level.

3.7 RUN Command (Entering Instructions from the File (Runfile))

This command is used to read commands from a file. The file from which data is to be read in is identified by keyword ’INFILE’. (The name of the runfile is composed of ’:AMD:R/’ and the specified file name).
When a wrong instruction is detected, the readin procedure is aborted; this can be prevented by entering the FUNC=CHECK parameter. The FUNC=CHECK function is, however, not generally enabled, since errors may interfere with the desired procedure. For example operations orginally linked to triggers might turn into immediately executable instructions. Error messages are output in the same way as with direct input. If the procedure is aborted, the line number of the first faulty statement is output as well. Valid instructions are executed and acknowledged as for the interactive level. Abortion on account of a file access error is separately reported.

The optional PROT=NO parameter can be used to suppress the AT statement acknowledgement. This makes sense whenever a great number of test orders are specified in a runfile.

Example:

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

Runfiles are files of the type BYTEFILE resident on a hard disk whose record length is required to be 80 bytes. (The name of the runfile is composed of ’:AMD:R/’ and the specified file name).

3.8 FILE Command (Controlling the Trace Buffer Logging Operation)

This command is used to assign DEBUG a log file on hard disk. Thus the trace buffer contents can be logged to this file. The keywords ’CURFILE’ and ’SUCFILE’ are used to specify the file that is to be written to. If only one file has been assigned, logging is terminated when the file is full. If two files have been assigned, both files are alternately used.

Parameter CUR=CLOSE can be used to close the file that has been assigned with CURFILE. Analogously, parameter SUC=CLOSE is specified to close the file assigned with SUCFILE. A closed file is no longer used for logging. A switchover to the remaining file is initiated automatically. A closed file can be reassigned with CURFILE and SUCFILE. (The name of the log file is composed of ’:AMD:P/’ and the specified file name).

Examples:

```
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: If the only file that is still assigned is closed with SUC=CLOSE or CUR=CLOSE, logging is aborted immediately. Hence, it is possible that not all of the trace buffer entries have been written.
If a file assigned in "File" does not yet exist, it will be set up by DEBUG (length: 10000 bytes). If it does exist, it will be overwritten. The file should, however, be a SAM file.

3.9 PRINT Command (Printing the Trace Buffer and the Log File)

This command may be used outside the DEBUG session as well. There are two PRINT command applications:

- Without parameter INFILE, the trace buffer is output;
- with parameter INFILE, the specified log file is output.

Only closed files can be output. (The name of the log file is composed of’:AMD:P/’ and the specified file name). A file is closed when it is full (a message to this effect is issued to the user) through a FILE command with CUR=CLOSE or SUC=CLOSE, or through the TERM command.

The user can select output of the trace buffer or the log file by specifying selection criteria. If parameter INF=CUE is entered, output is suppressed altogether, the system only lists the number of entries that are available with the specified selection.

Possible selection critera for the PRINT command:

- Time (TIMEB, TIMEE)

  Only entries made between TIMEB and TIMEE are output. Default value for TIMEB is 00:00:00, for TIMEE 23:59:59.

- Date (DATEB, DATEE)

  Only entries made between DATEB and DATEE are output. Default value for DATEB and for DATEE: current date

- Daily (PERIOD)

  Only entries made between TIMEB and TIMEE are output, on the days between DATEB and DATEE. If the PERIOD parameter is not entered, all entries that have been made between DATEB,TIMEB and DATEE,TIMEE will be output.

- Trigger entry

  Only the trigger entries are output. This supports program sequencing. If this parameter is not entered, all entries will be output (i.e. also TRACE, ACT, etc.).

- Label name (LABEL)

  This parameter serves to output only those entries that belong to the test orders which match the specified name. With a partially qualified label (for example LABEL=AB*), all entries of test orders that match the specified partial name are output. If LABEL is entered, no instructions will be output.

- The last entries (REC)

  This is possible only when the trace buffer is output. Output is limited to the last n entries; only those of them being output, that are subject to selection. If the TEC parameter is entered in addition, REC identifies the number of entries to be output (with special consideration to selection) (values for n: 1 bis 255).

- Defining the first entry (TEC)

  This is possible only when the trace buffer is output. TEC specifies after how many entries output should be started. Counting always starts with the oldest entry. If TEC is not entered, output will start with the oldest entry. If the trace buffer is written to during output, the entry to be output is likely to be overwritten. A message is issued and printing is aborted. If the trace buffer does not contain as many entries as indicated by TEC, TEC is ignored (possible values : 1 to 65535)
  Any combination of the above selection criteria is possible.

Examples:

