1 Preface

The **ASSEMBH** assembler is available as two separate software packages (selectable units) with different levels of functionality.

ASSEMBH

- assembly of assembler source programs into object modules or link-and-load modules
- availability of the standalone listing generator ASSLG in addition to output of listings in standard format
- support for structured programming, i.e. enhancements including macros for structured programming and listing generator programs for Nassi-Shneiderman diagrams and structure lists
- ILCS interface for structured programming
- symbolic debugging of assembler programs by the creation of LSD records for the Advanced Interactive Debugger AID
- support for the ASSEMBH diagnostic routine ASSDIAG
- output of structured lists when using the structured programming macros in ASSEMBH
- support for the ESA instructions

ASSEMBH-BC

The ASSEMBH-BC assembler is the ASSEMBH basic configuration with reduced functionality.

- assembly of assembler source programs into object modules or link-and-load modules
- output of listings in standard format

1.1 Brief product description

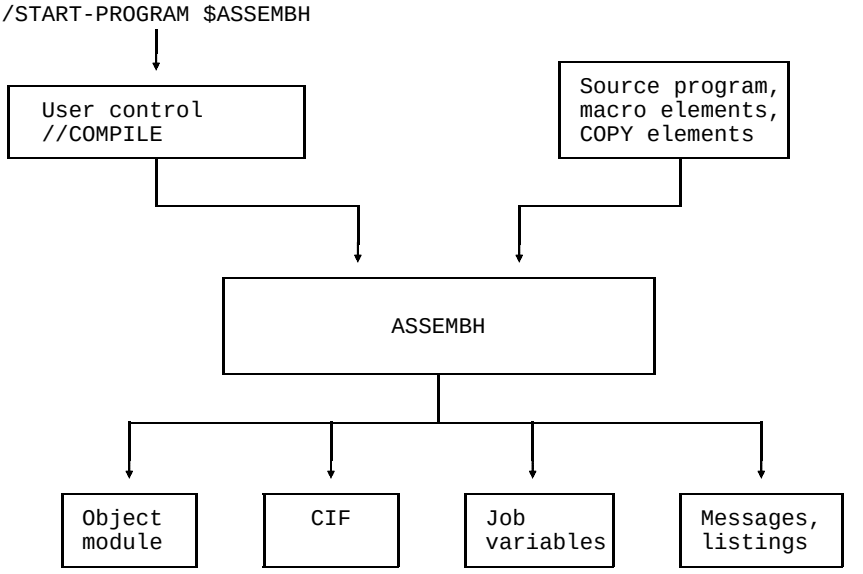
ASSEMBH is a two-pass assembler. The structure of the assembler determines how a source program is processed. This results in a few incompatibilities with ASSEMB V30 (see section 2.4.3, COMPILER-ACTION option).
Listings are generated from internal log information, i.e. from the **C**ompiler **I**nformation **F**ile (**CIF**).

Functions during the first pass of the assembler
All instructions in the source program, including any COPY elements and macro elements (see "ASSEMBH (BS2000) Reference Manual" [1]), are read in, subjected to a syntax analysis, and converted into an intermediate language by completing all required macro processing steps. Listing information related to the source program is stored.

Functions during the second pass of the assembler
The object module is generated from the intermediate language, and object-related listing information is stored.

Standard listing generator
The standard listing generator creates listings from the internally logged information.

Overview of the data flow of ASSEMBH
The input and output options of ASSEMBH are discussed in chapter 3.



General data flow of ASSEMBH

1.2 Target group

This manual is intended for users wishing to create, use or maintain assembly or macro language programs in BS2000. A basic knowledge of the operating system is required.

1.3 Summary of contents

This manual describes the use of ASSEMBH in BS2000.
Chapters 2 and 3 deal with assembly using ASSEMBH;
chapters 4 and 10 describe the support offered for structured programming;
chapters 5 and 6 explain linking, loading and starting, and the associated listings;
chapter 7 explains how programs written in other programming languages can be interfaced with ASSEMBH programs;
chapters 8 and 9 describe diagnostic and debugging aids, and
the appendix contains the ASSEMBH messages, an overview of the format of the assembler instructions and a comparison of *COMOPT and COMPILE statements.

The assembly and macro language for the ASSEMBH assembler is described in the "ASSEMBH (BS2000) Reference Manual" [1].

1.4 Changes since the last version of the manual

Various corrections have been made throughout the manual and are not listed separately.
The most important technical developments and changes are as follows:

The SOURCE-FORMAT=STRUCTURED operand in the LISTING option (see section 2.4.4) is used to generate structured lists (to do this, you must use the structured programming macros).
The GENERATE statement (see section 2.5) also allows the standalone list generator to generate structured lists.

The structured list generated using ASSEMBH is described in section 6.5.

The LISTING option has a new operand NOPRINT-PREFIX (see section 2.4.4).

The ESA instruction set is generated using the INSTRUCTION-SET=BS2000-ESA operand of the SOURCE-PROPERTIES option (see section 2.4.1.4). For details on ESA support, see section 5.8.

A module in LLM format is generated using the MODULE-FORMAT=LLM operand of the COMPILER-ACTION option (see section 2.4.2.1). See sections 3.2 and 6.6 for additional information.

The YES operand has been replaced by the AID operand in the TEST-SUPPORT option (see section 2.4.5).

Chapter 5 on "Linking, loading and starting" has been reworked and restructured and now includes a new section (section 5.2) on "Linking with BINDER".

The list of machine instructions (see the appendix, section 11.3) now includes the ESA instructions.

Any functional changes and additions to the current product version can be found in the chapter "Manual supplements".

1.5 Notational conventions

The following notational conventions (metalanguage) are used to represent the format of BS2000 commands and program instructions in this User Guide:

*STD	Uppercase letters, digits, and special characters that are not part of the metalanguage denote keywords or constants, all of which must be specified in the given form.
name	Lowercase letters denote variables, which must be replaced by current values during input.
YES NO	Underlining is used to identify the default value that is automatically inserted if no explicit specification is made.
⌈YES⌉ { } ⌊NO ⌋	Braces enclose alternatives. One of the indicated values must be selected. If the underlined default value is desired, no entry is required.
YES / NO	A slash between adjacent entries also indicates alternatives from which one value must be selected. No specification is needed if the indicated default value is desired.
[ ]	Square brackets enclose optional specifications which may be omitted.
( )	Parentheses are constants and must be specified.
␣	This character indicates that the entry of at least one space character is essential in the syntax.
...	Ellipses are used to indicate that the preceding unit may be repeated more than once.
[,...]	A comma followed by ellipses means that the preceding unit may be repeated more than once, but must be separated by a comma in each case. The square brackets indicate that the specification is optional.

Note

A different metasyntax is used for the SDF interface (see section 2.3.2).

2 Assembly

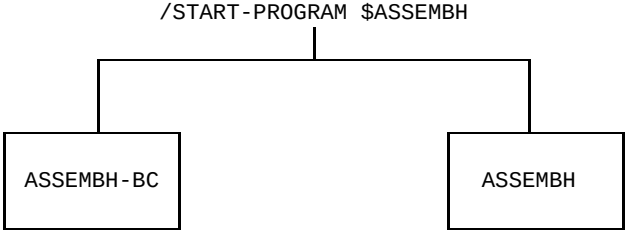
2.1 Calling ASSEMBH

The assembler processes one source program at a time. A source program consists of a sequence of instructions (assembler instruction statements, machine instructions, macro calls and macro instruction statements) and remarks. Each source program may be made up of one or more assembly units. Individual assembly units in the source program are separated from one another by means of END instructions. A separate object module is generated by the assembler for each assembly unit.

The assembler run is controlled by user options as well as the assembly language instruction statements in the source program.

Once the assembler has been started, the options are read in and evaluated via the SDF interface (see "Introductory Guide to the SDF Dialog Interface, User Guide" [5] and "BS2000/OSD-BC Commands" [6]).
*COMOPT control options continue to be supported (see sections 11.4 and 11.5) for compatibility reasons.

The **ASSEMBH-BC** and **ASSEMBH** assemblers are started via the SDF command interface as follows:



2.2 Controlling ASSEMBH

2.2.1 Simple assembly

Simple assembly means:
an assembler run with one source program containing a single assembly unit.

From the time the assembler is started up to its termination, the assembler is controlled by options and the contents of the source program. Options are entered via the SDF statements (//COMPILE...) of SYSSTMT. //END terminates the assembler run.

The following statements are required for an assembler run:

```
/START-PROG $ASSEMBH
//COMPILE...
//END
```

2.2.2 Multiple assembly

Multiple assembly means:
an assembler run with a source program containing more than one assembly unit.

In the case of a multiple assembly, no options are read between the processing of assembly units, i.e. all assembly units of the source program are assembled with exactly the same options from start to finish.

The following statements are required for a multiple assembly:

```
/START-PROG $ASSEMBH
//COMPILE...
//END
```

The source program specified via the COMPILE statement must have the following format:

```
START
.
.               1st assembly unit
.
END
START
.
.               2nd assembly unit
.
END
.
.
```

Note

If the source program as well as the options are read in from SYSDTA, the assembler is restarted instead of a multiple assembly being performed. Use of the //END instruction allows the assembler to be terminated after each assembly unit without the generation of an EOF.

2.2.3 Restarting the assembler

Restart means:
that several source programs can be assembled in succession with a single call to the assembler.

Unless they are explicitly overwritten, all option settings from the preceding assembly (except those for the source program and the output of the object module) remain in effect. This is achieved by using the restart function of the assembler, which is activated by means of the following control statements:

```
/START-PROG $ASSEMBH
//COMPILE SOURCE=...
//COMPILE SOURCE=...
   .
   .
   .
//COMPILE SOURCE=...
//COMPILE SOURCE=...
//END
```

2.3 SDF interface of ASSEMBH

ASSEMBH is controlled via the SDF interface, i.e. options must be entered in SDF format. Commands given on the operating system level may be specified in the earlier ISP format as well as in SDF format.

SDF provides the following options in interactive mode:

- input from the data terminal with user guidance at three different levels, hereafter called the "menu mode";
- input from the data terminal without user guidance in two different forms, hereafter called the "expert mode";
- input from a procedure file

The user can temporarily switch from expert mode to menu mode. When working in menu mode with medium or minimum guidance, a temporary switch to the next-higher guidance level is also possible.
Inputs from a procedure file are effected in expert mode.
In addition to a temporary switch from one SDF mode to another, it is also possible to switch modes permanently. This is achieved by means of the following SDF command (or the SDF statement //MOD-SDF-OPT; see section 2.3):

/MODIFY-SDF-OPTIONS
GUIDANCE=UNCHANGED/EXPERT/NO/MAXIMUM/MEDIUM/MINIMUM

The meaning of each setting is given below:

UNCHANGED
: The existing (default) setting applies.

EXPERT
: Expert mode; the system prompts for the input of commands and statements with "/" and "//", respectively; no interactive syntax checking; detailed error messages; blocked command input. This mode is set by default following the LOGON command.

NO
: Expert mode; the system prompts for the input of commands and statements with "%CMD:" and "%STMT", respectively; interactive syntax checking (i.e. correction of incorrect inputs without repeating the entire command); blocked command input (multiple commands separated by logical end-of-line characters can be issued simultaneously).

MAXIMUM
: Menu mode; maximum help level, i.e. all operand values with options; help texts for commands and operands.

MEDIUM
: Menu mode; all operand values without options; help texts for commands only.

MINIMUM
: Menu mode; minimum help level, i.e. only default values of operands; no options; no help texts.

2.3.1 Processing the operand form

After starting ASSEMBH, the user can respond to the statement prompt by entering a '?' or specifying //MOD-SDF-OPT GUIDANCE=MAX in order to call up the operand form of the COMPILE statement, where each operand is queried individually.

```
PROGRAM  : ASSEMBH                        STATEMENT: COMPILE


SOURCE                = *SYSDTA
MACRO-LIBRARY         = *NONE
COPY-LIBRARY          = *NONE
SOURCE-PROPERTIES     = STD
COMPILER-ACTION       = MODULE-GENERATION(MODE=STD,MODULE-FORMAT=OM)
MODULE-LIBRARY        = *OMF
COMPILATION-INFO      = NONE
LISTING               = STD
TEST-SUPPORT          = NO
COMPILER-TERMINATION = STD
CORRECTION-CYCLE      = NO
COMPILATION-SPACE     = STD
NEXT = *CONTINUE
*EXECUTE"F3" or + or Next-stmt or *CANCEL"K1"
```

Some of the important points to be noted in connection with processing the operand form are summarized below. A detailed description on the use of SDF can be found in the manual "Introductory Guide to the SDF Dialog Interface, User Guide" [5].

Special inputs

?	as an operand value returns help texts and an indication of the value range for this operand. If the message "CORRECT INCORRECT OPERANDS" was issued by SDF following a preceding incorrect entry, the question mark returns additional and more detailed error messages. The remainder of the line need not be deleted.
!	as an operand value resets the default value for the operand in question, assuming the displayed default value was overwritten earlier. The remainder of the line need not be deleted.
<operand>(	An opening parenthesis after an operand that begins a structure outputs the sub-form for the associated structure. Operands specified after the opening parenthesis are shown in the sub-form.
–	as the last character in an input line causes a continuation line to be output (see example in section 5.6; up to 9 continuation lines are possible per operand).
LZF key	deletes all characters in the input line as of the cursor.

Function keys

K1	terminates the current operand form and switches to the one above it in the hierarchy. Equivalent to *CANCEL in the NEXT line.
K2	interrupts a running program (e.g. the assembler) or a running procedure.
K3	repeats the operand form that was last output. Equivalent to *RESTORE in the NEXT line.
F2	checks inputs for syntax errors. Equivalent to *TEST in the NEXT line.
F3	executes the current operation. Equivalent to *EXECUTE in the NEXT line.

NEXT line

Below the NEXT line of each menu page is a list of possible specifications. The concepts are either self-explanatory or explained in the manual "Introductory Guide to the SDF Dialog Interface, User Guide" [5].

+, –	pages forward and backward in the operand form.
++, --	opens the first or last page of the operand form.
*EXECUTE	executes the current operation. Equivalent to the F3 key.
*CONTINUE	pages forward in the form if the end of the form has not been reached. Otherwise, the current operation is executed.
*TEST	checks inputs for syntax errors. Equivalent to the F2 key.
*CANCEL	terminates the current form and switches to the one above it in the hierarchy. Equivalent to the K1 key.
*RESTORE	repeats the form that was last displayed. Equivalent to the K3 key.
<statement>?	executes the current operation and then outputs the operand form for the specified <statement>. Operand values that have already been specified are transferred to the operand form.
<statement>	executes the current operation and then the specified <statement> as well. If no operand values are explicitly specified, the predefined operand values are carried over.
?	switches to the next (higher) help text level for the current input.

*DOWN(<operand>)
displays the sub-form for the specified <operand>, which is defined in a structure.

*UP switches from the sub-form back to the operand form that precedes it in the hierarchy.

Example

The example below illustrates how the operand form of the COMPILE statement is processed. A source program named test1, which resides as an element with version number 6 in the PLAM library plamlib, is to be assembled.
The assembled program, the object module, and the assembler listing are to be output to the PLAM library plamlib.

The name and location of the source program are specified with the SOURCE option; the location of the object module is specified with the MODULE-LIBRARY option, and the location of the assembler listing with the LISTING option.
The options of the COMPILE statement are described in section 2.4.

ASSEMBH is started. When the statement prompt (//) appears, entering a question mark causes the operand form for the COMPILE statement to be displayed.

```
/START-PROG $ASSEMBH
%  BLS0500 PROGRAM 'ASSEMBH', VERSION 'V1.xxxx' OF 'yyyy-mm-dd' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1991. ALL
   RIGHTS RESERVED
%  ASS6010 V 1.xxxx OF BS2000 SIEMENS ASSEMBH READY
%//?
```

```
PROGRAM  : ASSEMBH                        STATEMENT: COMPILE


------------------------------------------------------------------------
SOURCE                = *SYSDTA
MACRO-LIBRARY         = *NONE
COPY-LIBRARY          = *NONE
SOURCE-PROPERTIES     = STD
COMPILER-ACTION       = MODULE-GENERATION(MODE=STD,MODULE-FORMAT=OM)
MODULE-LIBRARY        = *OMF
COMPILATION-INFO      = NONE
LISTING               = STD
TEST-SUPPORT          = NO
COMPILER-TERMINATION  = STD
CORRECTION-CYCLE      = NO
COMPILATION-SPACE     = STD
------------------------------------------------------------------------
NEXT = *CONTINUE
       *EXECUTE"F3" or + or Next-stmt or *CANCEL"K1"
```

The possible operands for each option can be queried. For example, we could enter a question mark for the SOURCE and LISTING options:

```
PROGRAM  : ASSEMBH                        STATEMENT: COMPILE


SOURCE                = ?SYSDTA
MACRO-LIBRARY         = *NONE
COPY-LIBRARY          = *NONE
SOURCE-PROPERTIES     = STD
COMPILER-ACTION       = MODULE-GENERATION(MODE=STD,MODULE-FORMAT=OM)
MODULE-LIBRARY        = *OMF
COMPILATION-INFO      = NONE
LISTING               = ?TD
TEST-SUPPORT          = NO
COMPILER-TERMINATION = STD
CORRECTION-CYCLE      = NO
COMPILATION-SPACE     = STD
------------------------------------------------------------------------------
NEXT = *CONTINUE
       *EXECUTE"F3" or + or Next-stmt or *CANCEL"K1"
```

The possible operands are output:

```
PROGRAM  : ASSEMBH                        STATEMENT: COMPILE
OPERANDS : SOURCE=*SYSDTA,LISTING=STD

------------------------------------------------------------------------------
SOURCE                = *SYSDTA
                        *SYSDTA or full-filename_1..54 or *LIBRARY-ELEMENT(LIBRAR
                        Y=?,ELEMENT=?)
                        specification of the file containing the source
MACRO-LIBRARY         = *NONE
COPY-LIBRARY          = *NONE
SOURCE-PROPERTIES     = STD
COMPILER-ACTION       = MODULE-GENERATION(MODE=STD,MODULE-FORMAT=OM)
MODULE-LIBRARY        = *OMF
COMPILATION-INFO      = NONE
LISTING               = STD
                        STD or PARAMETERS()
                        selection of size and structure of the standard listing
TEST-SUPPORT          = NO
COMPILER-TERMINATION = STD
CORRECTION-CYCLE      = NO
COMPILATION-SPACE     = STD
------------------------------------------------------------------------------
NEXT = *CONTINUE
       *EXECUTE"F3" or + or Next-stmt or *CANCEL"K1"
```

We now enter the operand values for the following options:
SOURCE: library name plamlib and element name test1 with version 6
MODULE-LIBRARY: library name plamlib
LISTING: library name plamlib

```
PROGRAM  : ASSEMBH                        STATEMENT: COMPILE
OPERANDS : SOURCE=*SYSDTA,LISTING=STD

-------------------------------------------------------------------------------
SOURCE               = (plamlib,test1(6))
                       *SYSDTA or full-filename_1..54 or *LIBRARY-ELEMENT(LIBRAR
                       Y=?,ELEMENT=?)
                       specification of the file containing the source
MACRO-LIBRARY        = *NONE
COPY-LIBRARY         = *NONE
SOURCE-PROPERTIES    = STD
COMPILER-ACTION      = MODULE-GENERATION(MODE=STD,MODULE-FORMAT=OM)
MODULE-LIBRARY       = plamlib
COMPILATION-INFO     = NONE
LISTING              = par(output=(plamlib))
                       STD or PARAMETERS()
                       selection of size and structure of the standard listing
TEST-SUPPORT         = NO
COMPILER-TERMINATION = STD
CORRECTION-CYCLE     = NO
COMPILATION-SPACE    = STD
-------------------------------------------------------------------------------
NEXT = *CONTINUE
       *EXECUTE"F3" or + or Next-stmt or *CANCEL"K1"
```

```
%  ASS6011 ASSEMBLY TIME: 183 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT : NO ERRORS
%  ASS6006 LISTING-GENERATOR TIME : 531 MSEC
%//


Following the assembly, the assembler will once again issue a statement
prompt. The assembler run can now be terminated with the END statement.

%//END
%  ASS6012 END OF ASSEMBH
```

2.3.2 Metasyntax for the SDF interface

The format overview of the COMPILE statement (see section 2.4) is divided into two fields. The first field contains the COMPILE statement (COMPILE); the second field contains the possible options together with the operand values.

The meanings of special characters (so-called metacharacters) used in the format are explained in the table below:

Designation	Meaning	Examples
UPPERCASE	Uppercase letters denote keywords. Some keywords begin with *	LITERAL = YES SOURCE = *SYSDTA
=	The equal sign associates an operand name with the associated operand value.	MODULE-LIBRARY = *OMF
< >	Angle brackets identify variables for which value sets are defined by data types and their suffixes (see tables 2 and 3).	VERSION = <text 1..24>
Underlining	Underlining is used to indicate the default value of an operand.	LISTING = STD
/	A slash separates alternative operand values.	LASER-PRINTER = NO / ND2
(...)	Parentheses identify operand values that introduce a structure.	SYMBOL = NO / YES(...)
Indentation	Indentation shows the respective relationship with each higher-ranking operand.	SYMBOL = NO / YES(...) YES(...)

Designation	Meaning	Examples
|	The vertical bar identifies related operands of a structure. Each bar indicates the start and end of a structure, within which further structures may occur. The number of vertical bars before an operand indicates the level of the structure.	*LIBRARY-ELEMENT(...) | LIBRARY = | ,ELEMENT = | | VERSION =
,	The comma precedes other operands at the same structural level.	,LITERAL = NO / YES ,MACRO = NO / YES
list-poss(n):	A list can be constructed from the operand values that follow list-poss. If (n) is specified, a maximum of n elements may appear in the list. If the list includes more than one element, it must be enclosed in parentheses.	list-poss: <full-filename> / *LINK list-poss(256): <name 1..1>

Note

Constant operand values sometimes begin with an asterisk (*). This applies if an alternative to the constant is a data type whose character set allows the string of the constant to be specified.