```
PRINT:TIMEE=14:30:00,DATEB=10/01/83,TLOG=ON
                                       Year
                                    Month
                                 Day
```

Only the trigger entries that were created between 10.1.83, 0.00 hours and until 14:30 hours on the current date are output from the trace buffer.

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

All entries created daily between 00.00 hours and 15.20 hours from 17.02.84 until, are output from the log file specified.

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

The last 5 trigger entries whose label starts with ’A’ are output from the trace buffer.

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

The number of trigger entries present in the trace buffer is output.

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

The last entry in the trace buffer is output. (Trace buffer does not hold 60000 entries)

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

Five entries are output, starting with what is currently the 333rd entry.

3.10 TRIGGON Command (Activate Trigger Mode)

This command activatess the mode in which test order activities are directly output on the terminal.
This mode is called trigger mode.
Logging takes place only via the DP286 interface using the Miniterminal Handler (MTH), irrespective of whether the dialog is carried on via AMO DEBUG or via Miniterminal Handler. Before the trigger mode can be activated, the trigger selection must have been entered (see SEL command). However, if, for some reason, the trigger command has been issued beforehand, DEBUG generates a SEL command with the default criteria. Triggering starts with the entry which, at activation time, was the last entry in the trace buffer. If the trace buffer entry operation is faster than logging, a message to this effect will be issued. Then the procedure continues with the most recent entry. The trigger mode is independent of the logging to hard disk (log file) procedure. Even with outputs in progress,

entries can be made. To this end, the ESC key (at Qume) must be pressed (H’1B). Triggering is then suspended until the entry is terminated with the RETURN key. If entries are terminated with the ESC key, triggering will stay suspended.

3.11 TRIGGOFF Command (Deactivate Trigger Mode)

This command can be used to deactivate triggering.

3.12 SEL Command (Set Trigger Mode Selection)

This command can be used for setting the selections for the trigger mode. This command does not yet activate the trigger mode. The criteria remain set until a new SEL command is given. This may also be done when the trigger mode is already ON. The SEL command syntax is identical with the PRINT command syntax, i.e. the same parameters as for PRINT can be specified. However, the parameters INFILE, TEC, REC and INF are not evaluated. If LABEL is not specified, default value LABEL=’*’ is used. If logging (triggering) is to continue after midnight (00.00 hrs.), parameter DATEE must be specified, since the actual date (at entry time) is the default value.

3.13 LIST Command (Output of Lists)

This command may be used to output the following lists:

List of all triggers, at which tasks are stopped,

- List of all DEF symbols,
- List of standard DEF symbols,
- List of newly defined DEF symbols,
- List of all DEBUG variables,
- List of all trigger points or breakpoints
- List of all macros.

The desired list is identified by keyword ’TAB’.

Examples:

```
LIST:TAB=STOP      (Display of stopped tasks)
LIST:TAB=DEF       (Display of DEF symbols)
LIST:TAB=DEFS      (Display of standard DEF symbols)
```

```
LIST:TAB=DEFN     (Display of newly defined DEF symbols)
LIST:TAB=DCL      (Display of DEBUG variables)
LIST:TAB=AT       (Display of test jobs)
LIST:TAB=MAC      (Display of macros)
```

Examples (with output):

```
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 Command (Define Symbols)

The DEF command assigns an address and a length to the specified name (maximum 26 characters). These DEF symbols can be used instead of numeric addresses in AT and D statements.

A DEF statement comes in two different formats:

- the address that is to be assigned to the symbol is specified either numerically or
- through an existing DEF symbol and a relative address (offset).

In the first case, a specific length must be specified, in the second case, and unless its length is explicitly specified, the new symbol is assigned the old symbol length.

Examples:

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

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

   DEF &HALT = &MARKE + 20;

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

Explanation:

Address 1238:5678 is now assigned the name of &MARKE in length 1, and address 4E68:10D0 is assigned the name of &BUFFER in length H’20. (Command ’D &BUFFER;’ would be used to dump the memory contents in length H’20) from this address onward. Subsequently, symbol &HALT is assigned the address 1238:5698 in length 1. Memory area &BUFFER2 is assigned the address of &BUFFER, with the offset increased by decimal 50 with the hexadecimal length 200, in this case: address 4E68:1102.

IMPORTANT: For certain applications, standart symbol definitions have already been provided. See the following sections.

Standard DEF symbols to suppress acknowledgements:

For controlling the logging of DEBUG activities in the trace buffer and on terminals, the following symbols are provided:

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

TRUE is preset, i.e. the statements are written in the trace buffer and/or output. ’SET <symbol> = FALSE;’ serves to suppress entry of the corresponding statement in the trace buffer and ’SET <symbol> = TRUE;’ to re-enable it.

Examples:

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

SET instructions are no longer entered in the trace buffer.

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

ACT instructions are again entered in the trace buffer

Standard DEF symbols to write instructions into the trace buffer:

```
   &WRITE_IN
```

FALSE is preset, i.e. the instructions are not written into the trace buffer and/or output.
’SET &WRITE-IN = TRUE!’ serves to activate entry into the trace buffer and ’SET &WRITE_IN = FALSE!’ to deactivate it again.

Standard DEF symbols for Exit routines:

```
&PATCH1

&PATCH2

&PATCH3

&GG_P_AR_ONE

&GG_P_AR_TWO
```

These are symbols of addresses that are accessed by DEBUG when the EXEC instruction is used at a test point (trigger). For the first three symbols, 18 bytes are available, otherwise 14 bytes are available for coding (e.g. exit to a patch area). The test object registers are not made available.

Standard DEF symbols to choose the language:

The language in which error messages are output may be either GERMAN or ENGLISH. ENGLISH is preset.

with

```
SET &LANGUAGE = ’D’!
```

a switchover to GERMAN is initiated and

with

```
SET &LANGUAGE = ’E’!
```

a switchover is made to ENGLISH.

3.15 REDEF Command (Redefine Symbol)

REDEF is used to assign a new address and/or length to an already existing symbol. If the length is not specified, the old symbol length is retained. As with the DEF instruction, the address assigned to an existing symbol may be specifed numerically or with the aid of a DEF symbol.

Examples:

```
DEF &A 1238:0 (7);

DEF &B = &A + 100 (5);

REDEF &A = &B + 10;        /* &A: address 1238:110, length 5 */

REDEF &B 6458:21 (1A);

REDEF &A E100:123;         /* &A: address E100:123, length 5 */
```

3.16 DCL Command (Define DEBUG Variable)

DCL defines variables (name max. 26 characters) with modes %INT, %BYT, %POI, %CHAR or %BOOL. Up to 6 variables of the same mode can be defined in parenthesis. DEBUG variables can be modified by way of SET instructions and output by way of D instructions. They can be used in address modifications and in expressions (= expressions with the following arithmetic operations + , - , * , /).

Examples:

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

DCL %ZAHL %INT;

DCL %B %BOOL;

DCL %C %POI;
```

Now the following variables exist: the character mode variables %ABC and %XYZ, the pointer variable %C, the integer variable %ZAHL and the Boolean variable %B which can be changed for example with the following SET instructions

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

SET %B = TRUE;

SET %ZAHL = 5;

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

or whose contents can be output for example with

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

.

3.17 REDCL Command (Redefine DEBUG Variable)

REDCL can be used to provide an already existing variable with a new mode. This redefinition is not effective in operations where this variable has already been used. The REDCL instruction does not reinitialize the variable.

Examples:

```
DCL %B %BOOL;

REDCL %B %INT;
```

Firstly, a Boolean variable %B is defined. It is converted into an integer variable with REDCL.

3.18 AT Command (Define Test Order)

AT is used to define a test order for the specified address. It can be accessed with the label (name of up to 8 characters). The operations that are intended to take place at the test point are delimited by keywords ’DO’ and ’OD’. Test orders without operations are admissible as well. (Application: if reaching the trigger point is of importance). Within a test order, an entry is requested with a prompt, whose dialog status is ’:’ (for example with *:>).

IMPORTANT: Test orders must be defined before they can be activated. However, mere definition of a test order will not initiate any test activities. Once a test order has been defined, it can be activated and deactivated any number of times, as long as it is not deleted with the REMAT statement. Before an AT command can be processed, all instructions that have not yet been executed are forcibly executed.

The data collected at a trigger point is written into the trace buffer and can be saved in parallel in a log file. It can be output on a terminal with PRINT.

Examples:

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

Now, test order LABEL is defined at address E680:C69. It may be activated with ’ACT L1AX2;’. On running through an active test order, the operations specified after DO are carried out. Test order GRISMURD is defined at the address defined by &MARKE. This test order does not require any operations to be carried out. Only attaining this test order is logged.

3.19 REMAT Command (Delete Test Order)

REMAT enables deactivated test orders to be deleted. Subsequently, they can no longer be reactivated or run through again. The names of deleted test orders can be used again. If all test orders are to be deleted, ’ALL’ must be specified instead of label name. In doing so, care should be taken to delete only inactive test orders. If a test order cannot be deleted, a message will be issued.

Examples:

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

First test order LAB1 is defined. It is deleted again with REMAT LAB1. REMAT ALL deletes all deactivated test orders.

3.20 MDEF Command (Define Macro)

MDEF defines a macro which is addressed using its macro name (up to 8 characters).
A maximum of 20 macros may be defined.
The MDEF instruction serves to combine the following operations (up to MEND) into one macro operation. Within a macro definition, an entry is requested with a prompt whose dialog status is '.' (for example with *.>).

Before the MDEF command is processed, all instructions that have not yet been executed are forcibly executed.

The macro operation can be executed by calling the macro (MCALL).

Example:

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

3.21 MCALL Command (Call Macro Operation)

MCALL is used to call a previously defined macro.

Example:

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

3.22 REMMAC Command (Delete Macro)

REMMAC is used to invalidate a previously defined macro. The macro can be reactivated by redefining it (with the same name). A macro deleted with REMMAC will be ignored upon MCALL. MCALL will not be carried out again before the macro has been redefined with the same name.

REMMAC will not make any memory space available in the macro table. If a table is full, it remains full.

Example:

```
REMMAC MACNAME1;
```

3.23 RUNON Command (Runfile Online: Activate Logging)

Following these instructions, entries are written in the file (runfile) specified with the ’IN’ command. The RUNON instruction must be placed in a separate line. Wrong entries are logged as well. RUNOFF is set after the ’IN’ command. In case no file has been assigned with IN, this instruction has no effect.

3.24 RUNOFF Command (Runfile Online: Deactivate Logging)

This instruction terminates logging of entries in the file. Changeover from RUNON to RUNOFF can be repeated as often as desired. The ’IN’ command is followed by RUNOFF.

IMPORTANT: The files generated can be specified as runfiles in the RUN command.
The RUNOFF instruction should be placed in a separate file, otherwise it would not be clear whether the entry containing RUNOFF is written or not.

3.25 EXEC Command (Start Exit Routine)

EXEC <routine> is used to start exit routines; i.e. DEBUG branches to the procedure assigned to <routine>. The procedures have been provided by DEBUG as dummy procedures. They must first be supplied with the desired commands through patching. As a rule, an additional patch area is required, the length of dummies being only 14 to 18 bytes. As an alternative, dummies may be replaced by the desired procedures in the DEBUG source. However, this would require a new production of the Hicom debugger.

Example:

```
EXEC GG_P_AR_ONE;
```

3.26 STOP Command (Stop Task)

STOP will place in the wait state any task that has reached a trigger point where the STOP command is defined. The remaining tasks continue running. When a task is stopped, this is displayed at the terminal. In addition to the STOP message, the current register contents of element 80286 of the stopped task are output.

3.27 CONT Command (Continue Task)

CONT will continue all tasks stopped at a trigger point. ’CONT ALL;’ will continue all tasks.
In case several tasks have been stopped at a trigger point, they can only be continued together.

Examples:

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

3.28 BREAK Command (Stop Processor)

If this command is issued as an instruction, two functions will be performed.

- Firstly, the break mode is activated, provided it was still inactive. As a result, BREAK operations will be effective from this time onward.
- Secondly, the processor is stopped. In this instance, the first prompt character (system status) is set to ’%’.

If an operation at a trigger point is concerned, the processor is stopped only in case the BREAK mode has already been activated. Otherwise, the operation is ignored and a corresponding message output. If the processor has been stopped by the break operation, a message is ouptput. In this case, the first prompt character (system status) is set to ’&’. The registers are available for addressing, D #R is allowed, procedure-related D is possible; with VIEW and INSP, entries T* and J* relate to the task that initiated the processor stop.

3.29 GO Command (Continue Processor)

This command comprises two functions.

- Firstly, it activates the break mode, provided it has not been activated yet. This causes the BREAK operations to become effective as from this moment.
- Secondly, the processor is thawed (continued) if it was frozen (stopped). This function has no effect on the definition of breakpoints. In this case, the first prompt character (system status) is set to ’?’.

3.30 GOFOR Command (Deactivate Break Mode)

This command has several subfunctions.

- The break mode is switched off. This causes the BREAK operations to become ineffective as from this moment.
- The processor is thawed (continued) if it was frozen (stopped). The breakpoints are deactivated and deleted. In this case, the first prompt character (system status) is set to ’*’.

3.31 GOTIL Command (Define Breakpoints)

This command has several subfunctions.

- The break mode is activated. This causes the BREAK operations to become effective as from this moment.
- The processor is continued if it had been stopped.

Breakpoints are defined and activated at the addresses specified. The name of the breakpoints is constructed from the specified address: the numerical entries of selector and offset are combined with leading zeros to form an 8-character name (4F8:120 becomes 04F80120). The breakpoints in Debug messages are identified by this name. If a breakpoint is reached, the processor is stopped and a message is issued via Miniterminal Handler (MTH). In this case, the first prompt character (system status) is set to ’&’. The registers are available for addressing, D #R is allowed, procedure-related D is possible; with VIEW and INSP, entries T* and J* relate to the task that initiated the processor stop. However, as before, only instructions but no operations (i.e. no IF, DOWHILE, MCALL,....) are permitted.

Example:

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

IMPORTANT: If the address is located in the Miniterminal Handler (MTH) or DEBUG, the processor will be stopped, however, no registers will be available. In this particular instance, the first prompt character is ’%’ ;

3.32 REMOVE Command (Delete Breakpoint)

The breakpoints defined with GOTIL are deactivated and deleted with this function. The breakpoints are not accessed under their generated names but under their addresses, i.e. exactly as for the GOTIL command.

3.33 DOWHILE Command (Define Loop)

DOWHILE serves for defining loop operations. The loop definition is terminated with 'OD'. A loop should contain at least one operation. Within DOWHILE an entry is requested with a prompt, where the dialog status is '-' (for example with *->). The execution of operations depends on the condition specified. This condition can be any logical or arithmetic operation with DEBUG variables, constants or addresses (length 1,2 or 4).
DOWHILE can have 7 nesting levels.

To avoid endless loops in trigger point processing, the total number of trigger loopings is limited to 255.

Examples:

```
       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 'LOOP';
       OD;
```

```
MEND;
```

3.34 ACT Command (Activate Test Order)

ACT activates one or several test order previously defined with AT. The test order is addressed under its label. If all test orders are to be activated, the label name to be entered is ’ALL’. Interrupt INT3 is diverted to DEBUG at the first ACT after a restart.

Examples:

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

The Call ’ACT ABC,XYZ;’ will activate the test orders ABC and XYZ. When passing through trigger ABC, test order LABEL is activated at the same time. The ’ACT ALL;’ call will activate all deactivated test orders.

3.35 DEACT Command (Deactivate Test Order)

DEACT deactivates test orders. The test order is addressed under its label. If all test orders are to be deactivated, ’ALL’ must be specified as a label name.

Examples:

Deactivate LABEL test order

```
DEACT LABEL;
```

Deactivate GRISMURD and LABEL

```
DEACT GRISMURD,LABEL;
```

Deactivate all active test orders

```
DEACT ALL;
```

Upon reaching the trigger point XYZ, deactivatte all test orders including the test order XYZ

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

3.36 LOGON Command (Activate Trigger Entry)

After LOGON, a trigger message is written in the trace buffer when a trigger is reached. After the START command, LOGON is set.

3.37 LOGOFF Command (Deactivate Trigger Entry)

After LOGOFF, no trigger message is written in the trace buffer when a trigger is reached. This does not apply to trigger points where operations have to be logged. For these, a trigger message is also transmitted after LOGOFF.

Example:

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

With LOGOFF a trigger entry is made only if the D command is executed as well. In all other cases, no entry will be made in the trace buffer.

3.38 D Command (Display Memory Contents)

D is used to display memory contents of any length, DEBUG variables, or the stack contents and register contents at the indicated code or data addresses. If the area that is to be output goes beyond the defined area, the display is output up to the end of the area defined by the selector. A message indicating that the display was shortened is issued. The representation of the contents is determined with the aid of a 'display picture'. It can be hexadecimal, binary, in the form of a character string or in dump format. If no value is specified, output is in dump format.
If desired, the user may specify a job identifier, which is then output together with the display acknowledgement. This allows the user to identify a display by means of a code of his choice, rather than by checking its address. The job identifier is output before the user data (address = contents). If no job identifier is specified, 0000 will be output. If the job identifier is #FFFF, only the user data will be output. For an explanation of the output format, see section 'Messages', under 'Additional Information'.

Examples:

```
(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;
```

Explanations:

(1) One byte of the indicated address is output in hexadecimals (default value).

(2) The bytes between Offset H'5678 and H'5721 of Selector H'1238 are output (= 170 bytes) .

(3) Starting from address ABC0:EF, 70 bytes are output in binary notation.

(4) Starting from the address defined by &MARKE, bytes are output in the implied length, in the form of a character string. Non-printing characters (< H'20 or > H'7E) are replaced by '.' The job identifier 1717 is output before trace data (address = contents)

(5) Output of the value of DEBUG variable %B

(6) Output of 32 bytes of the stack contents

(7) Output of the register assignment

(8) Outputting a call parameter in length 5. It is assumed that the parameter is defined as LOC parameter, therefore dereferencing takes place.

(9) Two memory areas are output, but in reverse order: the last area specified is output first, the first area specified is output last. This is the case with all display commands for which several areas have been specified.

Examples (6),(7) and (8) are only possible at the trigger point or in interactive mode if the processor has been stopped at the trigger point.

Examples (with output):

```
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 Command (Trace Memory Contents)

This statement is the same as command D (**see above**).