Example

ELEMENT = *ALL / <name>

The value "ALL" may be inserted for the data type name. To enable differentiation, the constant operand value of the same name must therefore begin with an asterisk (*) : *ALL

2.3.2.1 Data types and suffixes

Data type	Character set	Meaning
full-filename 1..54	A - Z, 0 - 9, $, #, @, period, hyphen	Fully qualified name of a cataloged file, a PLAM library, or a library element. It is not possible to use an underscore in element names as in LMS. The first character must be a digit or letter; the last character must not constitute a hyphen or period. Furthermore, the name must not be made up of only digits or special characters. The maximum length, including the user-id and cat-id, must not exceed 54 characters.
full-filename 1..8	A - Z, 0 - 9, $, #, @, period, hyphen	Link name of a cataloged file or PLAM library. The first character must consist of a letter or digit; the last character must not be a hyphen or period. Names made up of only digits or special characters are illegal. The maximum permissible length is 8 characters.
composed-name 1..24	A - Z, 0 - 9, $, #, @, period, hyphen	Version designation of a PLAM library element. The maximum length is equal to 24 characters. The character set supported by LMS may be used.
composed-name 1..64	A - Z, 0 - 9, $, #, @, hyphen	Name of a PLAM library element.
name 1..64	A - Z, 0 - 9, $, #, @	Prefix for macro and address names. As of SDF V2.0 an underscore is also possible.
integer 2..255	0 - 9	Specifies an interval (0-32767)
c-string (character-string)	EBCDIC characters	String of EBCDIC characters enclosed in single quotes. The string may be prefixed with the letter C.

The data types can have the following suffixes:

Suffix	Meaning
1..n integer m..n	Permitted number of characters. Specifies an interval.
without -gen(eration) -vers(ion) -cat-id	 No file generation or file generation group may be specified. No element version may be specified. No catalog identification may be specified.

Note

– On '@'
  As of PLAM V1.4, '@' may no longer be specified as a version for object module output.

2.4 COMPILE statement

This statement controls the assembly of an assembler source program. It includes the following operands at the topmost structural level:

```
COMPILE
```

```
For input support:

     SOURCE =
    ,MACRO-LIBRARY =
    ,COPY-LIBRARY =
    ,SOURCE-PROPERTIES =

For module generation:

    ,COMPILER-ACTION =
    ,MODULE-LIBRARY =

For CIF support:

    ,COMPILATION-INFO =

For listing support:

    ,LISTING =

For debugging support:

    ,TEST-SUPPORT =

For terminating the assembly:

    ,COMPILER-TERMINATION =

For activating the correction cycle:

    ,CORRECTION-CYCLE =

For maintenance support:

    ,MAINTENANCE-OPTIONS =

For reducing the virtual address space requirement:

    ,COMPILATION-SPACE =
```

2.4.1 Input support options

These options define the source program to be assembled, the macro libraries of the user, and libraries for COPY elements, along with the format of the source program, the instruction set, and a value for the global system variable symbol &SYSPARM.

```
COMPILE

 SOURCE =               source program
,MACRO-LIBRARY =        user macro libraries
,COPY-LIBRARY =         libraries for COPY elements
,SOURCE-PROPERTIES =    format of the source program, instruction set,
                        value for &SYSPARM
```

2.4.1.1 SOURCE option

Function

The SOURCE option can be used to specify from where the source program is to be read. If this option is omitted, the source program is read from SYSDTA.

Format

```
COMPILE

SOURCE = *SYSDTA /
         *SYSDTA-AFTER-BREAK /
         <full-filename 1..54> /
         *LIBRARY-ELEMENT(...)

    *LIBRARY-ELEMENT(...)
        |
        |  LIBRARY = <full-filename 1..54 without-gen-vers>
        |
        | ,ELEMENT = <composed-name 1..64>(...)
        |     |
        |     |  VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT /
        |     |            <composed-name 1..24>
```

SOURCE = *SYSDTA
The source program is read from SYSDTA.

SOURCE = *SYSDTA-AFTER-BREAK
An interrupt is generated after the options are read, and SYSDTA can be assigned via SYSCMD by using the ASSIGN-SYSDTA command. The source program will then be read via SYSDTA. The new assignment of SYSDTA will, however, only take effect after all options have been processed.

SOURCE = <full-filename 1..54>
Name of a cataloged file containing the source program.

SOURCE = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..54>
Name of a PLAM library containing the source program.

ELEMENT = <composed-name 1..64>(...)
Name of an S-type (source program) element of the specified PLAM library.

VERSION = *HIGHEST-EXISTING
The element with the highest existing version is used.

VERSION = *UPPER-LIMIT
The element with the highest possible version is used.

VERSION = <composed-name 1..24>
Version designation of the element.

Notes

– On entering operands:
The input of '*LIBRARY-ELEMENT (LIBRARY=...,ELEMENT=...)' may be omitted when specifying a library element.

Example

The entry:

```
//C SOURCE=*LIBRARY-ELEMENT(LIBRARY=lib,ELEMENT=element(VERSION=007))
```

can also be written as

```
//C SOURCE=(lib,element(007))
```

– On libraries
In addition to PLAM libraries, OSM source program libraries are allowed.

2.4.1.2 MACRO-LIBRARY option

Function

The MACRO-LIBRARY option can be used to specify a maximum of 100 user-own PLAM libraries from which macro elements are to be read (PLAM library elements of type M).

Format

```
COMPILE

MACRO-LIBRARY = *NONE /
                list-poss(100): <full-filename 1..54 without-gen-vers> /
                                *LINK(...)

    *LINK(...)
        LINK-NAME = <full-filename 1..8 without-gen-vers>
```

MACRO-LIBRARY = *NONE
No user-own macro library is assigned.

MACRO-LIBRARY = list-poss(100): <full-filename 1..54>
Names of the PLAM libraries which contain the macro elements.

MACRO-LIBRARY = list-poss(100): *LINK(...)

LINK-NAME = <full-filename 1..8>)
Designates the assigned link name of a macro library.

Notes

– On the search order:
  See "Search order for macro elements" in section 3.1.2.

– On list-possible
  It is possible to mix library names and link names in a list.

  Example

  ```
  /SET-FILE-LINK LINK-NAME=maclink,FILE-NAME=maclib
  ```

  The link name maclink is assigned to the macro library maclib.

  ```
  //C MAC-LIB=(maclib1,maclib2,*LINK(maclink))
  ```

  The macro libraries maclib1 and maclib2 are assigned together with the macro library maclib, which is assigned via the link name maclink.

– On libraries
  Besides the PLAM libraries, OSM macro libraries in MLU format are allowed.

2.4.1.3 COPY-LIBRARY option

Function

The COPY-LIBRARY option can be used to specify up to 100 user-own PLAM libraries from which COPY elements are to be read (PLAM library elements of type S or M).

Format

```
COMPILE

COPY-LIBRARY = *NONE /
               list-poss(100):
               <full-filename 1..54 without gen-vers>(...) /
               *LINK(...)

    <full-filename 1..54 without gen-vers>(...)
        |
        |  ELEMENT-TYPE = SOURCE-ONLY / MACRO-ONLY / BOTH

    *LINK(...)
        |
        |  LINK-NAME = <full-filename 1..8 without-gen-vers>
        |
        | ,ELEMENT-TYPE = SOURCE-ONLY / MACRO-ONLY / BOTH
```

COPY-LIBRARY = *NONE
No user-own COPY library is assigned.

COPY-LIBRARY = list-poss(100): <full-filename 1..54>(...)
Name of the PLAM library containing the COPY elements.

ELEMENT-TYPE = SOURCE-ONLY / MACRO-ONLY / BOTH
Names the type (S, M) of the COPY elements to be read from the specified libraries (in the case of BOTH, first S, then M).

COPY-LIBRARY = *LINK(...)

LINK-NAME = list-poss(100): <full-filename 1..8>
Designates the assigned link name of a COPY library.

ELEMENT-TYPE = SOURCE-ONLY / MACRO-ONLY / BOTH
Names the type (S, M) of the COPY elements which are to be read from the specified libraries (in the case of BOTH, first S, then M).

Notes

– On the search order:
  See "Search order for COPY elements" in section 3.1.3.

– On the ELEMENT-TYPE entry
  The ELEMENT-TYPE entry is only valid for the library specified in each case.

– On list-possible
  It is possible to mix library names and link names in a list.

  Example

  ```
  /SET-FILE-LINK LINK-NAME=coplink,FILE-NAME=coplib
  ```

  The link name coplink is assigned to the library coplib.

  ```
  //C COPY-LIB=(coplib1(ELEM-TYPE=MAC-O),coplib2,*LINK(coplink))
  ```

  The libraries coplib1 and coplib2 are assigned directly; the library coplib is assigned via the link name coplink.

– On libraries
  In addition to the PLAM libraries, OSM source program libraries and OSM macro libraries in MLU format are permitted.

2.4.1.4 SOURCE-PROPERTIES option

Function

The SOURCE-PROPERTIES option can be used to define the format of the source program, the instruction set, and a value for the system variable symbol &SYSPARM.

Format

```
COMPILE

SOURCE-PROPERTIES = STD / PARAMETERS(...)

    PARAMETERS(...)
        |  FROM-COLUMN = 1 / <integer 1..70>
        | ,TO-COLUMN = 71 / <integer 2..255>
        | ,CONTINUATION-COLUMN = 16 / <integer 1..255> / NO-CONTINUATION
        | ,LOW-CASE-CONVERSION = NO / YES
        | ,INSTRUCTION-SET = HOST-STD / BS2000-ESA / BS2000-XS / BS2000-NXS
        |                   / DUET
        | ,PREDEFINED-VARIABLES = NONE / SYS(...)
        |    SYS(...)
        |        |  SYSPARM = <c-string 1..255>
```

SOURCE-PROPERTIES = STD
The default values of the PARAMETERS(...) structure are used.

SOURCE-PROPERTIES = PARAMETERS(...)

FROM-COLUMN = 1 / <integer 1..70>
Defines the begin column for the assembly of a source line.

TO-COLUMN = 71 / <integer 2..255>
Defines the end column for the assembly of a source line.

CONTINUATION-COLUMN = 16 / <integer 1..255> / NO-CONTINUATION
Defines the begin column for the continuation line of an instruction in the source. No continuation line is used if NO-CONTINUATION is selected.

LOW-CASE-CONVERSION = NO / YES
If YES, lowercase letters are converted to uppercase (see "ASSEMBH (BS2000) Reference Manual" [1]).

INSTRUCTION-SET = HOST-STD / BS2000-ESA / BS2000-XS / BS2000-NXS / DUET
Defines the instruction set to be used (see section 11.3).
Depending on the hardware/software interface of the CPU, BS2000-ESA, BS2000-XS or BS2000-NXS is used for HOST-STD.

PREDEFINED-VARIABLES = NONE / SYS(...)
Passes external user information to a system symbol.

SYSPARM = <c-string 1..255>
Assigns a value to the system variable symbol &SYSPARM.

Notes

– On entering operands
The ’SOURCE-PROPERTIES’ and ’PARAMETERS( )’ entries may be omitted.

Example

The complete specification

```
//C SOURCE-PROPERTIES=PARAMETERS(FROM-COLUMN=2)
```

can be written as

```
//C SOURCE-PROPER=(FROM-COLUMN=2)
```
or

```
//C S-PRO=(2)
```
or

```
//C FROM-COLUMN=2
```

’PREDEFINED-VARIABLES’ and SYS( ) may also be omitted when entering ’SYSPARM’.

Example

An alternative way of entering

```
//C S-PRO=PREDEFINED-VARIABLES(SYS(SYSPARM='100'))
```
is

```
//C S-PRO=(SYSPARM='100')
```

– On the format of the source program

  If NO-CONTINUATION is selected, lines will not be continued in the next line. Otherwise, the continuation character must be entered in the end column + 1, and the continuation line must begin as of the continuation column. The following rule must be observed:

  begin column ≤ continuation column ≤ end column

  The begin column must always be less than the end column.

  Default values apply if an illegal entry is made.

– On column specifications

  For macro elements

  In the case of inputs from macro elements, FROM-COLUMN, TO-COLUMN, and CONTINUATION-COLUMN are predefined according to standard format as 1, 71, and 16, respectively. This generally applies to all macro elements (from system macro libraries and user-own macro libraries).

  For source deck macros

  A macro definition in the source text is processed as a source line, i.e. the options set for the source text apply.

  For COPY elements

  COPY elements are read in the same way as the line containing the COPY instruction statement. This means that a COPY instruction in the source text or in a source deck macro is processed like a line of source, i.e. with the same options in effect as those that were set for the source text. A COPY instruction in a library macro is read in standard format.

2.4.2 Options for object module generation

These options control the output of an object module or link-and-load module

```
COMPILE

,COMPILER-ACTION =   generation of object module or link-and-load module
,MODULE-LIBRARY =    library for modules
```

2.4.2.1 COMPILER-ACTION option

Function

The COMPILER-ACTION option determines whether an object module or linkand-load module is to be generated or only a syntax check performed.

Format

```
COMPILE

COMPILER-ACTION = MODULE-GENERATION(...) / SYNTAX-CHECK(...)

    MODULE-GENERATION(...)
        |  MODE = STD / F-ASSEMB-COMPATIBLE
        | ,MODULE-FORMAT = OM / LLM(...)
        |    LLM(...)
        |        |  EXTERNAL-NAMES = STD / TRUNCATED
    SYNTAX-CHECK(...)
        |  MODE = STD / F-ASSEMB-COMPATIBLE
```

COMPILER-ACTION = MODULE-GENERATION(...)

MODE = STD
A syntax check is performed, and a module is generated.

MODE = F-ASSEMB-COMPATIBLE
Specific incompatibilities with the F-Assembler (ASSEMB) are prevented by specifying this operand. Processing is then compatible with ASSEMB V30.0A.

MODULE-FORMAT = OM
A module in OM format (object module format) is generated, which is stored either in the temporary EAM object module file (default) or as an R-type library element (see the MODULE-LIBRARY option).

MODULE-FORMAT = LLM(...)

EXTERNAL-NAMES = STD / TRUNCATED

A module is generated in LLM format (link-and-load module format). External names are truncated to 32 characters (STD) or 8 characters (TRUNCATED). It can be stored only as an (L-type) library element (see the MODULE-LIBRARY option).

COMPILER-ACTION = SYNTAX-CHECK(...)

MODE = STD
No module is generated; only a syntax check is performed.

MODE = F-ASSEMB-COMPATIBLE
Specific incompatibilities with the F-Assembler (ASSEMB) are prevented by specifying this operand. Processing is then compatible with ASSEMB V30.0A.

Notes

On F-ASSEMB-COMPATIBLE

- The programs concerned should be modified since this operand will be dropped in the future.

On specific incompatibilities

- The processing of SETA and SETB operands is compatible with the F-Assembler.
- The notation C'...' is permitted in SETA, SETB, and relational expressions. The C is ignored.
- If a character value cannot be converted, no message is issued. The replacement value null is used for further computations.
- Incorrect operands for SPACE and EJECT are ignored.
- Remarks with MNOTE
  If the delimiting blank between the MNOTE operands and the remarks field is missing, everything that follows the closing single quote will be treated as a remark. Errors concerning unpaired quotes are not detected.

2.4.2.2 MODULE-LIBRARY option

Function

The MODULE-LIBRARY option can be used to specify where the module (object module or link-and-load module) is to be output.

Format

```
COMPILE

MODULE-LIBRARY = *OMF / <full-filename 1..54 without gen-vers>(...)

     <full-filename 1..54 without gen-vers>(...)
          |
          | ELEMENT = *STD(...) /
          |           <composed-name 1..64>(...)
          |
          |  *STD(...)
          |      |
          |      | VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
          |      |           <composed-name 1..24>
          |
          |  <composed-name 1..64>(...)
          |      |
          |      | VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
          |      |           <composed-name 1..24>
```

MODULE-LIBRARY = *OMF
The object module is placed in the temporary EAM object module file.

MODULE-LIBRARY = <full-filename 1..54 without gen-vers>(...)
Name of the PLAM library in which the object module (OM format) or link-and-load module (LLM format) is to be placed. For LLMs, the MODULE-LIBRARY option must be used to specify a library. If no library is specified, a message is issued.

ELEMENT = *STD(...)
Name of the object module (library element of type R) or link-and-load module (library element of type L). The element is assigned the name of the first control section. If the first control section is unnamed, no module is generated, and an appropriate message is issued.

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = *INCREMENT
The element is assigned the incremented version.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = <composed-name 1..24>
Version designation of the element.

ELEMENT = <composed-name 1..64>(...)
Name of the element.

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = *INCREMENT
The element is assigned the incremented version.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = <composed-name 1..24>
Version designation of the element.

Notes

- On the length of the element name
  Only element names with a maximum of 8 characters are currently processed by the linkage editor TSOSLNK. For further processing with the BINDER linkage editor or the DBL linking loader, element names of LLMs can be up to 32 characters long.
- On '@'
  As of PLAM V1.4, '@' may no longer be specified as a version for object module output.
- On VERSION = *INCREMENT (incremented version)
  See version designation *INCREMENT and automatic version incrementation in the "LMS User Guide" [8].

2.4.3 Option for CIF support
COMPILATION-INFO option

Function

This option controls whether the CIF information is to be stored in a PLAM library.

Format

```
COMPILE

COMPILATION-INFO = NONE / PARAMETERS(...)

   PARAMETERS(...)
      INFORMATION = STD / MAXIMUM
     ,OUTPUT = *LIBRARY-ELEMENT(...)

        *LIBRARY-ELEMENT(...)
           LIBRARY = <full-filename 1..54 without gen-vers>

           ,ELEMENT = <composed-name 1..64>(...)
              VERSION = *UPPER-LIMIT / *INCREMENT /
                        *HIGHEST-EXISTING / <composed-name 1..24>
```

COMPILATION-INFO = NONE
The CIF is only created temporarily in order to generate the listing.

COMPILATION-INFO = PARAMETERS(...)

INFORMATION = STD / MAXIMUM
Defines the scope of information in the CIF.

Meaning of STD: only the information that was requested in the listing operand is provided.

Meaning of MAXIMUM: the entire information is made available. This has no effect on the standard listing, which is controlled via the LISTING operand.

OUTPUT = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..54 without gen-vers>
Name of a PLAM library in which the CIF information is stored.

ELEMENT = <composed-name 1..64>(...)
Name of the library element (type H).

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = *INCREMENT
The element is assigned the incremented version.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = <composed-name 1..24>
Version designation of the element.

Notes

– On the formation of element names in a multiple assembly
In the case of a multiple assembly, a separate CIF element is stored in the specified library for each assembly unit. The element name for the n-th assembly unit (where n≥ 2) is formed by appending '.n' to the CIF element name of the first assembly unit:

cifelementname.n (the version remains the same)

– On VERSION = *INCREMENT (incremented version)
See version designation *INCREMENT and automatic version incrementation in the "LMS User Guide" [8].

2.4.4 Option for listing support
LISTING option

Function

The LISTING option is used to specify the layout and scope of the assembler listing and where it is to be stored.

Format

```
COMPILE

LISTING = STD / PARAMETERS(...)

  PARAMETERS(...)
     SOURCE-PRINT = NO / WITH-OBJECT-CODE(...) / SOURCE-ONLY(...) /
                    ERRORS-ONLY(...)

       WITH-OBJECT-CODE(...)
          PRINT-STATEMENTS = ACCEPTED / IGNORED
          LINE-NUMBERING = NO / YES

       SOURCE-ONLY(...)
          PRINT-STATEMENTS = ACCEPTED / IGNORED
          LINE-NUMBERING = NO / YES

       ERRORS-ONLY(...)
          LINE-NUMBERING = NO / YES

     ,SOURCE-FORMAT = STD / STRUCTURED(...)

       STRUCTURED(...)
          EVALUATED-NEST-LEVEL = 1 / ALL
         ,INDENTATION-AMOUNT = 2 / <integer 1..8>
         ,FIXED-AREA-START = NONE / <integer 60..255>
         ,STRUCT-MACRO-PRINT = STD / OBJECT-CODE-ONLY /
                               WITH-OBJECT-CODE / NO-OBJECT-CODE

     ,MACRO-PRINT = STD / PARAMETERS(...)

       PARAMETERS(...)
          NOPRINT-NEST-LEVEL = 255 / <integer 1..255>
         ,NOPRINT-PREFIX = *NONE / list-poss(256): <name 1..64>
         ,TITLE-STATEMENTS = ACCEPTED / IGNORED
         ,MACRO-ORIGIN-INFO = SEPARATE / INSERTED
```

continued>

continued

```
,MIN-MESSAGE-WEIGHT = NOTE / WARNING / SIGNIFICANT / SERIOUS / FATAL

,CROSS-REFERENCE = STD / ALL / NO / PARAMETERS(...)

   PARAMETERS(...)
      SYMBOL = NO / YES(...)
        YES(...)
          WITH-ATTRIBUTES = NO / YES
         ,REFERENCED-ONLY = NO / YES
         ,PREFIX = ALL / EXCEPT(...) / ONLY(...)
            EXCEPT(...)
              CHARACTERS = list-poss(256): <name 1..64>
            ONLY(...)
              CHARACTERS = list-poss(256): <name 1..64>
      ,LITERAL = NO / YES
      ,MACRO = NO / YES
      ,COPY = NO / YES
      ,DIAGNOSTICS = NO / YES

,EXTERNAL-DICTIONARY = NO / YES

,LAYOUT = STD / PARAMETERS(...)

   PARAMETERS(...)
      LINES-PER-PAGE = 60 / <integer 15..255>
     ,LASER-PRINTER = NO / ND2
     ,FORMAT = STD / F-ASSEMB-COMPATIBLE(...)
        F-ASSEMB-COMPATIBLE(...)
          MESSAGE-PLACEMENT = SEPARATE / INSERTED

,OUTPUT = *SYSLST / *NONE / <full-filename 1..54> /
          *LIBRARY-ELEMENT(...) / *SAVLST

   *LIBRARY-ELEMENT(...)
      LIBRARY = <full-filename 1..54 without-gen-vers>
     ,ELEMENT = <composed-name 1..64>(...)
        VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
                  <composed-name 1..24>
```

LISTING = STD
The default values of the PARAMETERS(...) structure are used.

LISTING = PARAMETERS(...)

SOURCE-PRINT =
Controls the listing of the source program.

SOURCE-PRINT = NO
The source program is not listed.

SOURCE-PRINT = WITH-OBJECT-CODE(...)
Listing of source lines with object code.

PRINT-STATEMENTS = ACCEPTED / IGNORED
The NOGEN, OFF, and NOCOPY entries of the PRINT statement are either executed or ignored.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.

SOURCE-PRINT = SOURCE-ONLY(...)
Listing of source lines only, i.e. without object code.

PRINT-STATEMENTS = ACCEPTED / IGNORED
The NOGEN, OFF, and NOCOPY entries of the PRINT statement are either executed or ignored.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.
Numbering commences at 100 with an increment of 100 to 8 positions. In the case of source deck macros, no numbering is performed.

SOURCE-PRINT = ERRORS-ONLY(...)
Only source lines containing errors are listed.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.

SOURCE-FORMAT = STD
The default values of the STRUCTURED(...) structure are used.

SOURCE-FORMAT = STRUCTURED(...)
A structured listing is generated provided the predefined structured programming macros were used in the source program (structure macros, "@-Makros").

EVALUATED-NEST-LEVEL = 1 / ALL
Either only those structure macro calls that occur in the source or all of them are listed (including those called by generation).

INDENTATION-AMOUNT = 2 / <integer 1...8>
Specifies in columns the amount of indentation (and thus also the spacing between the vertical structure lines).

FIXED-AREA-START = NONE / <integer 60...255>
Specifies the column as of which the source program is not to be changed or moved by the structuring.

STRUCT-MACRO-PRINT =
Controls the listing of the structure macros.

STRUCT-MACRO-PRINT = STD
Structure macros are listed in the same way as other macros.

STRUCT-MACRO-PRINT = OBJECT-CODE-ONLY
Only the generated object code is output for all structure macros. This has the same effect as specifying PRINT NOGEN,CODE. The NOPRINT-PREFIX option is ignored.

STRUCT-MACRO-PRINT = WITH-OBJECT-CODE
The object code is listed with the associated generated source representation. In the case of macros that are excluded from the listing by means of the NOPRINT-PREFIX option or a PRINT NOGEN source statement, only the object code is listed.

STRUCT-MACRO-PRINT = NO-OBJECT-CODE
The object code is not listed for structure macros.

MACRO-PRINT =
Controls the listing of macro elements in the source listing.

MACRO-PRINT = STD
The default values of the PARAMETERS(...) structure are used.

MACRO-PRINT = PARAMETERS(...)

NOPRINT-NEST-LEVEL = 255 / <integer 1..255>
Defines the maximum macro nesting level up to which generation is listed.

NOPRINT-PREFIX = *NONE / <name 1..64>
Defines a list of macro name prefixes (256) that are not to be listed. The PREFIX-EXCEPTION = <name 1..1> operand is now supported only for compatibility considerations. When NOPRINT-PREFIX is set, PREFIX-EXCEPTION is no longer evaluated.

TITLE-STATEMENTS = ACCEPTED / IGNORED
TITLE statements generated by macros are either executed or ignored.

MACRO-ORIGIN-INFO = SEPARATE / INSERTED
Defines where the macro identification line (version, creation date, and link name of the macro library) is placed in the listing. With SEPARATE, the message is placed in the macro XREF listing; with INSERTED, it additionally appears after the macro instruction.

MIN-MESSAGE-WEIGHT = NOTE / WARNING / SIGNIFICANT / SERIOUS / FATAL
Defines the minimum error weight as of which errors are to be included in the listing; only these errors are included in the summary line.

CROSS-REFERENCE = STD / ALL / NO / PARAMETERS(...)
Controls the scope of cross-reference listings.

CROSS-REFERENCE = STD
The default values of the PARAMETERS(...) structure are used.

CROSS-REFERENCE = ALL
Signifies that the cross-reference listings are to be output in the most comprehensive form; that is, the following values are applicable: SYMBOL=YES (WITH-ATTRIBUTES=YES, REFERENCED-ONLY=NO, PREFIX=ALL), LITERAL=YES, MACRO=YES, COPY=YES, DIAGNOSTICS=YES.

CROSS-REFERENCE = PARAMETERS(...)

SYMBOL = NO / YES(...)
Controls output of the reference list for symbols (symbol XREF).

WITH-ATTRIBUTES = NO / YES
Determines whether the associated attributes, which refer to the mode of access, are also to be output.
W Write access
R Read-only access by instructions
A Address access
E EQU/ORG instructions

REFERENCED-ONLY = NO / YES
Defines whether only referenced symbols are to be output.

PREFIX = ALL / EXCEPT(...) / ONLY(...)
Enables or suppresses the output of symbols with a specific prefix.

PREFIX = EXCEPT(CHARACTERS=<name 1..64>)
Defines the prefix of symbols to be excluded from the output (256).

PREFIX = ONLY(CHARACTERS=<name 1..64>)
Defines the prefix of symbols to be output (256).

LITERAL = NO / YES
Determines output of the reference list for literals.

MACRO = NO / YES
Determines output of the reference list for macros.

COPY = NO / YES
Determines output of the reference list for COPY elements.

DIAGNOSTICS = NO / YES
Determines output of the reference list for the assembler flags that have occurred.

EXTERNAL-DICTIONARY = NO / YES
Determines whether external references of the assembled module (ENTRY, EXTRN, WXTRN etc.) are to be included in the listing.

LAYOUT =
Defines the layout of the listing.

LAYOUT = STD
The default values of the PARAMETERS(...) structure are used.

LAYOUT = PARAMETERS(...)

LINES-PER-PAGE = 60 / <integer 15..255>
Defines the number of lines in each page of the listing.

LASER-PRINTER = NO / ND2
Defines whether a laser printer listing is to be output.

FORMAT = STD
The listing is created in the standard format of ASSEMBH.

FORMAT = F-ASSEMB-COMPATIBLE(...)
A listing is produced in a format that is compatible to the F-Assembler (ASSEMB V30.0A).

MESSAGE-PLACEMENT = SEPARATE / INSERTED
Determines where error messages are to be placed in the listing. SEPARATE results in a flag in the source line and an entry in the diagnostic XREF listing; INSERTED causes the error message to be additionally printed after the incorrect source line.

OUTPUT =
Names the output medium for the assembler listing.

If you start an assembly by using the diagnostic routine ASSDIAG (see chapter 8) and require the corresponding listing, this will only be produced if you terminate ASSDIAG with END L.
(Specifying END without L will not produce a listing.)

OUTPUT = *SYSLST
The assembler listing is output to the system file SYSLST.

OUTPUT = *NONE
The assembler listing is not output.

OUTPUT = <full-filename 1..54>
The assembler listing is output to a cataloged file.

OUTPUT = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..54>)
Designates the library name for the output of assembler listings.

ELEMENT = <composed-name 1..64>(...)
Name of an element of type P.

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = *INCREMENT
The element is assigned the incremented version.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = <composed-name 1..24>
Version designation of the element.

OUTPUT = *SAVLST
The assembler listing is output with an ISAM key (see COMOPT SAVLST).

Notes

On entering operands

– The 'LISTING' and 'PARAMETERS( )' entries may be omitted.

Example

The specification:

```
//C LISTING=PARAMETERS(SOURCE-PRINT=ERRORS-ONLY)
```

can also be entered as

```
//C SOURCE-PRINT=ERRORS-ONLY
```
or

```
//C S-PRI=ERR-O
```

– Entries for 'MACRO-PRINT' ('NOPRINT-NEST-LEVEL' etc.) can be made without specifying 'MACRO-PRINT' and 'PARAMETERS( )'.

Example

The specification:

```
//C MACRO-PRINT=PARAMETERS(NOPRINT-NEST-LEVEL=20)
```

can also be entered as

```
//C NOPRINT-NEST-LEVEL=20
```

– Entries for ’CROSS-REFERENCE’ can be made without specifying ’PARAMETERS( )’, ’SYMBOL’, and ’YES( )’.

Example

The specification:

```
//C CROSS-REFERENCE=PARAMETERS(SYMBOL=YES(WITH-ATTRIBUTES=NO))
```

can also be entered as

```
//C CROSS-REF=(WITH-ATTR=NO)
```

– Entries for ’LAYOUT’ can be made without specifying ’PARAMETERS( )’.

Example

The specification:

```
//C LAYOUT=PARAMETERS(LASER-PRINTER=ND2)
```

can also be entered as

```
//C LAYOUT=(LASER-PRINTER=ND2)
```

– Entries for ’OUTPUT’ can be made without specifying ’*LIBRARY-ELEMENT( )’.

Example

The specification:

```
//C SOURCE=filename,OUTPUT=*LIB-ELEM(LIB=lib)
```

can also be entered as

```
//C SOURCE=filename,OUTPUT=(lib)
```

– On VERSION = *INCREMENT (incremented version)
See version designation *INCREMENT and automatic version incrementation in the "LMS User Guide" [8].

2.4.5 Option for debugging support TEST-SUPPORT option

Not supported by ASSEMBH-BC !

Function

The TEST-SUPPORT option controls whether LSD information is generated and stored in the object module.
The LSD information in the object module is a prerequisite for symbolic debugging with AID (see chapter 9, "The Advanced Interactive Debugger AID", and the manual "AID, Debugging of ASSEMBH Programs" [2]).

Format

```
COMPILE
```

```
TEST-SUPPORT = NO / NONE / AID
```

TEST-SUPPORT = NO / NONE
Symbolic debugging with AID is not supported.

TEST-SUPPORT = AID
Symbolic debugging with AID is supported.
ASSEMBH stores a consistency constant with a length of 8 bytes after the first control section in the object module. This constant is used by AID to ensure consistency between the object module and the LSD information.

2.4.6 Option to terminate assembly COMPILER-TERMINATION option

Function

The COMPILER-TERMINATION option can be used to define termination conditions and nesting levels to be interpreted by the assembler.

Format

```
COMPILE

COMPILER-TERMINATION = STD / PARAMETERS(...)

    PARAMETERS(...)
        MAX-ERROR-WEIGHT = WARNING / SIGNIFICANT / SERIOUS / FATAL
        MAX-ERROR-NUMBER = 32767 / <integer 0..32767>
        MAX-MACRO-NEST-LEVEL = 255 / <integer 1..255>
        MAX-COPY-NEST-LEVEL = 5 / <integer 1..255>
```

COMPILER-TERMINATION = STD
The default values of the PARAMETERS(...) structure are used.

COMPILER-TERMINATION = PARAMETERS(...)

MAX-ERROR-WEIGHT = WARNING / SIGNIFICANT / SERIOUS / FATAL
Defines the error severity (weight) as a termination condition, i.e. the error class at which assembly is to be terminated.

MAX-ERROR-NUMBER = 32767 / <integer 0..32767>
Defines a number of errors as a termination condition. The assembly is to be terminated as soon as this number is exceeded.

MAX-MACRO-NEST-LEVEL = 255 / <integer 1..255>
Defines the maximum nesting level for macro elements.

MAX-COPY-NEST-LEVEL = 5 / <integer 1..255>
Defines the maximum nesting level for COPY elements.

Notes

– On entering operands
  The 'COMPILER-TERMINATION' and 'PARAMETERS( )' entries may be omitted.

  Example

  The specification

```
//C COMPILER-TERMINATION=PARAMETERS(MAX-ERROR-NUMBER=10)
```

  can also be entered as

```
//C MAX-ERROR-NUMBER=10
```

– The following applies if the maximum nesting level for macro elements (MAX-MACRO-NEST-LEVEL) and COPY elements (MAX-COPY-NEST-LEVEL) is exceeded:

  For macro elements: the macro instruction is ignored.
  For COPY elements: the COPY call is ignored.
  For COPY within macro definitions: the COPY level at the time the macro definition was read applies.

2.4.7 Option to activate the correction cycle CORRECTION-CYCLE option

Not supported by ASSEMBH-BC !

Function

The CORRECTION-CYCLE option can be used to specify whether and under which conditions the diagnostic routine ASSDIAG is to be called (see chapter 8) for diagnostic analysis of the assembly and interactive correction of source code.

Format

```
COMPILE
```

```
CORRECTION-CYCLE = NO / YES(...)

  YES(...)
    |
    |  ACTIVATION-WEIGHT = ALWAYS / NOTE / WARNING / SIGNIFICANT / SERIOUS
```

CORRECTION-CYCLE = NO

CORRECTION-CYCLE = YES(...)

ACTIVATION-WEIGHT =
Defines the error severity (weight) at which ASSDIAG is to be called.

ACTIVATION-WEIGHT = ALWAYS
Regardless of the result of the assembly, ASSDIAG is called at the end of an assembly unit.

ACTIVATION-WEIGHT = NOTE / WARNING / SIGNIFICANT / SERIOUS
ASSDIAG is called at the end of an assembly unit if the specified error severity is reached.

Note

ASSDIAG can be used to correct source text lines and start the assembly again. A corresponding assembler listing is output only if ASSDIAG is terminated with END L. (If END is given without L, no listing is output.) This cycle is repeated until the set error weight is no longer reached (i.e. the correction is successful and the assembly is executed without errors) or until the user terminates the cycle in ASSDIAG (see chapter 8).

2.4.8 Option for maintenance support MAINTENANCE-OPTIONS option

Function

MAINTENANCE-OPTIONS can be used to execute tests for CCW channel instructions.

Format

```
COMPILE

MAINTENANCE-OPTIONS = STD / PARAMETERS(...)

    PARAMETERS(...)

        CHANNEL-INSTRUCTIONS = NO / YES
```

MAINTENANCE-OPTIONS = STD
The default values of the PARAMETERS(...) structure are used.

MAINTENANCE-OPTIONS = PARAMETERS(...)

CHANNEL-INSTRUCTIONS = NO / YES
Support of tests for CCW channel instructions.

Note

This option is executed in "expert mode" only, i.e. is not available in interactive mode with guidance (menu mode).

2.4.9 Option for reducing the virtual address space requirement COMPILATION-SPACE option

Function

COMPILATION-SPACE enables assembly and list generation to be performed in a smaller virtual address space, albeit at the cost of some performance degradation.

Format

```
COMPILE
```

```
COMPILATION-SPACE = STD / SMALL
```

COMPILATION-SPACE = STD
Assembly and list generation take place in the virtual XS address space.

COMPILATION-SPACE = SMALL
Assembly and list generation take place in a reduced virtual address space, with attendant performance degradation.

Note

A user wanting to produce a very extensive listing on a 25-bit machine will have to set up a CIF (by specifying the SDF option COMPILATION-INFO, see section 2.4.3). Otherwise there is a risk of storage bottlenecks occurring as a result of the CIF information placed in virtual memory, and of assembly being aborted.

2.5 The standalone listing generator ASSLG

Not supported by ASSEMBH-BC !

The standalone listing generator is started with the following command:

```
/START-PROGRAM $ASSLG
```

2.5.1 GENERATE statement

Function

The standalone listing generator ASSLG creates listings from the CIF information stored in a library (see COMPILATION-INFO, section 2.4.3). This is done via the GENERATE statement.

Format

```
GENERATE

 COMPILER-INFO-FILE = *LIBRARY-ELEMENT(...)

    *LIBRARY-ELEMENT(...)
       |  LIBRARY = <full-filename 1..54 without gen-vers>
       | ,ELEMENT = <composed-name 1..64>(...)
       |    |  VERSION = *HIGHEST-EXISTING / *UPPER-LIMIT /
       |    |            <composed-name 1..24>

,SOURCE-PRINT = NO / WITH-OBJECT-CODE(...) / SOURCE-ONLY(...) /
                ERRORS-ONLY(...)

     WITH-OBJECT-CODE(...)
       |  LINE-NUMBERING = NO / YES

     SOURCE-ONLY(...)
       |  LINE-NUMBERING = NO / YES

     ERRORS-ONLY(...)
       |  LINE-NUMBERING = NO / YES

,SOURCE-FORMAT = STD / STRUCTURED(...)

     STRUCTURED(...)
       |  EVALUATED-NEST-LEVEL = 1 / ALL
       | ,INDENTATION-AMOUNT = 2 / <integer 1..8>
       | ,FIXED-AREA-START = NONE / <integer 60..255>
       | ,STRUCT-MACRO-PRINT = STD / OBJECT-CODE-ONLY /
       |                       WITH-OBJECT-CODE / NO-OBJECT-CODE

,MACRO-PRINT = STD / PARAMETERS(...)

     PARAMETERS(...)
       |  MACRO-ORIGIN-INFO = SEPARATE / INSERTED

,MIN-MESSAGE-WEIGHT = NOTE / WARNING / SIGNIFICANT / SERIOUS / FATAL
```

continued>

continued

```
,CROSS-REFERENCE = STD / ALL / NO / PARAMETERS(...)

     PARAMETERS(...)
       |
       |  SYMBOL = NO / YES(...)
       |
       |     YES(...)
       |       |
       |       |  WITH-ATTRIBUTES = NO / YES
       |       | ,REFERENCED-ONLY = NO / YES
       |       | ,PREFIX = ALL / EXCEPT(...) / ONLY(...)
       |       |
       |       |     EXCEPT(...)
       |       |       |
       |       |       |  CHARACTERS = list-poss(256): <name 1..64>
       |       |       |
       |       |     ONLY(...)
       |       |       |
       |       |       |  CHARACTERS = list-poss(256): <name 1..64>
       |
       | ,LITERAL = NO / YES
       | ,MACRO = NO / YES
       | ,COPY = NO / YES
       | ,DIAGNOSTICS = NO / YES

,EXTERNAL-DICTIONARY = NO / YES

,LAYOUT = STD / PARAMETERS(...)

     PARAMETERS(...)
       |
       |  LINES-PER-PAGE = 60 / <integer 15..255>
       | ,LASER-PRINTER = NO / ND2
       | ,FORMAT = STD / F-ASSEMB-COMPATIBLE(...)
       |
       |     F-ASSEMB-COMPATIBLE(...)
       |       |
       |       |  MESSAGE-PLACEMENT = SEPARATE / INSERTED

,OUTPUT = *SYSLST / *SAVLST / <full-filename 1..54 without gen-vers> /
          *LIBRARY-ELEMENT(...)

     *LIBRARY-ELEMENT(...)
       |
       |  LIBRARY = <full-filename 1..54 without gen-vers>
       |
       | ,ELEMENT = <composed-name 1..64>(...)
       |   |
       |   |  VERSION = *UPPER-LIMIT / *INCREMENT / *HIGHEST-EXISTING /
       |   |            <composed-name 1..24>

,GENERATION-SPACE = STD / SMALL
```

COMPILER-INFO-FILE = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..54>
Name of the library in which the CIF information is stored (see the COMPILATION-INFO option).

ELEMENT = <composed-name 1..64>(...)
Name of the library element.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = <composed-name 1..24>
Version designation of the element.

SOURCE-PRINT =
Controls the listing of the source program.

SOURCE-PRINT = NO
The source program is not listed.

SOURCE-PRINT = WITH-OBJECT-CODE(...)
Listing of source lines with object code.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.

SOURCE-PRINT = SOURCE-ONLY(...)
Listing of source lines only, i.e. without object code.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.

SOURCE-PRINT = ERRORS-ONLY(...)
Only source lines containing errors are listed.

LINE-NUMBERING = NO / YES
Specifies whether the lines from the source are to be consecutively numbered in the identification field (columns 73-80) in the assembler listing.

SOURCE-FORMAT = STD
The default values of the STRUCTURED(...) structure are used.

SOURCE-FORMAT = STRUCTURED(...)
A structured listing is generated provided the predefined structured programming macros were used in the source program (structure macros, "@-Makros").

EVALUATED-NEST-LEVEL = 1 / ALL
Either only those structure macro calls that occur in the source or all of them are listed (including those called by generation).

INDENTATION-AMOUNT = 2 / <integer 1...8>
Specifies in columns the amount of indentation (and thus also the spacing between the vertical structure lines).

FIXED-AREA-START = NONE / <integer 60...255>
Specifies the column as of which the source program is not to be changed or moved by the structuring.

STRUCT-MACRO-PRINT =
Controls the listing of the structure macros.

STRUCT-MACRO-PRINT = STD
Structure macros are listed in the same way as other macros.

STRUCT-MACRO-PRINT = OBJECT-CODE-ONLY
Only the generated object code is output for all structure macros. This has the same effect as specifying PRINT NOGEN,CODE. The NOPRINT-PREFIX option is ignored.

STRUCT-MACRO-PRINT = WITH-OBJECT-CODE
The object code is listed with the associated generated source representation. In the case of macros that are excluded from the listing by means of the NOPRINT-PREFIX option or a PRINT NOGEN source statement, only the object code is listed.

STRUCT-MACRO-PRINT = NO-OBJECT-CODE
The object code is not listed for structure macros.