3.40 SET Command (Modify Memory Contents)

SET modifies memory contents and DEBUG variables. The memory address can be representaed numerically or with the aid of DEF symbols and DEBUG variables. A maximum of 240 bytes per SET instruction can be modified. A job identifier, which is output together with the SET acknowledgement, can be specified if desired. This allows the user to identify a SET by means of a code of his choice, without having to check the address. This job identifier is output before the user data (address = contents). If no job identifier is specified, 0000 will be output. If the job identifier is #FFFF, only the user data will be output.

Examples:

```
(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;
```

Explanations:

(1) DEBUG variable %B is assigned value TRUE,

(2) variable %ABC is assigned printable character 'A' (ABCD being the order identifier) and

(3) variable %ZAHL is assigned value 101.

(4) The area addressed by &BEREICH is transferred in length (&ENDE-&ANFANG+1) to the location addressed by &ANFANG.

(5) A 5 is written in location H'18:H'34.

(6) The area addressed on the right is transferred with length 11 to memory location H'20:H'45. The contents of H'20:H'88 are interpreted as a pointer and incremented by 4 to represent the address of the area to be transferred.

(7) This example is identical with example (4), except that in this case addresses are indicated numerically.

(8) Address &ADDR is assigned value H'FC, and address &ADDR+1 is assigned value H'12; 2 bytes are assigned as a word (applies to length 2 only).

(9) Address H'AB0:H'CD is assigned value H'12, address H'AB0:H'CD+1 is assigned value H'34, and address AB0:CD+2 is assigned value H'56. Hence, transmission is byte-by-byte.

(10) At address FE0:C, value H'90 is entered with length 9. If only one byte is provided on the right, the length is interpreted as a repetition factor.

(11) Logging of ACT operations is deactivated.

(12) Port 1404 is set to 0 (watchdog is deactivated).

3.41 VIEW Command (Output of Object Information)

The VIEW command is used to list information on objects and catalogs.

Objectes	Catalogs
JOB	MODCATI
TASK	MODCATA
SEGMENT	SYSCATI
SEMAPHOR	SYSCATA
MAILBOX	
QUEUE	
POOL	
REGION	

The number of objects involved can be limited to objects belonging to a specified object (e.g. all tasks that belong to a job).

Examples:

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

3.42 INSP Command (Output of JOB and TASK Information)

The INSP command is used to obtain comprehensive information about an object. There is a simplified syntax for the user's own task and job. Otherwise the object's token must be specified. This can be done directly (with #) or indirectly by specifying an address where the token may be found. Then the object type and the specific data for this object are output.
The following objects are possible: JOB, TASK, MAILBOX, POOL, BUFFER, SEGMENT, QUEUE, REGION and SEMAPHORE.

Examples:

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

3.43 IF Command (Define Conditions)

IF makes the execution of operations dependent on the conditions defined. These conditions permit any kind of arithmetic and logic operations with DEBUG variables, constants and memory contents (Length 1, 2 or 4). The first prompt character (system status) is set to '-' within IF. IF can have seven nesting levels. The syntax (IF, THEN, ELSE, FI) is identical with that of CHILL. Example:

```
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;
```

Empty ELSE branches may be omitted.

Relational operators for defining conditions:

EQ	equal	(=)
NE	not equal	(/=)
GT	greater than or equal	(>)
GE	greater than or equal	oder gleich (>=)
LT	less than	(<)
LE	less than or equal	(<=)
AND	and	
OR	or	
NOT	not	
XOR	exclusive OR	

3.44 ! Command (Execute Instructions)

When an exclamation mark is entered, all instructions which were previously only closed with ’;’ will be executed.
An instruction that is closed with ’!’ instead of ’;’ will be executed immediately.
For instructions within a trigger definition or a macro definition, this instruction is ignored. When the table storing the instructions is full, the instructions are carried out automatically upon receipt of a corresponding message.

The number of commands that may be chained depends on the number of operations stored. However, instructions can be input at any time. Chaining is discontinued (i.e. instructions are carried out), when an AT or MDEF instruction is given. Example:

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

or, as distinguished from the above

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

In the first case, the D statements are executed jointly, in the second case, they are executed immediately after each entry.

3.45 WRITE Command (Write Comments)

WRITE can be used to write comments in the trace buffer with length 1-80, in order to protocol entries.

Example:

```
WRITE ’This is a comment’;
```

3.46 END Command (Terminate Interactive Level)

This command initiates transition to the control level. For transition to the control level, the SAVEON status is set to SAVEOFF. This function is now only required when a PRINT command longer than 79 characters is to be entered.

4 Adressing

4.1 Numerical Addressing Carried out with the Aid of DEF Symbols

An address in the 80286 processor chip is composed of

- the selector

and

- an offset.

A numerical address can be assigned to a symbol. This symbol can be used in all DEBUG commands, in replacement of a numerical address.