MACRO-PRINT =
Controls the listing of macro elements in the source listing.

MACRO-PRINT = STD
The default values of the PARAMETERS(...) structure are used.

MACRO-PRINT = PARAMETERS(...)

MACRO-ORIGIN-INFO = SEPARATE / INSERTED
Defines where the macro identification line (version, creation date, and link name of the macro library) is placed in the listing. With SEPARATE, the message is placed in the macro XREF listing; with INSERTED, it also appears after the macro instruction.

MIN-MESSAGE-WEIGHT = NOTE / WARNING / SIGNIFICANT / SERIOUS / FATAL
Defines the minimum error weight for errors to be included in the listing; only these errors are entered in the summary line.

CROSS-REFERENCE = STD / ALL / NO / PARAMETERS(...)
Controls the scope of cross-reference listings.

CROSS-REFERENCE = STD
The default values of the PARAMETERS(...) structure are used.

CROSS-REFERENCE = ALL
Signifies that the cross-reference listings are to be output in the most comprehensive form; that is, the following values are applicable:
SYMBOL=YES (WITH-ATTRIBUTES=YES, REFERENCED-ONLY=NO, PREFIX=ALL), LITERAL=YES, MACRO=YES, COPY=YES, DIAGNOSTICS=YES.

CROSS-REFERENCE = PARAMETERS(...)

SYMBOL = NO / YES(...)
Controls output of the reference list for symbols (symbol XREF).

WITH-ATTRIBUTES = NO / YES
Determines whether the associated attributes, which refer to the mode of access, are also to be output.
W Write access
R Read-only access by instructions
A Address access
E EQU/ORG instructions.

REFERENCED-ONLY = NO / YES
Defines whether only referenced symbols are to be output.

PREFIX = ALL / EXCEPT(...) / ONLY(...)
Enables or suppresses the output of symbols with a specific prefix.

PREFIX = EXCEPT(CHARACTERS=<name 1..64>)
Defines the prefix of symbols to be excluded from the output (256).

PREFIX = ONLY(CHARACTERS=<name 1..64>)
Defines the prefix of symbols to be output (256).

LITERAL = NO / YES
Determines output of the reference list for literals.

MACRO = NO / YES
Determines output of the reference list for macros.

COPY = NO / YES
Determines output of the reference list for COPY elements.

DIAGNOSTICS = NO / YES
Determines output of the reference list for the assembler flags that have occurred.

EXTERNAL-DICTIONARY = NO / YES
Determines whether external references of the assembled module (ENTRY, EXTRN, WXTRN, etc.) are to be included in the listing.

LAYOUT =
Defines the layout of the listing.

LAYOUT = STD
The default values of the PARAMETERS(...) structure are used.

LAYOUT = PARAMETERS(...)

LINES-PER-PAGE = 60 / <integer 15..255>
Defines the number of lines in each page of the listing.

LASER-PRINTER = NO / ND2
Defines whether a laser printer listing is to be output.

FORMAT = STD
The listing is created in the standard format of ASSEMBH.

FORMAT = F-ASSEMB-COMPATIBLE(...)
A listing is produced in a format that is compatible with the F-Assembler (ASSEMB V30.0A).

MESSAGE-PLACEMENT = SEPARATE / INSERTED
Determines where error messages are to be placed in the listing. SEPARATE results in a flag in the source line and an entry in the diagnostic XREF listing; INSERTED causes the error message to be additionally printed after the incorrect source line.

OUTPUT =
Names the output medium for the assembler listing.

OUTPUT = *SYSLST
The assembler listing is output to the system file SYSLST.

OUTPUT = *SAVLST
The assembler listing is output with an ISAM key (see COMOPT SAVLST).

OUTPUT = <full-filename 1..54>
The assembler listing is output to a cataloged file.

OUTPUT = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..54>
Defines the library name for output of the assembler listing.

ELEMENT = <composed-name 1..64>(...)
Name of the element (type P).

VERSION = *UPPER-LIMIT
The element is assigned the highest possible version.

VERSION = *INCREMENT
The element is assigned the incremented version.

VERSION = *HIGHEST-EXISTING
The element is assigned the highest existing version.

VERSION = <composed-name 1..24>
Version designation of the element.

GENERATION-SPACE = STD / SMALL
When SMALL is specified, the lists are generated in a reduced virtual address space, with attendant performance degradation.

3 Input/output of ASSEMBH

3.1 Input sources of ASSEMBH

Input to ASSEMBH consists of source text and user control statements, i.e. options (see chapter 2).
The source text is stored in a source program. Some parts of the source text can be generated via macro elements or read in from COPY elements during assembly. The options control the assembly sequence and the inputs and outputs of the assembler.

- Source program
  A source program can be either
  - entered via the system file SYSDTA, i.e.
    directly from a terminal or
    by assigning SYSDTA to a file or library, or
  - read from a file or library.

  Library elements of type S from a PLAM library or OSM source program library are permitted.
- Macro element
  A macro element is read from a PLAM library (element type M) or from an OSM macro library (MLU format).
- COPY element
  A COPY element is read from a PLAM library (element of type S or M) or from an OSM source program library or an OSM macro library (MLU format).

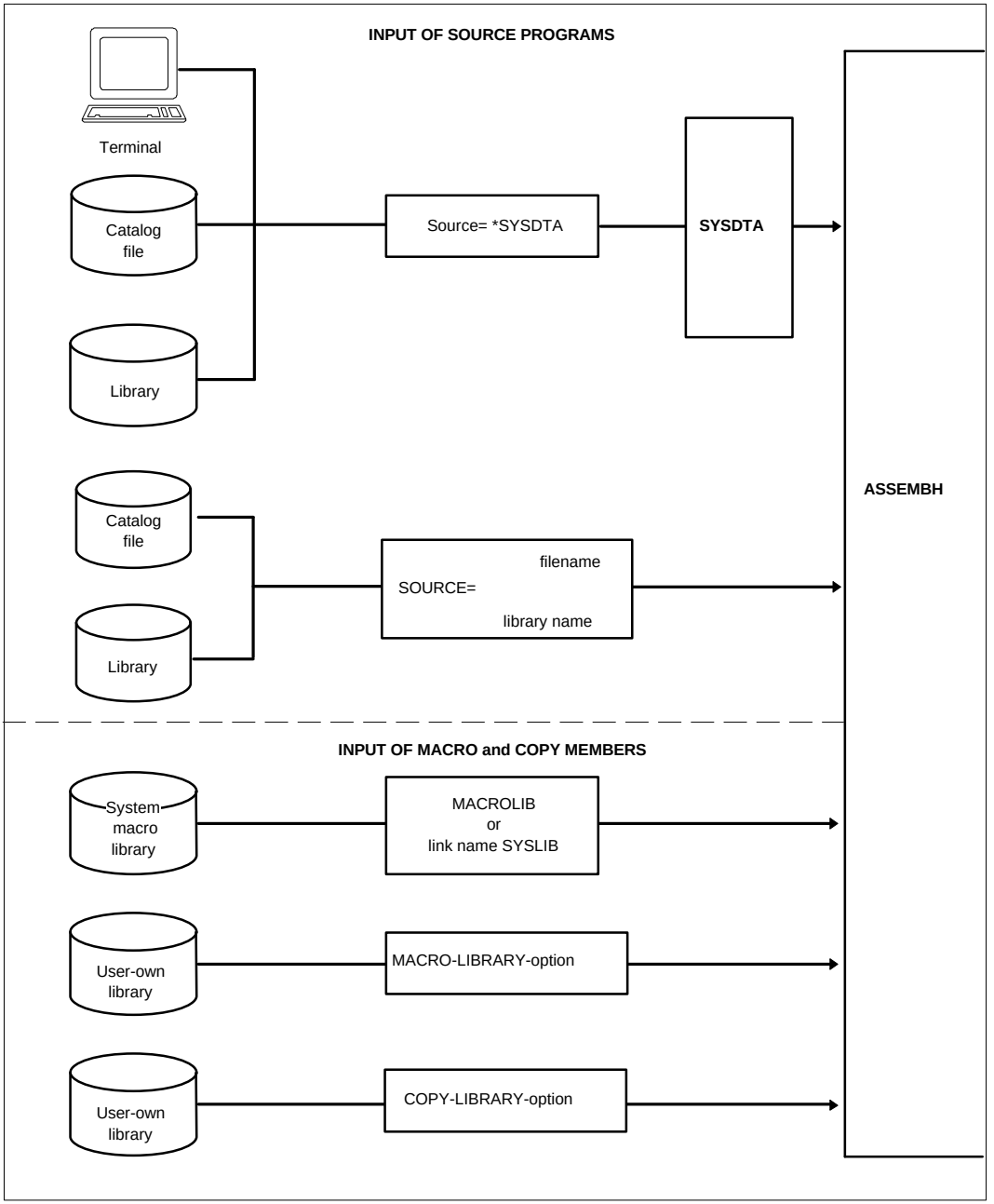


Fig. 3-1: Input sources of ASSEMBH

3.1.1 Input of the source program

The assembler interprets the content of a file as source text. The maximum permitted length for each line of source is 255 characters.
The option SOURCE-PROPERTIES, LOW-CASE-CONVERSION (see section 2.4.1.4) enables the use of both uppercase and lowercase letters in the source text (see chapter 2.1 in "ASSEMBH (BS2000) Reference Manual" [1]).
The default settings for the source text to be interpreted are columns 1 (begin column), 71 (end column), 72 (continuation character), and 16 (continuation column).
These values can be changed with the help of the SOURCE-PROPERTIES option (see section 2.4.1.4).
The default settings for the begin, end, and continuation columns can also be changed with the ICTL instruction (see "ASSEMBH (BS2000) Reference Manual" [1]).

Input via SYSDTA

Source text is normally entered from the terminal via the system file SYSDTA. After the start of ASSEMBH and the input of 'C', i.e.
//COMPILE SOURCE=*SYSDTA, ...default values..., ASSEMBH responds with '*' and requests the input of source text.

If the source text is to be read via SYSDTA from a file or a library element, SYSDTA must be assigned to a cataloged file or a library element (element type S from a PLAM library or an OSM source program library) before calling the assembler. The assignment is made using the SDF command ASSIGN-SYSDTA. The file or library element must contain a //COMPILE statement and an //END statement.

Example

```
                        ⎡ filename                                ⎤
/ASSIGN-SYSDTA TO-FILE=⎨                                          ⎬
                        ⎣ *LIB-ELEM(LIB=library,ELEM=element) ⎦

/START-PROGRAM $ASSEMBH
```

Input from files

The input of a source program from cataloged SAM or ISAM files is effected via the SOURCE option (see section 2.4.1.1).

Example

```
//COMPILE SOURCE=filename
```

Input from libraries

The input of a source program from libraries is effected via the SOURCE option (see section 2.4.1.1). Library elements of type S from PLAM libraries and from OSM source program libraries are permitted.

Example

```
//COMPILE SOURCE=(library,element)
```

3.1.2 Input of macro elements

In most cases, macro definitions are not entered in the source program, but are stored in macro libraries in the form of macro elements (see "ASSEMBH (BS2000) Reference Manual" [1]).
Macro elements of type M from PLAM libraries and from OSM macro libraries (MLU format) are permitted.
The source program itself contains only a macro instruction. During assembly, a sequence of instruction statements is generated from the macro definition under the control of parameters, and these instructions are incorporated into the source program.
The columns of text in a macro definition are interpreted on the basis of default values, i.e. with column 1 as the begin column, column 71 as the end column, column 72 as the continuation character, and column 16 as the continuation column.
Changes made via the SOURCE-PROPERTIES option or the ICTL instruction have no effect.

There are two types of macro libraries:

- user-own macro libraries and
- the system macro library MACROLIB ($TSOS.MACROLIB), which is accessible to all users.

User-own macro libraries

The MACRO-LIBRARY option (see section 2.4.1.2) can be used to specify up to 100 private user macro libraries. If the user-own macro libraries are to be addressed via link names, the appropriate SET-FILE-LINK command must be issued before starting the assembler.

Example

```
/SET-FILE-LINK LINK-NAME=maclink,FILE-NAME=maclib
```

The link name maclink is assigned to the macro library maclib.

```
//C MAC-LIB=(maclib1,maclib2,*LINK(maclink))
```

The macro libraries maclib1 and maclib2 are assigned together with the macro library maclib, which is assigned via the link name maclink.

System macro library

The system macro library is specified with a file control block (FCB) which contains LINK=SYSLIB. If a user wishes to use his own file as the system macro library, he can do so in two different ways:

– by assigning the file link name SYSLIB to his file with the SET-FILE-LINK command. A corresponding REMOVE-FILE-LINK command must be given by the user in this case as well.

  Example

  ```
  /SET-FILE-LINK LINK-NAME=SYSLIB,FILE-NAME=filename
  ```

– by renaming his file to the standard file name MACROLIB.

  Example

  ```
  /MOD-FILE-ATTR FILE-NAME=filename,NEW-NAME=MACROLIB
  ```

3.1.2.1 Search order for macro elements

During assembly, the system macro library and up to 100 user-own macro libraries may be accessed. When a macro instruction is processed, any unknown macro definition is searched for and read in the following order:

1st user-own macro library
. . .
. . .
100th user-own macro library
System macro library MACROLIB

The search order for macro elements from user-own macro libraries corresponds to the order of the libraries specified in the MACRO-LIBRARY option.

If there are several identically named elements of type M in a PLAM library, but with different versions, the element with the highest version is always used.

Inner macros of macros from the system macro library are searched for in this library only, assuming the corresponding macro definition has not already been read in.

3.1.3 Input of COPY elements

The source program itself contains only the COPY instruction. During the assembly run, this instruction is executed, and the stored sequence of instructions is copied from the library into the source program.
COPY elements from a PLAM library (element type S or M), an OSM source program library, or an OSM macro library (MLU format) can be copied into a source program with the COPY instruction (see "ASSEMBH (BS2000) Reference Manual" [1]).

The COPY-LIBRARY option (see section 2.4.1.3) can be used to specify up to 100 user-own libraries. PLAM libraries (element types S and M), OSM source program libraries, and OSM macro libraries (MLU format) are permitted.
The operand ELEMENT-TYPE = SOURCE-ONLY or MACRO-ONLY can be used to limit the search for an element to only the source partition or macro partition of a PLAM library.

Examples

```
/SET-FILE-LINK LINK-NAME=coplink,FILE-NAME=coplib
```

The link name coplink is assigned to the library coplib.

```
//C COPY-LIB=(coplib1(ELEMENT-TYPE=MACRO-ONLY),coplib2,*LINK(coplink))
```

The libraries coplib1 and coplib2 are assigned directly; the library coplib is assigned via the link name coplink.

```
//C C-L=coplib3(S-O)
```

The library coplib3 (element type S) is assigned.

3.1.3.1 Search order for COPY elements

The search order for COPY elements corresponds to the order in which libraries are specified in the COPY-LIBRARY option (see section 2.4.2).

Example

```
//C COPY-LIB=(coplib2,coplib1,coplib3)
```

The first library to be searched for COPY elements is the library coplib2, then coplib1, and finally coplib3.

The type entry is evaluated for each library. If BOTH is specified, the S-partition of the library will be searched first, followed by the M-partition.

If the source or macro partition of a PLAM library contains a number of elements with the same name, but different versions, the element with the highest existing version is always used.

3.2 Outputs of ASSEMBH

The following outputs are generated by ASSEMBH under the control of options:

– Object modules
  Object modules (OMs) are output to a PLAM library (element type R) or to the *EAM file (OMF).

– Link-and-load modules
  Link-and-load modules (LLMs) are output to a PLAM library (element type L).

– Messages
  The start and end messages of ASSEMBH are output to SYSOUT (on the display terminal or to the SYSOUT file).

– Listings
  Listings are output via the LISTING, OUTPUT option (see section 2.4.4) to SYSLST, to a file, or to a PLAM library (element type P). A detailed description of all listings is provided in chapter 6.

– Compiler Information File (CIF)
  The CIF is used for generation of the assembler listing and is created as a temporary file by default. If desired, it can be output to a PLAM library (element type H) by using the COMPILATION-INFO option (see section 2.4.3).

– Monitoring job variable (MONJV)
  If the user has assigned a monitoring job variable for the assembly, the assembler supplies this job variable with a status indicator and return code at the end of the assembly (see section 3.2.3).

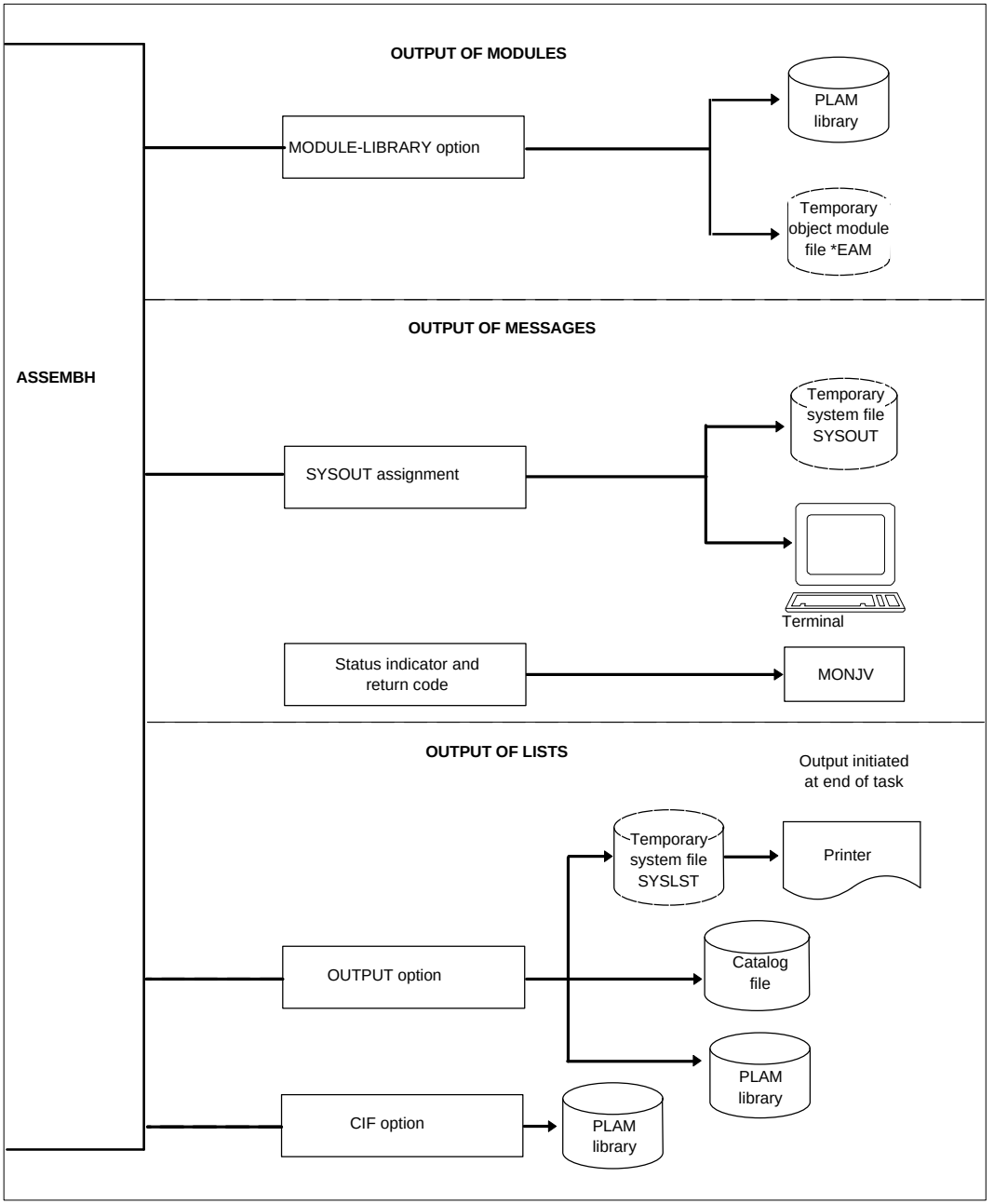


Fig. 3-2: Outputs of ASSEMBH

3.2.1 Output of the object module

The COMPILER-ACTION option (see section 2.4.2.1) generates by default an object module, and the MODULE-LIBRARY option (see section 2.4.2.2) can be used to specify where the object module is to be placed.

Example

```
//C MOD-LIB=plamlib
```

The object module with the name of the first named CSECT is stored in the PLAM library plamlib.

The assembler translates a source program into machine language. This direct result of an assembly is an object module (OM) or link-and-load module (LLM). Although the object module already consists of machine code, it can only be executed after it has been linked and loaded (see chapter 5: "Linking, loading and starting").

The assembly language instructions and statements are converted into machine instructions in accordance with the selected instruction set and output to the object module as TXT entries. A special option enables the selection of different instruction sets (see the SOURCE-PROPERTIES, INSTRUCTION-SET option in section 2.4.1.4).

The associated linkage editor and loader information is stored in ESD and RLD entries.

An object module is normally made up of the ESD, TXT, RLD, and END records. If desired, the TEST-SUPPORT option can be used (see section 2.4.5) to additionally generate LSD entries for symbolic debugging with AID (this function is not supported by ASSEMBH-BC).

Entries in the object module

ESD	Linkage editor and loader information (definition and reference of external symbols)
LSD	Debugging information for AID
TXT	Instructions and statements in machine code
RLD	Linkage editor and loader information (relocation of addresses)
END	End information of the object module

ESD = External symbol dictionary
LSD = List for symbolic debugging
TXT = Text information
RLD = Relocation directory

3.2.2 Output of a link-and-load module

The COMPILER-ACTION option (see section 2.4.21.) can be used to generate a link-and-load module (LLM), and the MODULE-LIBRARY option (see section 2.4.2.2) must then be used to specify the library to which the module is to be output.

Example

```
//C COMP-ACT=(MODULE-FORMAT=LLM),MOD-LIB=plamlib
```

The link-and-load module with the name of the first specified CSECT is stored in the PLAM library plamlib.

The assembler translates a source program into machine language. This direct result of an assembly is an object module (OM) or link-and-load module (LLM). Although the link-and-load module already consists of machine code, it can only be executed after it has been linked and loaded (see chapter 5: "Linking, loading and starting").

The assembly language instructions and statements are converted into machine instructions in accordance with the selected instruction set and output to the link-and-load module as TXT entries. A special option enables the selection of different instruction sets (see the SOURCE-PROPERTIES, INSTRUCTION-SET option in section 2.4.1.4).

The associated linkage editor and loader information is stored in ESV and LRLD entries.

A link-and-load module is normally made up of the ESV, TXT, LRLD and END records. If desired, the TEST-SUPPORT option can be used (see section 2.4.5) to additionally generate LSD entries for symbolic debugging with AID (this function is not supported by ASSEMBH-BC).

Entries in the link-and-load module

ESV	Linkage editor and loader information (definition and reference of external symbols)
LSD	Debugging information for AID
TXT	Instructions and statements in machine code
LRLD	Linkage editor and loader information (relocation of addresses)
END	End information of the link-and-load module

ESV = External symbols vector
LSD = List for symbolic debugging
TXT = Text information
LRLD = Local relocation dictionary

3.2.3 Monitoring the assembly with the monitoring job variable MONJV

The software product JV (Job Variables) permits jobs and programs running under BS2000 to be controlled and monitored (see "JV Job Variables, Reference Manual" [7]). The user defines a "monitoring job variable" that he specifies as an operand of a LOGON, ENTER-JOB or START-PROG command. The operating system enters in this job variable information about the current status of the program ("status indicator") as well as other information defined on the program level ("return code"). The user can query this information at the end of the program. Further jobs and programs may then be controlled on the basis of this information.

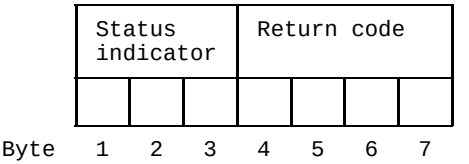
After a program has been assembled with the assembler, the monitoring job variable is supplied with a status indicator and a return code.

If ASSEMBH is called via the subroutine interface, the MONJV value is returned via parameters.

Since multiple assemblies and restarts are allowed, the following applies:
The program-monitoring job variable contains the values from the assembly section in which the highest error weight occurred.

3.2.2.1 Structure of the monitoring job variables

The MONJV value is divided into a status indicator with a length of 3 bytes and a return code with a length of 4 bytes.



The status indicator is entered left-justified in the first three bytes; the return code is set from byte 4 to byte 7.

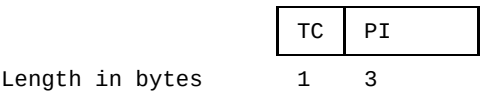
Status indicator

The 3-digit status indicator in the monitoring job variable is set by the assembler as follows:

Status indicator	Termination code TC	
$T_	0 1	Normal termination
$A_	2 3	Abnormal termination

Return code

The 4-digit return code in MONJV is structured as follows:



TC = Termination code
PI = Program information

TC = termination code; may assume the following values:

TC	Explanation
0	Normal termination. No warnings or errors occurred, at the most NOTES.
1	Normal termination. Warnings or errors of class WARNING/SIGNIFICANT/SERIOUS have occurred. (See the table for PI below).
2	Abnormal termination. A termination criterion set by an option was reached. (Max. error weight, max. error number; see section 2.4.6).
3	Abnormal termination. An error of class FATAL, an I/O error, or an assembler error was identified.

PI = program information; may assume the following values:

PI	Explanation	Text:HIGHEST ERROR-WEIGHT: on terminal	in listing
000	No flags and no MNOTES reported. No information messages.	NO ERRORS	-
001	Information messages were output.	NOTES	-
002	Highest error class that occurred	WARNING	0
003	Highest error class that occurred	SIGNIFICANT	1
004	Highest error class that occurred	SERIOUS	2
005	Highest error class that occurred	FATAL	3
006	Assembler error, I/O error	FAILURE	3

Possible combinations

		Assembler error weight						
		-	Inf. mess.	WAR	SIG	SER	FAT	Assem. error
Status indicator	TC	Program information PI						
		000	001	002	003	004	005	006
$T_	0	x	x					
$T_	1			x	x	x		
$A_	2*)			x	x	x		
$A_	3*)						x	x

*) In these cases a branch to a job step is made.

The job variable is initialized by the operating system with the command:

```
/START-PROG $ASSEMBH,MONJV=jvname
```

jvname may be up to 41 characters in length and may consist of all letters, the digits 0 - 9, and the special characters -, @, #, and $.

Example

In the example below, the assembly run is monitored with the job variable JOBVAR. The linkage editor is to be called only if the assembler has reported no errors and no warnings.

```
/BEGIN-PROC LOGG=A,PAR=YES(PROC-PAR=(&PROG),ESC-CHAR=C'&')
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/START-PROG $ASSEMBH,MONJV=JOBVAR  ————————————————————————  (1)
//C SOURCE=(PLAMLIB,&PROG),MOD-LIB=PLAMLIB,LIST=(OUTPUT=(PLAMLIB))
//END
/SET-JOB-STEP
/SHOW-JV JV-ID=NAME(JV-NAME=JOBVAR)  ————————————————————————  (2)
/SKIP-COM TO-LABEL=ENDED,IF=JV(COND=(JOBVAR,4,4)>'0001')  ————————  (3)
/START-PROG $TSOSLNK
PROG &PROG,LIB=PLAMLIB
INCLUDE &PROG,PLAMLIB
END
/SKIP-COM TO-LABEL=ENDED
/SET-JOB-STEP
/.ENDED END-PROC
```

(1) The START-PROG command assigns JOBVAR to the assembler as the program-monitoring job variable.

(2) The SHOW-JV command displays the value of the job variable.

(3) The SKIP-COM command is used to check whether the return code (bytes 4 - 7) contains a value greater than '0001'. If this is the case, the assembler has reported errors in the error class "warnings" or above, and the procedure branches to the label ".ENDED".

4 Runtime system for structured programming

4.1 General information

Structured programming in assembler requires the services of a runtime system (see "ASSEMBH (BS2000) Reference Manual" [1]) in order to run a program. The runtime system performs register saving and reserves and releases storage space for saving registers, and for the automatic and controlled areas.

This runtime system is provided as the module IASSRTS in the library SYSLIB.ASSEMBH.012.
The requesting and releasing of memory for register saving, automatic and controlled areas are based on ILCS (Inter-Language Communication Services) routines. The ILCS initialization module IT0INITS from the SYSLIB.ASSEMBH.012 library loads the required ILCS routines dynamically.
The ILCS module IT0ENTR is also provided in this library; this module contains all the entries of the ILCS routines apart from IT0INIT(S).

Memory management is completely dynamic.

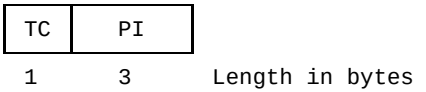
Assembler objects (with structured programming) that were assembled with the COLUMBUS Assembler V2.2F are compatible with and executable on the new runtime system as well.

4.2 Support for monitoring job variables

The new runtime system supports the use of monitoring job variables by transferring appropriate return codes to the ILCS routines in the case of normal as well as abnormal program termination. These return codes are passed on from there to the job variable.

In some error situations in which the ILCS has not yet been initialized, the monitoring job variables are set directly in the runtime system.

The runtime system creates the 4-digit return code for MONJV in the following format:



TC = termination code, with the following values:

TC	Explanation
0	Normal termination
3	Abnormal termination

PI = program information, with the following values:

PI	Explanation
000	No errors occurred during the program run.
001 . . . nnn	Errors occurred during the program run. The 3-digit numbers correspond to the 3 characters to the right of the messages of the runtime system (ASS7nnn, see section 11.1).

The following combinations are possible:

$T_0000 Normal program termination
$A_3001 Abnormal termination
.
.
.
$A_3nnn

5 Linking, loading and starting

5.1 General information

When a source program is assembled, one or more object modules or linkand-load modules are obtained as a result. The object modules generated by the assembler are placed in the temporary EAM file (OMF) of the current task or saved as elements (of type R) in a PLAM library. The link-and-load modules generated are saved in a PLAM library as (type L) elements (see section 3.2,
Outputs of ASSEMBH"). Although these modules already consist of machine code, they must first be linked into a load module in order to produce an executable program. Before the linked program can be executed, it must be loaded into working memory. The executable program is therefore called a "load module" (i.e. module to be loaded).

It is also possible to link in other modules, e.g. separately assembled source programs or subroutines in other languages. These additional modules may have been assembled or compiled at different times with different compilers.

The most important function of the linkage editor is to call the modules required for the executable unit from various sources (files, libraries) and link them to one another. Linking means that the linkage editor adds to each module those addresses which refer to areas outside the module (external references).

Various utilities are available in BS2000 for the tasks related to linking and loading:

- **Linkage editor BINDER (as of BS2000 V10.0)**

BINDER (see section 5.2) links object modules (OMs) and link-and-load modules (LLMs) to form a logically and physically structured loadable unit. This unit is known as a "link-and-load module" (LLM). BINDER stores an LLM as a type L element in a PLAM library.

- **Dynamic Binder Loader DBL (as of BS2000 V10.0)**
  (Dynamic Linking Loader DLL (up to BS2000 V9.5)

The Dynamic Binder Loader DBL (see section 5.3) links object modules (OMs) and link-and-load modules (LLMs) into a temporary program, loads it immediately into memory, and initiates the program run, all in a single operation. The program is automatically deleted after execution. DBL is primarily suitable for use in the debugging phase.

- **Static linkage editor TSOSLNK**

The static linkage editor TSOSLNK (see section 5.4) links object modules and stores the generated executable program (also called a "load module") in a cataloged file or in a PLAM library (element type C).

- **Static loader ELDE**

The static loader ELDE (see section 5.5) serves to load an executable program linked by means of TSOSLNK.

5.2 Linking with BINDER

BINDER links object modules (OMs) and link-and-load modules (LLMs) to form an LLM, which it stores as a type L element in a PLAM library. It is described in detail in the BINDER manual [10].

The object modules generated by ASSEMBH are saved either in the EAM file of the current task or as type R elements in a PLAM library.
The LLMs are saved as type L elements in a PLAM library.

Control statements for BINDER (selection)

```
/START-PROGRAM $BINDER                                                    (1)
 START-LLM-CREATION INT-NAME=name                                         (2)
 INCLUDE-MODULES LIB= {library} ,ELEM= {element}                          (3)
                     {*OMF    }       {*ALL   }
[INCLUDE-MODULES LIB=..., ELEM=...]                                       (4)
[RESOLVE-BY-AUTOLINK LIB=SYSLIB.ASSEMBH.012                               (5)
[RESOLVE-BY-AUTOLINK LIB=..., [SYMBOL-NAME=external-xref]]                (6)
[MODIFY-SYMBOL-VISIBILITY ..., VISIBLE=NO]                                (7)
 SAVE-LLM LIB=library, ELEM=element                                       (8)
 END                                                                      (9)
```

(1) BINDER is called.

(2) This statement generates a new LLM with the internal name "name" in the work area. The SAVE-LLM statement (see section 8) is used to save the generated LLM as a type L element in a PLAM library.

(3) library is the name of the PLAM library containing the modules. *OMF is the name of the EAM file.
element is the name of a module.
If *ALL is specified, all modules from the specified input source are linked in.

(4) An additional INCLUDE-MODULE statement can be used to link in additional modules from different libraries.

(5) The ASSEMBH runtime system (if you want to use structured programming) is linked in by means of RESOLVE-BY-AUTOLINK.

(6) Further RESOLVE-BY-AUTOLINK statements are used to inform BINDER of the external references (= module names) and the corresponding libraries or only those libraries to be searched for as yet unresolved external references using the autolink method.

(7) The MODIFY-SYMBOL-VISIBILITY statement can be used to mask out further BINDER runs. By default the symbols remain visible. See the section entitled 'Masking out symbols'.

(8) This statement saves the current LLM, which was generated by means of START-LLM-CREATION as a type L element in a PLAM library.

(9) The END statement is used to terminate the BINDER run.

With the INCLUDE-MODULES and RESOLVE-BY-AUTOLINK statements, LIB=*BLS-LINK can be specified instead of the library name (LIB=library). In this case, the libraries to be searched must be assigned the link name BLSLIBnn (00 ≤ nn ≤ 99). This happens before the BINDER is called using the ET-FILE-LINK command, e.g.:

```
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=SYSLIB.ASSEMBH.012
```

Provided all external references have been resolved, an LLM generated using BINDER can be loaded and started with the DBL without assigning alternative libraries:

```
START-PROGRAM *MODULE(LIB=library,ELEM=module,RUN-MODE=ADVANCED)
```

Symbol masking

Unlike with TSOSLNK, symbols (CSECTs, ENTRYs) are not masked out by default when BINDER is used for linkage. They then remain visible for subsequent linkage runs with BINDER or DBL.

This has the following effects during dynamic linkage with DBL: If a PLAM library contains individual modules generated by ASSEMBH and LLMs with a runtime system linked in, the external references to the runtime system are resolved from one of the prelinked modules during dynamic linkage and not from the runtime library. In this event, DBL issues a number of "DUPLICATES" warnings. The autolink mechanism first searches the library in which the individual module is located and then the runtime libraries assigned using the link name BLSLIBnn.

We recommend the following procedure to ensure that the external references are always resolved from the current runtime library rather than from one of the other modules during linkage:
– either store individual modules and prelinked modules in separate libraries
– or mask out the symbols with the MODIFY-SYMBOL-VISIBILITY statement

5.3 Dynamic linking and loading with DBL

With the Dynamic Binder Loader (DBL), object modules (OMs) and link-and-load modules (LLMs) are temporarily linked into a program, loaded into memory, and then executed. All three steps are performed in a single run. The generated program is automatically deleted after program execution.
The operation of DBL is described in detail in the manual "Binder-Loader-Starter" [9].

DBL has two modes of operation, or "run modes". The desired mode is selected with the RUN-MODE operand of the START-PROGRAM and LOAD-PROGRAM commands.

RUN-MODE=STD (default)
In this mode, DBL is compatible with DLL up to BS2000 V9.5 inclusive. Only object modules can be processed, not link-and-load modules (LLMs).

RUN-MODE=ADVANCED
In this mode, object modules and link-and-load modules (LLMs) can be processed. This mode is not described in the present manual. A detailed description may be found in the manual "Binder-Loader-Starter" [9].

The modules generated by the assembler are either placed in the temporary EAM file of the current task or entered as elements (of type R or L) in a PLAM library.

If object modules from the EAM file are to be linked, this file must be deleted before assembly by using the DEL-SYS-FILE OMF command.

The linkage run with DBL is initiated with the command START-PROG or LOAD-PROG. After the START-PROG command, the program is executed immediately. After LOAD-PROG, further commands (e.g. debugging aid commands) may be entered. In this case the program can subsequently be started with the RESUME-PROG command.

Commands for DBL

```
/{START-PROG }  [FROM-FILE=] *MODULE (LIB= {*OMF [,ELEM=*ALL]          }
 {LOAD-PROG  }                             {*OMF ,ELEMENT=module      }
                                           {                          }
                                           {library,ELEM=module       }
                                           {[,RUN-MODE=STD/ADVANCED]  }
```

*OMF designates the temporary EAM file (OMF) in which the assembler has placed the object module.

module Name of the module to be loaded.

library Name of the PLAM library which contains the module (OM/LLM) as an element with the name "module". This module must be of type R/L. If several elements with the same name are stored in the library, the element with the highest version is taken.

RUN-MODE=ADVANCED
This specification is required whenever link-and-load modules (LLMs) are to be processed.

5.4 Static linking with TSOSLNK

The static linkage editor TSOSLNK can be used to link object modules into a program and to save this program in a cataloged file or as an element (of type C) in a PLAM library.

Control statements for TSOSLNK

```
/START-PROG $TSOSLNK ──────────────────────────────────────────── (1)

*PROGRAM program  [,{FILENAM=file                }] ──────────── (2)
                    {LIB=library [,ELEM=element]}

*INCLUDE {module,library / *       } ──────────────────────────── (3)
         {(module,...),library / * }

[*RESOLVE [external-refs],library] ──────────────────────────── (4)

*END ────────────────────────────────────────────────────────── (5)
```

(1) The static linkage editor TSOSLNK is called.

(2) The PROGRAM statement defines where the program is to be stored.

program	The name to be given to the program must be entered here. If no further operand ("FILENAM or "LIB") has been specified, the name will be assigned to the cataloged file.
FILENAM=file	The "file" entry selects a name to be assigned to the cataloged file. The max. length including the cat-id and user-id must not exceed 54 characters.
LIB=library, ELEM=element	The program is stored under the name "element" in the named PLAM "library" as an element of type C. If only the "LIB" operand is specified, "program" will be assumed as the element name.

(3) The INCLUDE statement can be used to link in one or more modules from a library. Multiple modules entered in a list must be enclosed within parentheses. An asterisk (*) can be specified as the library name to designate the EAM file (OMF). Modules from different libraries can be linked by means of a sequence of INCLUDE statements.

(4) The RESOLVE statement indicates to the linkage editor the external references (= object module names) and the corresponding libraries (or just the libraries) that are to be searched with the autolink procedure (described below) for still unresolved external references.

(5) Inputs to the linkage editor TSOSLNK must be terminated with the END statement.

Autolink procedure of TSOSLNK

If the TSOSLNK linkage editor finds external references in an object module which cannot be resolved with the modules that were specified in INCLUDE statements, it will proceed according to the following autolink procedure:

– TSOSLNK will first search the library that was explicitly specified in the RESOLVE statement in connection with the external reference.

– If the external reference cannot be resolved by TSOSLNK in the first step, all libraries specified in RESOLVE statements are searched. The search proceeds in reverse order, i.e. the last RESOLVE statement is processed first, the next-to-last second etc.
Libraries that are not to be searched can be excluded by means of EXCLUDE statements.

– If the external reference cannot be resolved in the second step either, TSOSLNK will search the library TASKLIB, provided this has not been prevented with the NCAL statement or a corresponding EXCLUDE statement. If there is no library named TASKLIB under the user ID of the current task, TSOSLNK will use the library of the system, i.e. $TSOS.TASKLIB.

If unresolved external references remain even after the autolink procedure, TSOSLNK will output their names to SYSOUT and SYSLST in the form of a listing.

Example of a linkage run with TSOSLNK and starting with ELDE

The object modules created from the separate assembly of two program segments are to be linked into a single program.
The modules PROG1 and UP1 are located in the library PLAMLIB.

```
/START-PROG $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0E00' OF '1992-01-07' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1991. ALL
   RIGHTS RESERVED
PROGRAM PROG1,FILENAM=TESTASS
INCLUDE PROG1,PLAMLIB
INCLUDE UP1,PLAMLIB
END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: TESTASS
%  LNK0504 NUMBER PAM PAGES USED:      3
/START-PROG TESTASS
%  BLS0500 PROGRAM 'PROG1', VERSION ' ' OF '90-05-24' LOADED.
HERE IS PROG1
HERE IS UP1
HERE IS PROG1 AGAIN
```

5.5 Loading and starting programs using the loader ELDE

In order to run a program that has been linked, it must first be loaded into main memory. The static loader ELDE is provided in BS2000 for this purpose. Like DBL, the ELDE loader is invoked implicitly by the START-PROG and LOAD-PROG commands:

– The START-PROG command instructs ELDE to load the program into memory and start it. Since the program run is initiated immediately after loading, the files required by the program must be assigned beforehand.

– The LOAD-PROG command instructs ELDE to load the program into memory without starting it. This enables the input of further commands (e.g. for debugging) before the program run. The program itself can be subsequently started with the RESUME-PROG command.

The most important entries for the START-PROG and LOAD-PROG commands are indicated below. A detailed description of both commands can be found in the manual "BS2000/OSD-BC Commands" [6]. Refer to the previous page for an example.

```
/{LOAD-PROG  } FROM-FILE = {filename                                        }
 {           }             {                                              }
 {START-PROG }             {*PHASE(LIB=library,ELEM=module,VERS=version)  }
```

filename	Name of the cataloged file which contains the program generated by TSOSLNK.
library	Name of the PLAM library that contains the program generated by TSOSLNK, as an element of type C.
module	Name of the library element under which the program is stored.
version	Version of the library element, up to 24 characters.

5.6 Assembling and linking a structured assembler program

When assembling the source, the library SYSLIB.ASSMBH.012, which contains the macros for structured programming, must be specified as the macro library.

```
/START-PROGRAM  $ASSEMBH
//COMPILE SOURCE=sourcefile,-
//       MACRO-LIBRARY=SYSLIB.ASSEMBH.012,-
//       MODULE-LIBRARY=module-library
//END
```

When linking the program, the SYSLIB.ASSEMBH.012 library must be specified in order to link in the assembler runtime system.

```
/START-PROGRAM  $TSOSLNK
 PROG structured-program,...
 INCLUDE structured-program, module-library
  .
  .
  .
 RESOLVE,SYSLIB.ASSEMBH.012
 END
```

5.7 XS support

As of version V9.0, BS2000 supports not only the usual hardware, but also XS systems (XS stands for e**X**tended **S**ystem). These systems provide the user with considerably extended virtual address space: in contrast to the 16 megabytes available on previous non-XS systems, up to 2 gigabytes can be addressed on an XS system.