4.2 Addressing CHILL Variables

4.2.1 Basic Data Types

Basic data types comprise:

```
addr      numerical address or DEF symbol
distance,
length,
index     constant, DEBUG variable or arithmetic expression
```

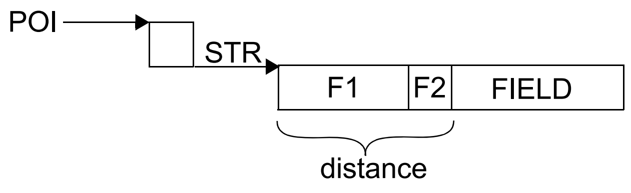
addressed object	CHILL syntax	DEBUG syntax
Variable	VAR	addr
Pointer	POI ->	addr ->
Structural component	STR.FIELD	addr+distance
Array element	ARY(INDEX)	addr+index*length
Chained pointer	POI -> -> -> ->	addr -> -> -> ->
Field of an element of an array of several structures	ARY(INDEX).FIELD	addr+index*length + distance
Element of an array of one structure	STR.ARY(INDEX)	addr+distance +
of a structure		index * length

Table 6 Overview of basic data types

4.2.2 Combinations

Basic data types, addressed by pointers

Basic data type	CHILL syntax	DEBUG syntax
3	POI → .STR.FIELD	addr → distance



Basic data type	CHILL syntax	DEBUG syntax
4	POI → .ADY(INDEX)	addr → length*index

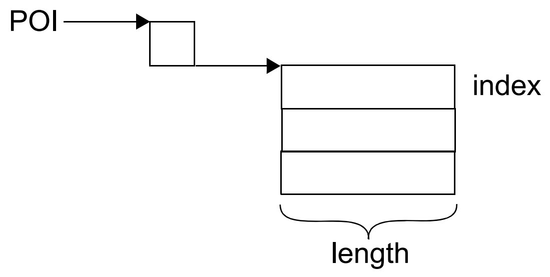


Figure 2 Basic Data Types 3 and 4 Addressed by Pointers

Dereferencing of basic data types

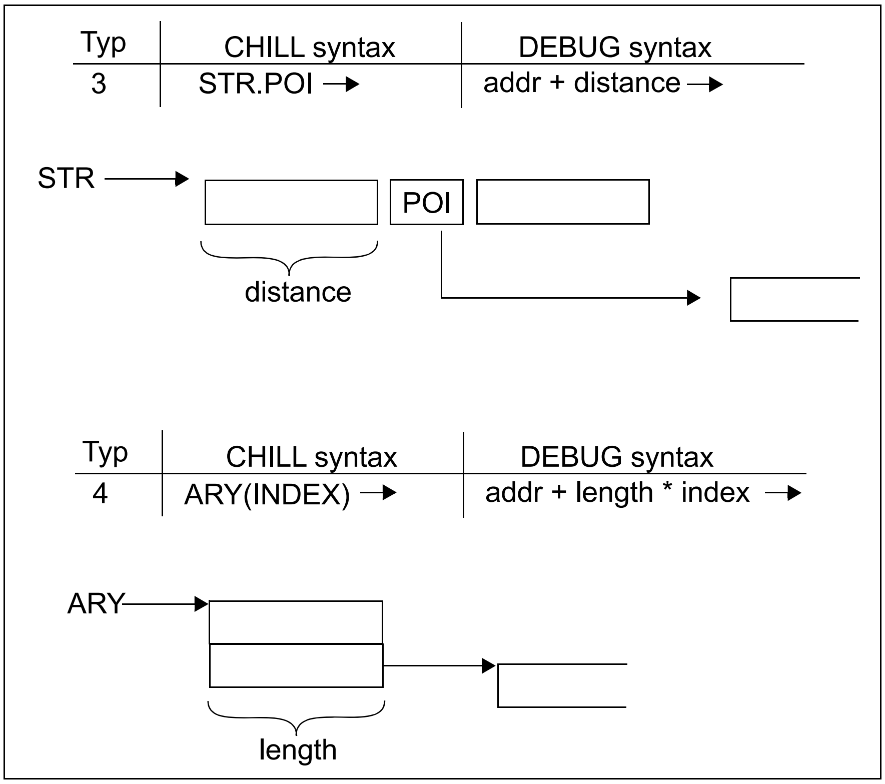


Figure 3 Basic data types 3 and 4 dereferencing

IMPORTANT: The expression used to modify an address in DEBUG is evaluated from right to left; a dereferencing arrow ('->') is interpreted as a parenthesis.

5 Special Applications

5.1 Testing with Dynamic Variables

Dynamic data can be addressed with DEBUG statements at the trigger point. The local variables of the procedure in which the trigger point is located, and of all surrounding procedures as well as their parameters, can be addressed. Instead of selector and offset, the address specifies nesting levels (from Chill compiler listing). P* marks the current procedure, Pn (n = B to Z) the static depth, where B is the highest level.

Example:

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

5.2 Testing with Processor Stop

Here, differences with the corresponding ICE statements are shown.

GOTIL has an additive effect, the previous GOTIL statements are retained as breakpoints. They can be deleted with the REMOVE statement.

DEBUG stops before executing the statement. The address of the following command is not available.

The processor is stopped by the fact that all tasks are suspended. The procedures not implemented as tasks continue running. The tasks of DEBUG and of the Miniterminal Handler (MTH) are not suspended.

5.3 Global example

Messages issued by DEBUG are not considered in this example. However, the prompt which is used to request the next entry is shown here. The numbers serve as references for the explanations.

```
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
```

Explanations of the above example:

ad 1) The DEBUG session is started. Simultaneous transition to interactive level.

ad 2) The PROTFILE/001 file is assigned as a log file.

ad 3) Entry of statements:

DEBUG variables %COUNTER with INT mode and %DEACT with %BOOL mode are defined.

The OPL trigger point is defined.
When this trigger point has been defined, %COUNTER is incremented by 1 after each run, as long as it is smaller than 13. Otherwise, the OPL trigger is deactivated and variable %DEACT is set to TRUE.

The OPL trigger is activated.

Symbol &BUFFER is defined with the buffer start address and length 16.

The contents of the memory defined by the DEF symbol are output.

ad 4) Printout of the trace buffer. Prior to printout, the statements that have not yet been executed will be processed.

ad 5) Entry of statements:

Symbol &MESGEN is defined.

Trigger AAA is defined. The area addressed by &BUFFER is written in the length implicitly defined by &BUFFER (1).

Trigger AAA is activated.

The memory area addressed by &MESGEN is read out in binary form. (length is explicitly defined as 3)

Trigger BBB is defined; BBB is to deactivate trigger AAA.

Trigger AAA is activated. Should it still be active, this ACT statement will be rejected.

Trigger BBB is activated.

7 bytes are output as a character string.

AAA is reactivated.

Triggers AAA and BBB are deactivated.

ad 6) Of the trace buffer entries, only the trigger information is to be output.

ad 7) Entry of statements:

Symbols &ONE and &TWO are defined.

Symbol &THREE is defined in relation to &TWO (offset+2), the length of &ONE is adopted.

The address range from &ONE to &TWO is set to hexadecimal 90 (16 bytes).

Value 144 is written in the memory that is addressed with symbol &THREE.

The contents of variables %COUNTER and %DEACT are output.

ad 8) Statements are read in from the command file RUNFILE1/003.

ad 9) The entire trace buffer is printed out.

ad 10) The DEBUG session is terminated.

ad 11) The entire log file is printed out.

6 Messages

6.1 DEBUG Message Classes

During a DEBUG session, a number of situations arise which DEBUG must report to the user. DEBUG groups the messages into six classes according to event types.

Error messages: (E)

This message class is used by DEBUG to report errors detected during internal tests, especially inconsistent table contents. Such errors can usually be corrected by starting the DEBUG session over again (i.e. TERM and START).

Syntax messages: (S)

This message class is used to report all syntax errors during command input.

Test order messages: (T)

This message class is used by DEBUG to report all events that occur while commands are being executed. User data is represented as binary numbers or characters.

Test order messages: (H)

Identical with class T. Hexadecimal notation of user data.

Information messages: (N)

This class comprises the message numbers which are not assigned long texts.

Information messages: (I)

This class comprises all remaining messages (information).

DEBUG messages can, in general, be written in the trace buffer and/or output to the terminal. The following table shows where messages will be transmitted to. For classes E and I there may be occasional variations.

Message class	Output to trace buffer	Output to DEBUG terminal
Error (E)	yes	yes
Syntax (S)	no	yes
Test order (T)	yes	no
Test order (H)	yes	no
Information (I)	no	yes

Table 7 Message Classes

Output to terminal: instructions only (no operations).

Output of instructions to the trace buffer is controlled by the standard DEF symbol &WRITE_IN (see function DEF).

Output of operations to the trace buffer is controlled by several standard DEF symbols (see function DEF).

6.2 DEBUG Message Format

A DEBUG message comprises a fixed and a variable component. The variable component is omitted in a number of messages. The fixed message component is omitted only in D and SET with order identifiers #FFFF, and in WRITE. The fixed variable components of a message are output on separate lines.

6.3 Fixed Message Component Format

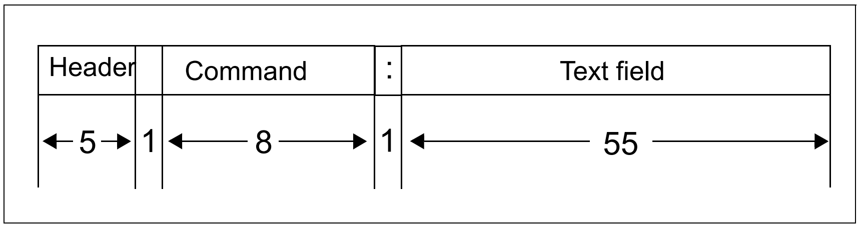


Figure 4 Fixed Message Component

Header format

The message header can be considered a message abbreviation.

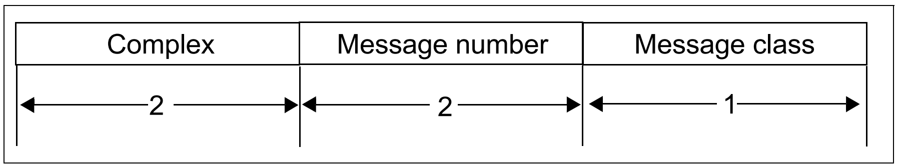


Figure 5 Message Header

Explanation:

Complex

The complex serves for localizing the DEBUG complex that initiates the message and for expanding the message number range.

Possible values:

00	syntax analyzer
01	syntax analyzer
03	command analyzer
04	special commands
05	interpreter
06	execution routines
08	system object inspection

Message number

The message number, a unique number between 1 and 99, is assigned for each complex.

Message classes

The following message classes exist:

E	Error message (software error)
S	Syntax message
T	Test order message
I	Information message
N	Message text not available
H	As for class T , if hexadecimal notation

Command

In some messages the command keyword is repeated.

Text field

Explanatory message text. This text is always issued in addition to the message.This service manual lists all the messages including the text field. No additional explanations are given.

Additional Information (variable part of the message) **see the chapter on setup of the additional information level.**

7 Syntax Error

7.1 Syntax Error Handling with Special Commands

The following tests are performed:

- A check is made to see whether the command exists and is permitted at that point in the session.
- In addition, a check is made to see whether the parameters exist, whether all required parameters have been specified and whether the parameter values are within limits.

In the event of an error, a message is output. However, the parameter in which the error was made is not indicated.

7.2 Syntax Error Handling for Normal Commands

Syntax error handling is based on the following principle: to facilitate correction of faulty input, a line-oriented, instruction-related syntax check is carried out. This method prevents an input of linked instructions from having to be re-entered in full.