This extension of address space on XS systems is enabled by the fact that 31 bits of an address word are used to form an address instead of 24 bits (as with non-XS systems).

Details on XS programming are available in the manual "Introductory Guide to XS Programming" [3].

5.8 ESA support

A new addressing mode for expanding the virtual address space is supported as of BS2000 V11.0. This extended addressing mode is only available on systems which include the appropriate new hardware (e.g. H130). These so-called ESA (Enterprise Systems Architecture) systems provide additional address space for data.
ESA systems allow you on the one hand to work with 24-bit or 31-bit addresses and, on the other, to work with data areas or in the program area only (see the "Executive Macros" User Guide [12]).
The ASSEMBH assembler (≥ V1.2A) supports the ESA instructions by means of the INSTRUCTION-SET = BS2000-ESA operand of the SOURCE-PROPERTIES option (see section 2.4.1.4).
The ESA commands are listed in the Appendix, section 11.3 and described in the "Assembler Instructions (BS2000)" Language Reference Manual [11].

6 Description of listings

The LISTING option (see section 2.4.4) can be used to define the layout, scope, and output location for listings.

Listings are not generated directly by ASSEMBH, but are produced by a listing generator.

Listings can also be created via the standalone generator ASSLG (see section 2.5), provided the CIF (Compiler Information File) was stored in a library by specifying the COMPILATION-INFO option (see section 2.4.3) when assembling the source.

Listings can be created in five different formats:

- Listings in standard format (ASSEMBH)
- Listing compatible with ASSEMB V30
- Laser printer listing
- SAVLST (listing with ISAM key)
- Structured listings

6.1 Listings in standard format

During the assembly, ASSEMBH generates an assembler listing (consisting of the individual listings described below).
Depending on the values specified in the LISTING option, the following specific listings can be created:

- an options listing (OPTIONS LISTING); this listing is always produced.
- an ESD listing (EXTERNAL SYMBOL DICTIONARY)
- a source program listing (SOURCE LISTING)
- a listing of used files and libraries
- cross-reference listings (XREF LISTINGS)

6.1.1 Options listing (OPTIONS LISTING)

The options listing contains all the COMPILE statement options and related operands and operand values which are valid for the current assembly.
This listing is always created during an assembly, which means that its output cannot be suppressed.

With *COMOPT control, the used options are output as with SDF control. The entered COMOPTs are indicated in an additional listing (see section 6.2, "Listing compatible with ASSEMB V30").

If the listing is generated with the standalone list generator, the valid operands of the GENERATE statement are contained in an additional list which precedes the options listing.

```
ASSEMBH LISTING                                                                    10:36:24  1994-03-07   PAGE 0001

SOURCE=:01KH:$HASSEMB.MES.XREF.ENGL,

MACRO-LIBRARY=MES.PLAM,

COPY-LIBRARY=MES.PLAM
  (ELEMENT-TYPE=BOTH),

SOURCE-PROPERTIES=PARAMETERS
  (FROM-COLUMN=1,TO-COLUMN=71,CONTINUATION-COLUMN=16,LOW-CASE-CONVERSION=NO,INSTRUCTION-SET=BS2000-XS,
  PREDEFINED-VARIABLES=NONE),

COMPILER-ACTION=MODULE-GENERATION
  (MODE=STD,MODULE-FORMAT=OM),

MODULE-LIBRARY=MES.PLAM
  (ELEMENT=*STD
    (VERSION=*UPPER-LIMIT)),

COMPILATION-INFO=NONE,

LISTING=PARAMETERS
  (SOURCE-PRINT=WITH-OBJECT-CODE
    (PRINT-STATEMENTS=ACCEPTED,LINE-NUMBERING=NO),
  SOURCE-FORMAT=STD,
  MACRO-PRINT=PARAMETERS
    (NOPRINT-NEST-LEVEL=255,NOPRINT-PREFIX=*NONE,TITLE-STATEMENTS=IGNORED,MACRO-ORIGIN-INFO=SEPARATE),
  MIN-MESSAGE-WEIGHT=SIGNIFICANT,CROSS-REFERENCE=PARAMETERS
    (SYMBOL=YES
      (WITH-ATTRIBUTES=YES,REFERENCED-ONLY=NO,PREFIX=ALL),
    LITERAL=YES,MACRO=YES,COPY=YES,DIAGNOSTICS=YES),
  EXTERNAL-DICTIONARY=YES,LAYOUT=PARAMETERS
    (LINES-PER-PAGE=60,LASER-PRINTER=NO,FORMAT=STD),
  OUTPUT=MES.LIST.XREF),

TEST-SUPPORT=NO,

COMPILER-TERMINATION=PARAMETERS
  (MAX-ERROR-WEIGHT=FATAL,MAX-ERROR-NUMBER=32767,MAX-MACRO-NEST-LEVEL=255,MAX-COPY-NEST-LEVEL=5),

CORRECTION-CYCLE=NO,

MAINTENANCE-OPTIONS=PARAMETERS
  (CHANNEL-INSTRUCTIONS=NO),

COMPILATION-SPACE=STD
```

6.1.2 ESD listing (EXTERNAL SYMBOL DICTIONARY)

The ESD listing is a listing of definitions and references to external names (symbols) for:

- control sections (CSECT, including AMODE and RMODE)
- common control sections (COM)
- dummy sections (DSECT, and the external dummy sections XDSEC)
- dummy registers (DXD)
- entry points to own assembly unit (ENTRY)
- entry points or address references to other assembly units (V-type constants, EXTRN, WXTRN)

The logged ESD information corresponds to the ESD records which are generated during the assembly and placed in the module. This information is required by the linkage editor and loader in order to link modules into executable programs.
The ESD listing is created by default. Its output can be suppressed with the option LISTING(EXTERNAL-DICTIONARY=NO).

Key to columns in the ESD listing:

Column **Meaning**

SYMBOL External name
This is either specified by the user in the appropriate statements (see TYPE column) or it is generated by ASSEMBH.
Unnamed CSECTs and COMs are listed as %CSECT and %COM respectively.
External names that are processed by the linkage editor are restricted to eight characters. Longer external names are truncated to eight characters for further processing, and are provided with a message.
If modules are output in LLM format, a maximum of 32 characters are permitted (see section 6.6).

TYPE Type of external name

CM Name of a common control section
(CM ≙ Common Memory; COM statement).
An unnamed common control section is listed as %COM in the SYMBOL column.

DS Name of a dummy section
(DS ≙ Dummy Section; DSECT statement).
In addition, this line has the designation (DUMMY) ahead of the SYMBOL column.

DX Name of a dummy register
(DXD statement).

ER Name of an external linkage address
(ER ≙ External Reference; EXTRN statement).

LD Name of a linkage address
(LD ≙ Label Definition; ENTRY statement).

SD Name of a control section
(SD ≙ Section Definition; CSECT or START statement).
An unnamed control section is listed as %CSECT in the SYMBOL column.

VC Name of an external linkage address
(VC ≙ V-Constant).

XD Name of an external dummy section
(XDSEC statement with operand D).
In addition, this line has the designation (DUMMY) ahead of the SYMBOL column.

XR Name of the reference for an external dummy section
(XDSEC statement with operand R).
In addition, this line has the designation (DUMMY) ahead of the SYMBOL column.

WX Name of a conditional external linkage address
(Weak External Reference; WXTRN statement).

ID Number of the external name
(ID ≙ Identification).
The external names are numbered consecutively for each module, starting at 0001.

ADDR Displacement from start of module if the module is in OM format or displacement from the start of the corresponding CSECT for a module in LLM format (see section 6.6) for definition of external names.
This displacement is given hexadecimally in bytes.

LENGTH Length of a control section or common control section (hexadecimal, in bytes).
No length specification is given for V-type constants and linkage addresses.

A/R-MODE In the left column (A-MODE), the addressing mode (24/31/ANY) for a control section is listed (AMODE statement).

In the right column (R-MODE), the load attribute (24/ANY) for a control section is listed (RMODE statement).

```
ASSEMBH LISTING                                                                                  10:36:24  1994-03-07   PAGE 0002
          SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE                EXTERNAL SYMBOL DICTIONARY
          TESTXREF  SD 0001 00000000 000068  24  24
          ADDRCOM   VC 0002
  (DUMMY) BEGIN     DS 0003 00000000 000020
          ADDRCOM   SD 0004 00000068 000020  24  24
          HCOM      CM 0005 00000000 000009  24  24
          %CSECT    SD 0006 00000088 000002  24  24
          %COM      CM 0007 00000000 000008  24  24
```

6.1.3 Source program listing (SOURCE LISTING)

The output of the source program listing is controlled by means of the LISTING(SOURCE-PRINT=) option.

The source program listing normally contains the source program and the object code. A message with the total number of errors is printed at the end of the listing. This is followed by the end message of the assembler with an indication of the version, date and time.

Key to columns in the ESD listing:

Column	Meaning
LOCTN	Location counter, hexadecimal (3 bytes).
OBJECT CODE	Object code, hexadecimal (6 bytes).
ADDR1	Address of first operand, hexadecimal (4 bytes).
ADDR2	Address of second operand, hexadecimal (4 bytes).
STMNT	Consecutive line number, starting at 1.
M	One digit, denoting the nesting depth of macros and COPYs: 1 Level 1 2 Level 2 etc. + means that these instructions have been generated by macro statements in the source program.
SOURCE STATEMENT	Source program text A line in the source program can comprise five entries. These are, from left to right: Names, operations, operands, remarks and continuation character.

In the case of a module in LLM format, see section 6.6 for the contents of the address fields LOCTN, ADDR1 and ADDR2.

Error indication

The assembler generates diagnostic messages in the event of errors during assembly (see section 11.1). Such messages follow the lines to which they relate.
The message line begins with an * and may appear as follows:

```
*     U10  *** ERROR *** ASS2110 SYMBOL UNDEF IS UNDEFINED
       |                    |
       v                    v
     Flag               Message number
```

```
ASSEMBH LISTING                                                                        10:36:24  1994-03-07   PAGE 0003
LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
000000                                      1      TESTXREF START
                                            2               PRINT NOGEN
                                            3      *
                                            4               COPY  MES.EQU
                      00000005              5    1 R5       EQU   5
                      00000006              6    1 R6       EQU   6
                      00000007              7    1 R7       EQU   7
                      00000008              8    1 R8       EQU   8
                      00000009              9    1 R9       EQU   9
                      0000000A             10    1 R10      EQU   10
                      0000000E             11    1 R14      EQU   14
                      0000000F             12    1 R15      EQU   15
                                           13      *
000000 05 50                               14      BEG      BALR  R5,0
000002                00000002             15               USING *,R5
000002 D2 02 50505053 00000052 00000055    16               MVC   FIELD,NUMBER
000008 47 F0 0000                          17               B     UNDEF
*    U10  *** ERROR *** ASS2110 SYMBOL UNDEF IS UNDEFINED
00000C D2 02 5050505E 00000052 00000060    18               MVC   FIELD,=C'456'
000012 47 F0 5053     00000055             19               B     NUMBER
*    D7   *** ERROR *** ASS0407 ALIGNMENT ERROR IN OPERAND 1
                                           20      *
                                           21               MNOTE 152,'BRANCH ADDRESS IS WRONG'
                                           22      *
000016 41 60 5030     00000032             23               LA    R6,INPUT
00001A                00000000             24               USING BEGIN,R6
00001A 58 F0 5056     00000058             25               L     R15,=V(ADDRCOM)
00001E 05 EF                               26               BALR  R14,R15
                                           27      *
000020                                     28               TERM
                                           31   2                 *,VERSION 010                                  00001300
                                           43      *
000032                                     44      INPUT    DS    CL32
000052                                     45      FIELD    DS    CL3
000055 F1F2F3                              46      NUMBER   DC    C'123'
                                           47      *
000000                                     48      BEGIN    DSECT
000000                                     49      NR       DS    CL2
000002                                     50      NAME     DS    CL10
00000C                                     51      STREET   DS    CL20
                                           52      *
000068                                     53      ADDRCOM  CSECT
000068 05 70                               54               BALR  R7,0
00006A                0000006A             55               USING *,R7
00006A 58 80 505A     0000005C             56               L     R8,=A(HCOM)
00006E                00000000             57               USING HCOM,R8
00006E D2 04 80045061 00000004 00000063    58               MVC   COM2,=C'12345'
000074                                     59               TERM
                                           62   2                 *,VERSION 010                                  00001300
                                           74      *
000000                                     75      HCOM     COM
000000                                     76      COM1     DS    F
000004                                     77      COM2     DS    CL5
                                           78      *
000088                                     79               CSECT
```

```
ASSEMBH LISTING                                                                    10:36:24  1994-03-07   PAGE 0004
LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
000088 05 90                                80             BALR  R9,0
00008A                0000008A              81             USING *,R9
                                            82     *
000000                                      83             COM
000000                                      84     COM3    DS    F
000004                                      85     COM4    DS    F
                                            86     *
000000                                      87             END   BEG
000058 00000000                             88                   =V(ADDRCOM)
00005C 00000000                             89                   =A(HCOM)
000060 F4F5F6                               90                   =C'456'
000063 F1F2F3F4F5                           91                   =C'12345'
FLAGS IN 00002 STATEMENTS, 000 PRIVILEGED FLAGS, 001 MNOTES
HIGHEST ERROR-WEIGHT : SERIOUS ERROR
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-07 AT 10:33:03
```

6.1.4 Listing of files and libraries used

This listing shows from where the source was obtained, which module was generated, and the macro and COPY libraries that were used.

```
ASSEMBH LISTING                                                                    10:36:24  1994-03-07   PAGE 0005
USED FILES AND LIBRARIES
SOURCE FILE   :   :01KH:$HASSEMB.MES.XREF.ENGL
MODULE LIBRARY  :      :01KH:$HASSEMB.MES.PLAM
MODULE ELEMENT  :      TESTXREF
VERS/DATE       :      @/1994-03-07
MACRO-LIBRARIES     LINKNAME  LIBRARY-NAME
                              MES.PLAM
                              :D:$TSOS.MACROLIB
COPY-LIBRARIES      LINKNAME  LIBRARY-NAME
                              :01KH:$HASSEMB.MES.PLAM
```

6.1.5 Cross-reference listings

Cross-reference listings show, in ascending order, the locations in the source program for:

- symbols (SYMBOL-XREF)
- literals (LITERAL-XREF)
- names of macros (MACRO-XREF)
- names of COPY elements (COPY-XREF)
- undefined symbols (UNDEFND SYMBOL-XREF)
- errors detected by the assembler, and user-own messages (DIAGNOSTIC-XREF: FLAG-XREF and MNOTE-XREF)

The FLAG-XREF and the MNOTE-XREF are created by default. All other cross-reference listings may be requested by means of the option LISTING(CROSS-REFERENCE).

When a SYMBOL-XREF is requested, the UNDEFND SYMBOL-XREF is also generated. By default, the attribute associated with the symbol is also shown in the 'REFERENCES' column of both listings. These attributes refer to the mode of access.
The following attributes are possible:

A : Address access
E : EQU / ORG instructions
R : Read-only access by instructions
W : Write access

In the UNDEFND SYMBOL-XREF, unnamed CSECTs and COMs are listed as %CSECT and %COM respectively.

In the case of a module in LLM format, see section 6.6 for the contents of the address field VALUE for the SYMBOL-XREF and LITERAL-XREF.

```
ASSEMBH LISTING                                                                 10:36:24  1994-03-07   PAGE 0006
SYMBOL    LEN  VALUE    DEFN    REFERENCES
%COM     00008 00000000 000083
%CSECT   00002 00000088 000079
ADDRCOM  00032 00000068 000053
ADDRCOM  00000 00000000 000000  000025A
BEG      00002 00000000 000014  000087
BEGIN    00032 00000000 000048  000024  000048
COM1     00004 00000000 000076
COM2     00005 00000004 000077  000058W
COM3     00004 00000000 000084
COM4     00004 00000004 000085
FIELD    00003 00000052 000045  000016W 000018W
HCOM     00009 00000000 000075  000056A 000057  000075
INPUT    00032 00000032 000044  000023A
NAME     00010 00000002 000050
NR       00002 00000000 000049
NUMBER   00003 00000055 000046  000016R 000019A
R10      00001 0000000A 000010
R14      00001 0000000E 000011  000026W
R15      00001 0000000F 000012  000025W 000026R
R5       00001 00000005 000005  000014W 000015
R6       00001 00000006 000006  000023W 000024
R7       00001 00000007 000007  000054W 000055
R8       00001 00000008 000008  000056W 000057
R9       00001 00000009 000009  000080W 000081
STREET   00020 0000000C 000051
TESTXREF 00104 00000000 000001
```

```
ASSEMBH LISTING                                                                 10:36:24  1994-03-07   PAGE 0007
LITERAL   LEN  VALUE    DEFN    REFERENCES
=A(HCOM)
        00004 0000005C 000089  000056
=C'12345'
        00005 00000063 000091  000058
=C'456'
        00003 00000060 000090  000018
=V(ADDRCOM)
        00004 00000058 000088  000025
```

```
ASSEMBH LISTING                                                                 10:36:24  1994-03-07   PAGE 0008
MACRO-NAME                                          LINKNAME TYPE     VERSION                 DATE       DEF-STMNT
         LIBRARY-NAME/SOURCE-NAME                   ELEMENT-NAME
         REFERENCES
##BAL                                                        M        010                     1988-06-14
         :D:$TSOS.MACROLIB                          ##BAL
         000034  000065
#INTF                                                       M        919                     1987-12-11
         :D:$TSOS.MACROLIB                          #INTF
         000029  000060
IDLKG                                                       M        002                     1987-12-11
         :D:$TSOS.MACROLIB                               IDLKG
         000030  000061
TERM                                                        M        010                     1988-06-15
         :D:$TSOS.MACROLIB                               TERM
         000028  000059
```

```
ASSEMBH LISTING                                                                 10:36:24  1994-03-07   PAGE 0009
COPY-NAME LIBRARY-NAME                              LINKNAME TYPE     VERSION                 DATE
         REFERENCES
MES.EQU   :01KH:$HASSEMB.MES.PLAM                                S        @                       1992-02-28
         000004
```

```
ASSEMBH LISTING                                                    10:36:24  1994-03-07   PAGE 0010
UNDEFND-SYMBOL REFERENCES
UNDEF          000017A



ASSEMBH LISTING                                                    10:36:24  1994-03-07   PAGE 0011
                                       DIAGNOSTICS
FLAG  MESSAGE AND STATEMENT NUMBERS
D7    ASS0407 ALIGNMENT ERROR IN OPERAND
                000019
U10   ASS2110 SYMBOL   IS UNDEFINED
                000017



ASSEMBH LISTING                                                    10:36:24  1994-03-07   PAGE 0012
                                       DIAGNOSTICS
       SEVERITY CODES OF MNOTES AND STATEMENT NUMBERS
       MNOTE WITH SEVERITY CODE 0152          000021
```

6.1.6 End message

Assembly time : Time required for an assembly, excluding the time to generate the listing.

End message of the listing generator with indication of version:

```
ASSEMBLY TIME :          0.543     SEC.
THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

6.2 Listing compatible with ASSEMB V30

The option LISTING=PAR(LAYOUT=PAR(FORMAT=F-ASSEMB-COMPATIBLE)) generates a listing that is compatible with ASSEMB V30.

Listings generated under *COMOPT control are always compatible with ASSEMB V30. The specified COMOPTs are listed in an additional options listing (USER'S OPTIONS).

```
ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0001

SOURCE=:01KH:$HASSEMB.MES.TEST1F.ENGL,

MACRO-LIBRARY=*NONE,

COPY-LIBRARY=*NONE,

SOURCE-PROPERTIES=PARAMETERS
  (FROM-COLUMN=1,TO-COLUMN=71,CONTINUATION-COLUMN=16,LOW-CASE-CONVERSION=NO,INSTRUCTION-SET=BS2000-NXS,
  PREDEFINED-VARIABLES=NONE),

COMPILER-ACTION=MODULE-GENERATION
  (MODE=F-ASSEMB-COMPATIBLE,MODULE-FORMAT=OM),

MODULE-LIBRARY=MES.PLAM
  (ELEMENT=*STD
    (VERSION=*UPPER-LIMIT)),

COMPILATION-INFO=NONE,

LISTING=PARAMETERS
  (SOURCE-PRINT=WITH-OBJECT-CODE
    (PRINT-STATEMENTS=ACCEPTED,LINE-NUMBERING=NO),
  SOURCE-FORMAT=STD,
  MACRO-PRINT=PARAMETERS
    (NOPRINT-NEST-LEVEL=255,NOPRINT-PREFIX=*NONE,TITLE-STATEMENTS=ACCEPTED,MACRO-ORIGIN-INFO=SEPARATE),
  MIN-MESSAGE-WEIGHT=SIGNIFICANT,CROSS-REFERENCE=PARAMETERS
    (SYMBOL=YES
      (WITH-ATTRIBUTES=NO,REFERENCED-ONLY=NO,PREFIX=ALL),
    LITERAL=YES,MACRO=YES,COPY=NO,DIAGNOSTICS=YES),
  EXTERNAL-DICTIONARY=YES,LAYOUT=PARAMETERS
    (LINES-PER-PAGE=60,LASER-PRINTER=NO,FORMAT=F-ASSEMB-COMPATIBLE
      (MESSAGE-PLACEMENT=SEPARATE)),
  OUTPUT=*SAVLST-AND-SYSLST),

TEST-SUPPORT=YES,

COMPILER-TERMINATION=PARAMETERS
  (MAX-ERROR-WEIGHT=FATAL,MAX-ERROR-NUMBER=32767,MAX-MACRO-NEST-LEVEL=255,MAX-COPY-NEST-LEVEL=5),

CORRECTION-CYCLE=NO,

MAINTENANCE-OPTIONS=PARAMETERS
  (CHANNEL-INSTRUCTIONS=NO),

COMPILATION-SPACE=STD



ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0002
          *** USER'S OPTIONS ***
*COMOPT SOURCE=MES.TEST1F.ENGL
*COMOPT XREF,ISD,SAVLST
*COMOPT MODULE=MES.PLAM
*END HALT
```

```
ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                                    11:46:25  94-03-07   PAGE 0003
         SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE                EXTERNAL SYMBOL DICTIONARY
         TEST1F    SD 0001 00000000 00002E  24  24
```

```
ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                                    11:46:25  94-03-07   PAGE 0004
FLAG LOCTN OBJECT CODE    ADDR1  ADDR2  STMNT M  SOURCE STATEMENT
   000000                                   1    TEST1F   START
                         000005             2    R5       EQU   5
                                            3    *
   000000 05 50                             4    BEGIN    BALR  R5,0
   000002                000002             5             USING *,R5
   000002 D2 02 501C501F 00001E 000021      6             MVC   FIELD,NUMBER
   000008 47 F0 501F     000021             7             B     NUMBER
                                            8             TERM
                                            9  1          #INTF INTNAME=TERM,REFTYPE=REQUEST,INTCOMP=001
                                           10  1          IDLKG VER=010,ALIGN=F
                                           11  2                *,VERSION 010                                    00001300
   00000C                                  12  2          CNOP  0,4                                            00002800
   00000C                                  13  2          DS    0F                                             00003500
                                           14  1          ##BAL 1,*+16
   00000C 45 10 501A     00001C           15  2          BAL   1,*+16
   000010 01                              16  1          DC    XL1'01'
   000011 00                              17  1          DC    XL1'00'
   000012 00                              18  1          DC    XL1'00'
   000013 04                              19  1          DC    XL1'04'
   000014 40404040                        20  1          DC    CL4'    '
   000018 00000075                        21  1          DC    XL4'00000075'
   00001C 0A 09                           22  1          SVC   9
                                          23    *
   00001E                                 24    FIELD    DS    CL3
   000021 F1F2F3                          25    NUMBER   DC    C'123'
   000024 07 00                           26    END      NOPR  0
   000000                                 27             END   BEGIN
   000026 9203101514384858                28                   =X'9203101514384858' CONSISTENCY CONSTANT FOR AID
FLAGS IN 00001 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : 1
THIS PROGRAM WAS ASSEMBLED BY ASSEMBHC    V 1.2A00
```

```
ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                                    11:46:25  94-03-07   PAGE 0005
USED FILES AND LIBRARIES
SOURCE FILE   :   :01KH:$HASSEMB.MES.TEST1F.ENGL
MODULE LIBRARY  :      :01KH:$HASSEMB.MES.PLAM
MODULE ELEMENT  :      TEST1F
VERS/DATE       :      @/1994-03-07
SYSTEM MACROLIBRARY :  :D:$TSOS.MACROLIB
MACRO-LIBRARIES     LINKNAME  LIBRARY-NAME
                    SYSLIB    :D:$TSOS.MACROLIB
```

```
ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0006
SYMBOL    LEN  VALUE    DEFN    REFERENCES
BEGIN    00002 00000000 000004  000027
END      00002 00000024 000026
FIELD    00003 0000001E 000024  000006
NUMBER   00003 00000021 000025  000006  000007
R5       00001 00000005 000002  000004  000005
TEST1F   00046 00000000 000001



ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0007
MACRO NAME VERS/DATE  DEFN      REFERENCES
##BAL      010/880614 SYSLIB    000014
#INTF      919/871211 SYSLIB    000009
IDLKG      002/871211 SYSLIB    000010
TERM       010/880615 SYSLIB    000008



ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0008
                                                DIAGNOSTICS
FLAG  MESSAGE AND STATEMENT NUMBERS
D7    ASS0407 ALIGNMENT ERROR IN OPERAND
                000007
ASSEMBLY TIME :          0.212     SEC.
THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

6.3 Laser printer listing

A listing specifically edited for laser printer output (called an "ND listing") can be generated by specifying the option LISTING=PAR(LAYOUT=PAR(LASER-PRINTER=ND2)). It differs from the standard listing in the following respects:

- The ND listing (source program listing) is divided into three sections:
  - object code
  - source program
  - additional information

  The object code and source program sections are identical to the standard listing. The additional information consists of:
  - ISAM key, if the assembled program is contained in an ISAM file.
  - Section names of symbols that represent addresses in instructions.
  - OPSYN listing shows the mnemonic operation code that was changed by means of an OPSYN statement.
  - STACK level indicates the nesting level for each STACK or UNSTK instruction:
    U_x for USING (where $1 \leq {}_x \leq 4$)
    P_x for PRINT (where $1 \leq {}_x \leq 4$)
  - MTRAC information is output completely.
    Restriction: The value of SETC variables will be printed up to a maximum of 50 characters.

- In all cross-reference and diagnostic listings, the statement numbers are increased to a total of 24 per line.

- Hardcopy printout of a laser printer listing.
  ASSEMBH can be instructed to generate an ND listing and save it in a file by specifying the following options:

  ```
  // COMPILE...,LISTING=PAR(LAYOUT=PAR(LASER-PRINTER=ND2),OUTPUT=filename)
  ```

  The maximum line length in the ND listing is 205 characters. Consequently, to obtain a printout on the laser printer, the appropriate paper format (FORM-NAME=) and character set (CHARACTER-SETS=) must be specified in the PRINT-FILE command. Suitable values may be requested from the system administrator.

  The following command can be used to obtain a printout of the listing:

  ```
  /PRINT-FILE FILE-NAME=filename,LAYOUT-CONTROL=PAR(FORM-NAME=format,
   CHARACTER-SETS=chars)
  ```

6.4 SAVLST (listing with ISAM key)

The option LISTING=PAR(OUTPUT=*SAVLST) can be used to create a listing that is in SAVLST format and is compatible with ASSEMB V30.

If an LLM format module is output, the contents of the following address fields change: ADDR in the ESD list, LOCTN, ADDR1 and ADDR2 in the source program list and VALUE in the SYMBOL-XREF and LITERAL-XREF cross reference lists (see section 6.6). The name field (SYMBOL) in the ESD list is extended to 32 characters.

```
0001000    ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0001
0001001              SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE                EXTERNAL SYMBOL DICTIONARY
0001002
0001003              TEST1F    SD 0001 00000000 00002E  24  24
```

```
0000001000 ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                    11:46:25  94-03-07   PAGE 0002
0000002000 FLAG LOCTN OBJECT CODE    ADDR1  ADDR2  STMNT M  SOURCE STATEMENT
0000101001     000000                               1    TEST1F   START
0000201001                          000005           2    R5       EQU   5
0000301001                                           3    *
0000401001     000000 05 50                          4    BEGIN    BALR  R5,0
0000501001     000002               000002           5             USING *,R5
0000601001     000002 D2 02 501C501F 00001E 000021    6             MVC   FIELD,NUMBER
0000701001 D   000008 47 F0 501F     000021           7             B     NUMBER
0000801001                                           8             TERM
0000901001                                           9  1          #INTF INTNAME=TERM,REFTYPE=REQUEST,INTCOMP=001
0001001001                                          10  1          IDLKG VER=010,ALIGN=F
0001101001                                          11  2                *,VERSION 010                                  00001300
0001201001     00000C                               12  2          CNOP  0,4                                          00002800
0001301001     00000C                               13  2          DS    0F                                           00003500
0001401001                                          14  1          ##BAL 1,*+16
0001501001     00000C 45 10 501A     00001C         15  2          BAL   1,*+16
0001601001     000010 01                            16  1          DC    XL1'01'
0001701001     000011 00                            17  1          DC    XL1'00'
0001801001     000012 00                            18  1          DC    XL1'00'
0001901001     000013 04                            19  1          DC    XL1'04'
0002001001     000014 40404040                      20  1          DC    CL4'    '
0002101001     000018 00000075                      21  1          DC    XL4'00000075'
0002201001     00001C 0A 09                         22  1          SVC   9
0002301001                                          23    *
0002401001     00001E                               24    FIELD    DS    CL3
0002501001     000021 F1F2F3                        25    NUMBER   DC    C'123'
0002601001     000024 07 00                         26    END      NOPR  0
0002701001     000000                               27             END   BEGIN
0002801001     000026 9203101514384858              28                   =X'9203101514384858' CONSISTENCY CONSTANT FOR AID
0002802000 FLAGS IN 00001 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
0002803000 HIGHEST ERROR-WEIGHT : 1
0002804000 THIS PROGRAM WAS ASSEMBLED BY ASSEMBHC    V 1.2A00
           SYSTEM MACROLIBRARY :  :D:$TSOS.MACROLIB
           DIAGNOSTIC FILE :      :01KH:$HASSEMB.SAVLST.ASSEMBH.TEST1F
```

```
        0 ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                  11:46:25  94-03-07   PAGE 0003
        0 SYMBOL    LEN  VALUE    DEFN    REFERENCES
        0
BEGIN   1 BEGIN
        2         00002 00000000 000004  000027
END     1 END
        2         00002 00000024 000026
FIELD   1 FIELD
        2         00003 0000001E 000024  000006
NUMBER  1 NUMBER
        2         00003 00000021 000025  000006  000007
R5      1 R5
        2         00001 00000005 000002  000004  000005
TEST1F  1 TEST1F
        2         00046 00000000 000001
        0 ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                  11:46:25  94-03-07   PAGE 0004
        0 MACRO NAME VERS/DATE  DEFN      REFERENCES
        0
##BAL   1 ##BAL
        2           010/880614 SYSLIB    000014
#INTF   1 #INTF
        2           919/871211 SYSLIB    000009
IDLKG   1 IDLKG
        2           002/871211 SYSLIB    000010
TERM    1 TERM
        2           010/880615 SYSLIB    000008



D070000010 ASSEMBH LISTING - FORMAT: F_ASSEMB_COMPATIBLE                                 11:46:25  94-03-07   PAGE 0005
D070000020                                              DIAGNOSTICS
D070000030 FLAG  MESSAGE AND STATEMENT NUMBERS
D070300011 D7    ASS0407 ALIGNMENT ERROR IN OPERAND
D070300022                 000007
0000000001 94-03-07 11:13:38 V 1.2A00     TEST1F
           :01KH:$HASSEMB.MES.TEST1F.ENGL                              51

           THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

6.5 Structured list

The user interface has become more simple now that structured Assembler lists have been integrated in ASSEMBH (see chapter 10, "Utility routines for structured programming").
You select the structuring function using the option LISTING=(,SOURCE-FORMAT=STRUCTURED,) (see section 2.4.4). This is done immediately after assembly.
You must use the predefined macros for structured programming (see the "ASSEMBH Reference Manual" [1] before you can create a structured list. These macros are also referred to as "structure macros" below.

You can also generate a structured list from a permanent CIF (H-type element in a PLAM library) created during a previous assembly run using the standalone list generator (see section 2.5). Select the structuring function using the SOURCE-FORMAT=STRUCTURED option in the GENERATE statement. If you choose to do this, you must ensure that COMPILATION-INFO=PAR(INFO=MAX) was specified for all the options of the structured list when the CIF was generated (see section 2.4.3).

The structured list is generated in standard ASSEMBH format. If you want to structure a list using the LISTING=(LAYOUT=(FORMAT=F-ASSEMB-COMPATIBLE)) option, you must use the relevant utilities (see chapter 10).

6.5.1 Features of the structuring function

1. Structure blocks are indicated by vertical and horizontal bars.
   Instructions and comments are indented by a specified value to indicate the structure level.
2. The indentation value can be specified for each structure level and a fixed column area can be defined.
   Option: `SOURCE-FORMAT=STRUCT(IDENTATION-AMOUNT=...,FIXED-AREA-START=...)`
3. Logging of structure macros can be controlled.
   Option: `SOURCE-FORMAT=STRUCT(,STRUCT-MACRO-PRINT=...)`

Examples

1. Assembly with ASSEMBH and structuring of the log in a single run.

```
START-PROG $ASSEMBH

  Options:
  SOURCE    = Input file containing unstructured source code
  MACRO-LIB = Library containing predefined macros
  LISTING   = (OUTPUT=assemb.list,
               NOPRINT-PREFIX=@,
               SOURCE-FORMAT=STRUCTURED(,INDENT-AMOUNT=n,
               STRUCT-MACRO-PRINT=OBJECT-CODE-ONLY),
               TITLE-STATEM=ACCEPTED,
               LINES-PER-PAGE=n)
```

2. Creation and structuring of the log from the CIF information stored in a library using the standalone list generator.

   The COMPILATION-INFO=PAR(INFO=MAX) option must be specified during assembly.

```
START-PROG $ASSLG

  Options:
  COMPILER-INFO-FILE = Input element containing the permanent CIF
  SOURCE-FORMAT      = STRUCTURED(INDENT-AMOUNT=n,
                       STRUCT-MACRO-PRINT=OBJECT-CODE-ONLY),
                       LINES-PER-PAGE=n
  OUTPUT             = assemb.list
```

Handling of structure errors

Any structure errors are indicated by the structure macros using MNOTES.
Once an error has occurred, the system attempts to continue the structuring process.

6.5.2 The print-edited assembly log

1. Structure blocks

A structure block begins with a start statement. These include the structure macros @BEGIN, @IF, @CASE, @CAS2, @WHILE, @CYCLE and @THRU. The structure block is terminated with the end statement @BEND. A structure block is indicated in the print-edited list by a horizontal line between the call to the structure macro and the right margin of the line of source code. At this point, the current structure level is entered. The start and end of a structure block are linked by a vertical line.

All the instructions and comments which belong to a structure block are logged in accordance with the nested structure and are indented by the correct amount.

2. Procedures

The body of a procedure is indicated by a horizontal line between @ENTR and @END. Instructions located outside the body of the procedure, i.e. before @ENTR and after @END, are not indented. This is not necessary, since it is not possible to nest procedure declarations. In addition, instructions which lie between @ENTR and the first structure block and between the last structure block and @END are not indented. Generally users store data declarations and DSECTS here. The structure of these remains unchanged.

@ENTR, @END and the first structure block following @ENTR start in column 10.

3. Exiting structure blocks

@BREAK, @EXIT and @PASS allow you to exit structure blocks. These structure macros are indicated by an arrow to the left which precedes the statement. Any name entry is entered in a separate line before the structure macro is called.

```
146     |||@THEN *-----------------------------------------------------
146     ||||      OKAY
152  3  ||||
156    <------@PASS NAME=PROC1
162  1  ||||
164  2  ||||
165     |||@ELSE *-----------------------------------------------------
```

4. Handling of instructions and comments

The handling of instructions is described in section 6.5.2.1 and the handling of comments is described in section 6.5.2.2.

5. Logging statements

The EJECT, SPACE and TITLE statements are not logged.

6. Statement numbers

If indentation results in a statement line being split into several separate lines, continuation lines are assigned the same statement number in the log as the first line.

6.5.2.1 Handling of instructions

1. Name entry in structure macro calls

Name entries are not indented. If the name is longer than 8 characters, it is entered in a separate line before the structure macro call as indicated by a horizontal line. The new line has the same number as the original statement.
If the name is 8 characters or shorter, it is retained in the same line as the call.

2. Name entry in assembler calls

Name entries are not indented. The structure is retained, even in the case of long name entries. If the name extends to the rightmost vertical structure line, the remainder of the line, as of the opcode, is stored in a continuation line.

```
        | | | | @BEGI *-----------------------------------------------6-
        | | | |   LR  1,1
SYMBOL_TRANSPORT MVC 0(1,2),0(3)
EXTREMELY_LONG_NAME_ENTRY
        | | | |   MVC SOURCE,TARGET  LONG SYMBOL
        | | | |   AR  2,2
        | | | | @BEND *-----------------------------------------------6-
```

3. Source line

The instructions are indented and, where necessary, split across a number of lines. The opcode, operands and comment field for instructions are indented according to the standard ICTL values (10,16). Continuation lines are generated if the instruction does not fit in the line, even when superfluous blanks are removed.

4. Continuation line

The instruction and all its continuation lines are provided with new line breaks and indented after any superfluous blanks have been compressed.
Superfluous blanks between the opcode, operand and comment field are compressed before any continuation lines are generated as a result of the required indentation.
Continuation lines are indented according to the nesting depth and operands are also indented according to the standard ICTL values (10,16).

```
@BEGI *-----------------------------------------------3-
 @BEGI *-----------------------------------------------4-
  MVC VERY_LONG_TARGET_NAME(L'VERY_LONG_TARGET_NAME),LON*
      G_SOURCE_NAME
 @BEND *-----------------------------------------------4-
 LR  1,1
@BEND *-----------------------------------------------3-
```

5. Macro call with operands

The opcode and operands are indented according to the nesting depth. If a line break is to be added to the line, the operands are also aligned in any continuation lines.

```
    @BEGI *-----------------------------------------6-
     @BEGI *-----------------------------------------7-
      @DATA CLASS=C,BASE=BASEREG,LENGTH=2000,INIT=ADDR*
            INIT
     @BEND *-----------------------------------------7-
    @BEND *-----------------------------------------6-
```

6. Macro call in alternative format

The opcode and operands are indented according to the nesting depth. The operands are also aligned in any continuation lines. If there is insufficient space in the line, the continuation lines are also provided with line breaks.

```
    @BEGI *-----------------------------------------6-
     @BEGI *-----------------------------------------7-
      @DATA CLASS=C,BASE=BASEREG,                     F
            LENGTH=2000,                              F
            INIT=ADDRINIT
     @BEND *-----------------------------------------7-
    @BEND *-----------------------------------------6-
```

6.5.2.2 Handling of comments

1. Comments in structure macro calls

A comment is always separated from the call in order to prevent the horizontal connection line to the level specification in the right margin from being interrupted. The comment is placed in a separate line after the call and indented.

The comment line is always placed before the expansion of the macro and assigned the original statement number. The comment starts immediately below the introductory "*" when a single-line comment still fits in the line. If there is insufficient space or if the comment is a multi-line comment (continuation lines), the comment is split over a number of lines and starts in the opcode field.

```
M SOURCE STATEMENT
        | @THEN *-------------------------------------------------3-
 *      | |     The structure word comment associated with @THEN
3       | |
3       | | generated statements
3       | |
```

2. Comments in Assembler instructions

The comment is retained in the same line. Multiple blanks are compressed starting at the end of the line and, where necessary, in the text. If there is not sufficient space, blanks between the operand and comment fields are compressed. If, despite compression of blanks, there is still insufficient space, the comment is split across a number of continuation lines. Depending on the length of the rest of the line, the entries are aligned with the start of the comment or with the operand field. An additional "*" indicates continuation in the operand field.

```
        | | | | | | | | @BEGI *-------------------------10-
        | | | | | | | |  MVC TARGET1,SOURCE1 COMMENT
**      | | | | | | | |                                    *
        | | | | | | | |  MVC TARGET2,SOURCE2 COMMENT-FIELD*
*       | | | | | | | |                    -ENTRY 2
**      | | | | | | | |                                    *
        | | | | | | | |  MVC TARGET3,SOURCE3 LONG-COMMENT-*
*       | | | | | | | |      *FIELD-ENTRY-THREE
        | | | | | | | | @BEND *-------------------------10-
```

3. Comment line

With comment lines, the introductory * is retained in column 1 and the comment text is indented according to the nesting depth. If there is insufficient space after the compression of blanks, the line is split and indented. The comment lines thus generated are also provided with an asterisk to indicate that they are comments.

```
        | | | | | | | | @BEGI *-----------------------------------------10-
*       | | | | | | | |      THIS LINE IS ALIGNED
        | | | | | | | |  AR  1,1
*       | | | | | | | |  THE COMMENT LINE IS ALIGNED AND CONTINUES IN A CONT*
*       | | | | | | | |  INUATION LINE
        | | | | | | | |  AR  2,2
        | | | | | | | | @BEND *-----------------------------------------10-
```

4. Comment lines in comment boxes

If columns 2 and 71 contain entries, this indicates that a comment cannot be moved. The vertical structure lines are interrupted.

```
        | | | | | | | | @BEGI *-----------------------------------------10-
        | | | | | | | |
*Comment is not indented                                                    *
        | | | | | | | | | AR  1,1
*****************************************************************************
**                                                                          *
**    COMMENT BOX:                                                          *
**                                                                          *
**    THE COMMENT LINES IN A COMMENT BOX ARE                                *
**    INTERRUPTED THROUGH TO THE END OF THE                                 *
**    COMMENT BOX.                                                          *
**                                                                          *
**    THE TEXT IS EASY TO READ                                              *
**                                                                          *
*****************************************************************************
        | | | | | | | | | LR  2,2
        | | | | | | | | @BEND *-----------------------------------------10-
```

5. Defining a right-aligned, fixed-position comment field

It is possible use an option to define a column area which cannot be moved. This area can be used to indicate correction statuses etc.
This does not apply to lines generated by macros (including structure macros).

Option:

```
LISTING = (OUT=assemb.list,SOURCE-FORMAT=STRUCT(,FIXED-AREA-START=m))
```

The new option FIXED-AREA-START can assume values 60 through 255 (default=NONE). It specifies the column in the source as of which no changes should be made during structuring.

Example

The following example shows an unstructured source program using the LISTING option SOURCE-FORMAT=STD (default).