From the user’s point of view, the error handling procedure is as follows:

Having entered one line, the user receives a message telling him whether or not this line was correct. If it was, the prompt, e.g. ’**>’, will appear. If an error is reported, the user only needs to re-enter his command from the faulty instruction onwards.

Example:

```
**>           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 the last line, an error is reported to the user. In this example, the user must enter his correction as follows:

```
D 1238:H’745(4);SET %B=1;OD;
```

Note: Commands ’AT’ and ’IF’ are exceptional, since these commands may contain one or several other instructions. Besides, commends ’THEN’ and ’ELSE’ may in their turn contain further instructions. For syntax testing, such nested instructions are converted into a succession of linear instructions.

Example: the AT statement

```
’LIN: AT xy DO D z;D v;OD;’
```

is composed of 4 statements:

```
’LIN: AT xy DO’,
’D z;’,
’D v;’ and
’OD;’.
```

Thus, the syntax error handling principle is applied to each of the commands seperately.

7.3 Syntax Error Handling in Runfiles

Statements in a runfile are checked exactly as with interactive input. The first wrong line will terminate the read-in of a command file. The user is told which kind of error occurred in which line.

As opposed to error handling in dialog mode, the instructions cannot be corrected immediately. Wrong nested instructions are terminated by DEBUG itself with the necessary number of ’FI’ and ’OD’. The commands are not logged at the terminal.

In order to avoid interupts due to errors in a runfile, parameter FUNC=CHECK can be declared in for the RUN command. This parameter must only be declared if the user is certain that the file contains no errors which are likely to start the DEBUG off on an undesired sub-procedure. For instance, if the trigger point definition is wrong, the insstructions for this trigger are no longer interpreted as operations but considered instructions and executed immediately.

8 Syntax in BNF

Let	`@`	be the empty symbol,
	`A`	a non-terminal,
	`X,Y,Z`	concatenations of terminals, non-terminals and some symbols (explained next).

The following list shows the meanings of the used symbols, which are similar to the BNF (Bacchus-Naur-Form):

	`A ::= X`	is a production (rewriting rule) for the non-terminal A. It defines the way in which A is built up from other terminals and/or non-terminals.
	`"a"`	a is a terminal.
	`X | Y`	means the alternate choice between X and Y. A ::= X | Y is short for A ::= X oder A ::= Y .
	`()`	many contain a set of symbols or may be used for reasons of clarity.
i.e.	`(X | Y) Z`	s short for X Z | Y Z .
	`[X]`	is defined as @ | X .
	`{X}`	is defined as @ | X | X X | X X X | ...
and		
	`--comment`	for comments.

The syntax:

--first the command level (highest level)

```
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 and
             --  max. 160 characters per line,,
                              --  on special commandsd ";" and "!" are optional
                              --  saveoff_cmd and saveon_cmd are allowed
                              --  as special command or instructionoperation
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 already defined
                        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"] not in 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         -- no with
                                                           io_port
                            |            bool_const        -- no with
                                                           io_port
                            |            "’" char "’"      -- no with
                                                           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 characters
index             ::=     number
write_cmd         ::=     string                -- 1 to 80 characters
```

-- the next level contains symbols used more than once --

```
at_label          ::=     name       -- max. 8 alphanumeric characters
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"]
                           -- "," missed if this parameter first
label             ::=      "*" | letter {letter | dec_digit} ["*"]
```

```
                          -- max. 8 characters
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]
trigger         ::=       (def_symbol | num_addr_abs) [("+" | "-") offset]
```

-- the auxiliary level (symbols used in previous levels)

```
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} "’"
                            -- ’ in string duplicate
symb_addr           ::=     def_symbol [addr_mode]
symb_addr_rng       ::=     def_symbol ["TO" def_symbol | [addr_mode] length]
```

-- the lowest level basic symbols, which are used frequently

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

```
-- Names ...
bool_var     ::=  name
debug_var    ::=  "%" name
def_symbol   ::=  "&" name
filename     ::=  name        -- first character must be alphabetic
                              -- max. 27 alphanumeric characters
macname      ::=  name        -- max. 8 alphanumeric characters
name         ::=  letter {letter | dec_digit | "_" | "."}

-- Numbers ...
length           ::=   "(" (number | register) ")"
                       if used in condition, only 1, 2 or 4 bytes allowed
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

-- Printable characters ...
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 User Information

9.1 General

Wherever there is at least one BLANK (’ ’), it may be replaced by any number of BLANKs and/or comments. A comment can take up any number of lines and is delimited by ’/*....

9.2 Generation Limit Values

The maximum values that apply when entering commands with DEBUG are listed below. The values can be changed only by re-writing the translating program.

Meaning	Value
Number of triggers	170
Number of macros	34
Number of standard DEF symbols	12
Number of definable DEF symbols	244
Number of DEBUG variables	100
Number of operations (sum total)	750
Number of operation expressions	510
Number of linkable instructions	34
Number of instruction expressions	68
Length of DEBUG variable names	26
Length of DEF symbol names	26
Length of trigger names	8
Length of macro names	8
Length of file names	32
Size of trace buffer (bytes)	17000
Size of log file	10000
Number of instruction characters	260
Number of operation characters	1000
Hexadecimals in instructions	400
Hexadecimals in operations	1700

Table 8 Generation Values

10 List of DEBUG Messages

All of the messages listed are messages addressed to the user. They are arranged according to complexes and message numbers.

The auxiliary data is described in the next section, sorted according to message numbers.

All messages numbered 00xxS and 01xxS include, as additional information, the last input line up to the faulty character string.

The text is shown in German with the English version below. Messages are output in one language only.

/***** 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 TAB=MAC	:MAKROTABELLE: :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 D	:TESTPUNKT DEAKTIVIERT :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

List of Figures

List of Tables

Index

mitel.com