```
*** STRUCTURED LISTING ***                                                                    17:03:59  1994-01-13   PAGE 0003
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                              1             PRINT NOGEN,CODE
                                              2             TITLE '*** STRUCTURED LISTING ***'
 000000                                       3     STRUCTLG    @ENTR TYP=E
 000000 90 EC D00C     0000000C             112   1
 000004 18 AF                               113   1
 000006 58 F0 A110     00000110             118   2
 00000A 05 EF                               126   3
 00000C 00000060                            127   2
 000010 E2E3D9E4D2E3D3C7                    128   2
 000018                                     131     LONG_SYMBOL @BEGIN
 000018                                     139     RTCCHECK @IF   ZE            CHECK RTC
 000018 12 11                               146             LTR   1,1
 00001A                                     147             @THEN               OKAY
 00001A 47 70 A028     00000028             153   3
 00001E                                     156             @PASS NAME=PROC1
 00001E 58 F0 A114     00000114             162   1
 000022 05 EF                               169   2
 000024                                     170             @ELSE               NOT OKAY
 000024 47 F0 A0B8     000000B8             174   1
 000028                                     178     ERROR    @CASE (1)          ERROR HANDLING
 000028 89 10 0001                          185   1
 00002C 48 11 A0B0     000000B0             186   1
 000030 47 F1 A030     00000030             187   1
 000034                                     188     CASE1    @BEGI              CASE 1
 000034 1A 11                               196             AR          1,1
 000036                                     197             @BEND              END OF CASE 1
 000036 47 F0 A0B8     000000B8             203   1
 00003A                                     204     CASE2    @BEGI              CASE 2
                                            212     *        MACRO CALL AND OPERANDS
                                            213             @BEGI
 00003A                                     219     DATAC1   @DATA            CLASS=C,BASE=5,LENGTH=1000
 00003A 58 F0 A118     00000118             225   1
 00003E 58 50 A11C     0000011C             226   1
 000042 05 EF                               234   2
 000044 001C                                235   1
 000046 C35C                                236   1
 000048 00000000                            237   1
                                            238     *        MACRO CALL IN ALTERNATIVE FORMAT
 00004C                                     239     DATAC2   @DATA      CLASS=C,BASE=BASEREG,                                F
                                            239                   LENGTH=2000,                                               F
                                            239                   INIT=ADRINIT
 00004C 58 F0 A118     00000118             245   1
 000050 58 60 A120     00000120             246   1
 000054 0700                                247   1
 000056 05 EF                               254   2
 000058 0020                                255   1
 00005A C3C1                                256   1
 00005C 00000108                            257   1
 000060 1A 11                               258             AR             1,1
                                            259     * COMMENT ALIGNED
                                            260     * THIS COMMENT DOES NOT FIT IN A SINGLE LINE IN THE CURRENT STRUCTURE
                                            261             PRINT GEN
                                            262             COLMAC@             ---- MACRO WITH STRUCTURE ELEMENTS
                                            263   1 * MACRO WITH STRUCTURE ELEMENTS
                                            264   1         @IF   EQ
```

```
*** STRUCTURED LISTING ***                                                    17:03:59  1994-01-13   PAGE 0004
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 000062 15 11                               271   1          CLR   1,1
                                            272   1          @THEN
 000064 47 70 A07C     0000007C             278   4
                                            281   1          @IF EQ
 000068 15 11                               288   1          CLR 1,1
                                            289   1          @THEN
 00006A 47 70 A07C     0000007C             295   4
                                            298   1          @IF   EQ
 00006E 15 11                               305   1          CLR   1,1
                                            306   1          @THEN
 000070 47 70 A07A     0000007A             312   4
 000074 1A 11                               315   1          AR    1,1
                                            316   1          @ELSE
 000076 47 F0 A07C     0000007C             320   2
 00007A 1A 12                               324   1          AR    1,2
                                            325   1          @BEND
                                            332   1          @BEND
                                            339   1          @BEND
                                            346              PRINT NOGEN,CODE
                                            347              @BEND
 00007C                                     353     DATAF1   @FREE BASE=5
 00007C 58 F0 A124     00000124             362   2
 000080 05 EF                               369   3
 000082 001C                                370   2
 000084 5C                                  371   2
 00008A                                     372     DATAF2   @FREE BASE=BASEREG
 00008A 58 F0 A124     00000124             381   2
 00008E 05 EF                               388   3
 000090 0020                                389   2
 000092 5C                                  390   2
 000098                                     391     CASE2_E  @BEND                END OF CASE 2
 000098 47 F0 A0B8     000000B8             398   1
 00009C                                     399              @BEGI                CASE 3
                                            406              PRINT GEN
                                            407              COLMAC               ── MACRO WITHOUT STRUCTURE ELEMENTS
                                            408   1 * MACRO WITHOUT STRUCTURE ELEMENTS
 00009C 1A 11                               409   1          AR  1,1
 00009E 18 11                               410   1          LR  1,1
 0000A0 1B 11                               411   1          SR  1,1
 0000A2 15 11                               412   1          CLR 1,1
 0000A4 41 10 0002                          413   1          LA    1,2
                                            414              PRINT NOGEN,CODE
 0000A8                                     415              @BEND                END OF CASE 3
 0000A8 47 F0 A0B8     000000B8             421   1
 0000AC                                     422              @BEND                END OF ERROR HANDLING
 0000AC 47 F0 A0B8     000000B8             428   1
 0000B0 0003                                429   1
 0000B2 0004                                430   1
 0000B4 000A                                431   1
 0000B6 006C                                432   1
 0000B8                                     434     RTC_END  @BEND                END OF RTC
                                            442              @BEND
                                            448     *
                                            449              @BEGI
 0000B8                                     455              @PASS NAME=PROC2
```

```
*** STRUCTURED LISTING ***                                                                    17:03:59  1994-01-13   PAGE 0005
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 0000B8 58 F0 A128     00000128             461   1
 0000BC 05 EF                               468   2
                                            469              @BEGI
                                            475              @BEGI
 0000BE 41 20 0001                          481     EXTREMELY_LONG_NAME_ENTRY          LA    2,1
                                            482              @BEGI
                                            488              @BEGI
                                            494              @BEGI
                                            500              @BEGI
                                            506              @BEGI
                                            512              @BEGI
                                            518     *                                                                     *
 0000C2 18 11                               519     NAME1    LR 1,1
 0000C4 1A 22                               520     NAME2    AR        2,2        COMMENT
 0000C6 D2 03 A0FCA0F8 000000FC 000000F8    521              MVC       LONG_TARGET_FIELD(L'LONG_TARGET_FIELD),LONG_SOURCE
                                            522     *                                                                     *
 0000CC D2 03 A0F4A0F0 000000F4 000000F0    523     EXTREMELY_LONGNAME_ENTRY  MVC VERY_LONG_TARGET,VERY_LONG_SOURCE
                                            524     *                                                                     *
 0000D2 41 20 0002                          525              LA    2,2        ASSEMBLER STATEMENT    WITH    LONG    COMMENT
                                            526     *                                                                     *
                                            527     *  BLANK COMPRESSION IN COMMENT FIELD                                 *
 0000D6 D2 03 A104A100 00000104 00000100    528              MVC  TARGET,SOURCE  COMM.FIELD         WITH          SPACES
                                            529     *
                                            530     ** COMMENT IS NOT INDENTED                                          *
 0000DC 1A 11                               531              AR    1,1
                                            532     ************************************************************************
                                            533     **                                                                    *
                                            534     **    COMMENT BOX:                                                    *
                                            535     **                                                                    *
                                            536     **    THE STRUCTURE LINES IN A COMMENT BOX ARE                        *
                                            537     **    INTERRUPTED THROUGH TO THE END OF THE                           *
                                            538     **    COMMENT BOX                                                     *
                                            539     **                                                                    *
                                            540     **    THE TEXT IS EASY TO READ                                        *
                                            541     **                                                                    *
                                            542     ************************************************************************
 0000DE 18 22                               543              LR    2,2
                                            544     *
                                            545              @BEND
 0000E0                                     551              @IF   ZE
 0000E0 12 11                               558              LTR   1,1
 0000E2                                     559              @THEN
 0000E2 47 70 A0E8     000000E8             565   3
 0000E6 1A 11                               568              AR    1,1
 0000E8                                     569              @BEND
                                            576              @BEND
                                            582              @BEND
                                            588              @BEND
                                            594              @BEND
                                            600              @BEND
                                            606              @BEND
                                            612              @BEND
                                            618              @BEND
 0000E8                                     624              @EXIT
 0000E8 58 F0 A12C     0000012C             633   2
```

```
*** STRUCTURED LISTING ***                                                                    17:03:59  1994-01-13   PAGE 0006
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 0000EC 05 EF                                640   3
 0000EE F1F0                                 641   2
 0000F0                                      642     VERY_LONG_SOURCE     DS      F      LONG SYMBOL NAME FOR TARGET ADDRESS
 0000F4                                      643     VERY_LONG_TARGET     DS      F      LONG NAME FOR SOURCE ADDRESS
 0000F8                                      644     LONG_SOURCE          DS      F
 0000FC                                      645     LONG_TARGET FIELD    DS      F
 000100                                      646     SOURCE               DS      F
 000104                                      647     TARGET               DS      F
 000108                                      648     ADRINIT  DS    F
                       00000006              649     BASEREG  EQU   6
 000110                                      650     ENTR_END @END
 000110 00000000                             654   1
 000114 00000130                             655   1
 000118 00000000                             656   1
 00011C 000003E8                             657   1
 000120 000007D0                             658   1
 000124 00000000                             659   1
 000128 00000168                             660   1
 00012C 00000000                             661   1
 000130                                      665     PROC1    @ENTR TYP=I
 000130 90 EC D00C     0000000C              672   1
 000134 18 AF                                673   1
 000136 58 F0 A028     00000158              678   2
 00013A 05 EF                                686   3
 00013C 00000060                             687   2
 000140 D7D9D6C3F1404040                     688   2
 000148                                      691             @PASS NAME=PROC2
 000148 58 F0 A02C     0000015C              697   1
 00014C 05 EF                                704   2
 00014E                                      705             @EXIT
 00014E 58 F0 A030     00000160              714   2
 000152 05 EF                                721   3
 000154 F1F0                                 722   2
 000158                                      723             @END
 000158 00000000                             727   1
 00015C 00000168                             728   1
 000160 00000000                             729   1
 000168                                      733     PROC2    @ENTR TYP=L
 000168                                      741             @EXIT
 000168 07 FE                                747   1
 000170                                      748             @END
                                             755             END
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-01-13 AT 17:02:51
```

```
*** STRUCTURED LISTING ***                                                                    17:03:59  1994-01-13   PAGE 0007
USED FILES AND LIBRARIES
SOURCE LIBRARY  :      :U:$ASS1.ESC.TSTLIB
SOURCE ELEMENT  :      STR.SOURCE
VERS/DATE       :      @/1993-08-24
MACRO-LIBRARIES     LINKNAME  LIBRARY-NAME
                              :U:$ASS1.ESC.TSTLIB
                              :U:$ASS1.VO.LIB
                              :U:$ASS1.ASS1.LIB
                              :H:$TSOS.SYSLIB.ASSEMBH.011
ASSEMBLY TIME :         16.081     SEC.
THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

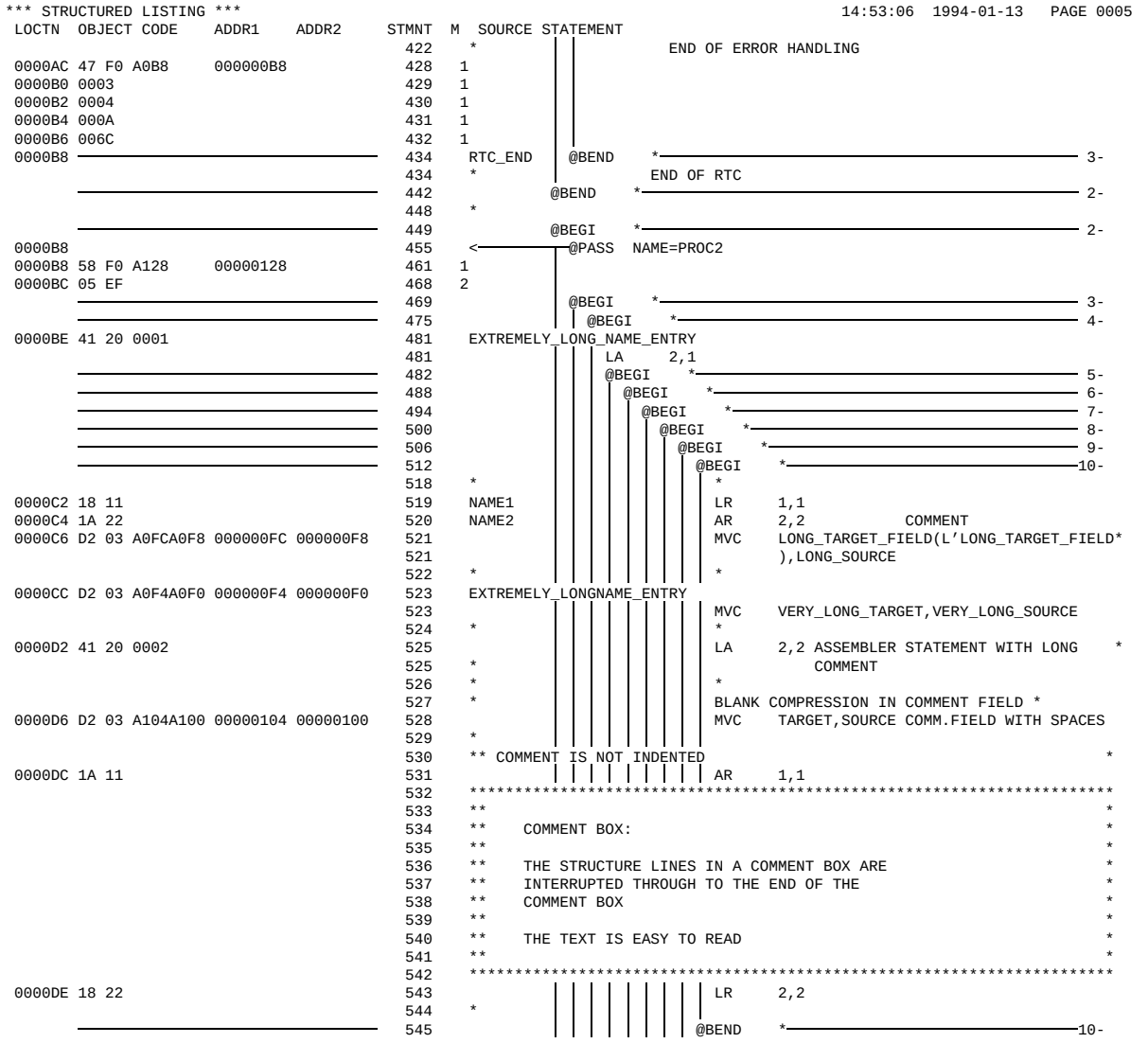
Example

The following example shows a source program structured with the LISTING option SOURCE-FORMAT=STRUCTURED (see section 2.4.4).

```
*** STRUCTURED LISTING ***                                                                    14:53:06  1994-01-13   PAGE 0001

SOURCE=*LIBRARY-ELEMENT
  (LIBRARY=:U:$ASS1.ESC.TSTLIB,ELEMENT=STR.SOURCE
    (VERSION=*UPPER-LIMIT)),

MACRO-LIBRARY=
  (ESC.TSTLIB,VO.LIB,ASS1.LIB,$TSOS.SYSLIB.ASSEMBH.011,$TSOS.SYSLIB.BS2CP.100),

COPY-LIBRARY=*NONE,

SOURCE-PROPERTIES=PARAMETERS
  (FROM-COLUMN=1,TO-COLUMN=71,CONTINUATION-COLUMN=16,LOW-CASE-CONVERSION=NO,INSTRUCTION-SET=BS2000-XS,
  PREDEFINED-VARIABLES=NONE),

COMPILER-ACTION=MODULE-GENERATION
  (MODE=STD,MODULE-FORMAT=OM),

MODULE-LIBRARY=*OMF,

COMPILATION-INFO=PARAMETERS
  (INFORMATION=STD,OUTPUT=*LIBRARY-ELEMENT
    (LIBRARY=ESC.TSTLIB,ELEMENT=STR.PROT
      (VERSION=6789-9876543210))),

LISTING=PARAMETERS
  (SOURCE-PRINT=WITH-OBJECT-CODE
    (PRINT-STATEMENTS=ACCEPTED,LINE-NUMBERING=NO),
  SOURCE-FORMAT=STRUCTURED
    (EVALUATED-NEST-LEVEL=ALL,INDENTATION-AMOUNT=2,FIXED-AREA-START=NONE,STRUCT-MACRO-PRINT=OBJECT-CODE-ONLY),
  MACRO-PRINT=PARAMETERS
    (NOPRINT-NEST-LEVEL=255,NOPRINT-PREFIX=@,TITLE-STATEMENTS=IGNORED,MACRO-ORIGIN-INFO=SEPARATE),
  MIN-MESSAGE-WEIGHT=SIGNIFICANT,CROSS-REFERENCE=PARAMETERS
    (SYMBOL=NO,LITERAL=NO,MACRO=NO,COPY=NO,DIAGNOSTICS=YES),
  EXTERNAL-DICTIONARY=YES,LAYOUT=PARAMETERS
    (LINES-PER-PAGE=60,LASER-PRINTER=NO,FORMAT=STD),
  OUTPUT=ESLL.STR.SOURCE),

TEST-SUPPORT=NO,

COMPILER-TERMINATION=PARAMETERS
  (MAX-ERROR-WEIGHT=FATAL,MAX-ERROR-NUMBER=32767,MAX-MACRO-NEST-LEVEL=255,MAX-COPY-NEST-LEVEL=5),

CORRECTION-CYCLE=NO,

MAINTENANCE-OPTIONS=PARAMETERS
  (CHANNEL-INSTRUCTIONS=NO),

COMPILATION-SPACE=STD



*** STRUCTURED LISTING ***                                                                    14:53:06  1994-01-13   PAGE 0002
          SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE                EXTERNAL SYMBOL DICTIONARY
  (DUMMY) @SAV      DS 0001 00000000 000058
          STRUKTLG  SD 0002 00000000 000170  24  24
          IASSENTR  VC 0003
          IASSCNTR  VC 0004
          IASSFREE  VC 0005
          IASSEXIT  VC 0006
```

```
*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0003
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             1            PRINT NOGEN,CODE
                                             3   *-------------------------------------------------------------------------------
000000                                       3    STRUCTLG @ENTR  TYP=E
000000 90 EC D00C     0000000C               112  1
000004 18 AF                                 113  1
000006 58 F0 A110     00000110               118  2
00000A 05 EF                                 126  3
00000C 00000060                              127  2
000010 E2E3D9E4D2E3D3C7                       128  2
000018                                       131   LONG_SYMBOL DS 0H
                                             131           @BEGIN    *------------------------------------------------------ 2-
000018 ----------------------------------    139   RTCCHECK  @IF    ZE    *------------------------------------------------- 3-
                                             139   *                      CHECK RTC
000018 12 11                                 146                  LTR    1,1
00001A ----------------------------------    147                 @THEN    *------------------------------------------------ 3-
                                             147   *                      OKAY
00001A 47 70 A028     00000028               153  3
00001E                                       156   <--------------@PASS  NAME=PROC1
00001E 58 F0 A114     00000114               162  1
000022 05 EF                                 169  2
000024 ----------------------------------    170                 @ELSE    *------------------------------------------------ 3-
                                             170   *                      NOT OKAY
000024 47 F0 A0B8     000000B8               174  1
000028 ----------------------------------    178   ERROR       @CASE  (1)    *------------------------------------------- 4-
                                             178   *                        ERROR HANDLING
000028 89 10 0001                            185  1
00002C 48 11 A0B0     000000B0               186  1
000030 47 F1 A030     00000030               187  1
000034 ----------------------------------    188   CASE1         @BEGI    *------------------------------------------- 5-
                                             188   *                      CASE 1
000034 1A 11                                 196                  AR     1,1
000036 ----------------------------------    197                 @BEND    *------------------------------------------- 5-
                                             197   *                      END OF CASE 1
000036 47 F0 A0B8     000000B8               203  1
00003A ----------------------------------    204   CASE2         @BEGI    *------------------------------------------- 5-
                                             204   *                      CASE 2
                                             212   *             MACRO CALL AND OPERANDS
                                             213                 @BEGI    *------------------------------------------ 6-
00003A                                       219   DATAC1          @DATA  CLASS=C,BASE=5,LENGTH=1000
00003A 58 F0 A118     00000118               225  1
00003E 58 50 A11C     0000011C               226  1
000042 05 EF                                 234  2
000044 001C                                  235  1
000046 C35C                                  236  1
000048 00000000                              237  1
                                             238   *
                                             239   *             MACRO CALL IN ALTERNATIVE FORMAT
00004C                                       239   DATAC2          @DATA  CLASS=C,BASE=BASEREG,                          F
                                             239                         LENGTH=2000,                                  F
                                             239                         INIT=ADRINIT
00004C 58 F0 A118     00000118               245  1
000050 58 60 A120     00000120               246  1
000054 0700                                  247  1
000056 05 EF                                 254  2
000058 0020                                  255  1
00005A C3C1                                  256  1
00005C 00000108                              257  1
000060 1A 11                                 258                          AR     1,1
```

```
*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0004
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                            259    *                  |  |  |  |  COMMENT ALIGNED
                                            260    *                  |  |  |  |  THIS COMMENT DOES NOT FIT IN A SINGLE LINE IN THE   *
                                            260    *                  |  |  |  |  CURRENT STRUCTURE
                                            261                       |  |  |  |  PRINT  GEN
                                            262                       |  |  |  |  COLMAC@ ---- MACRO WITH STRUCTURE ELEMENTS
                                            263   1*                  |  |  |  |  MACRO WITH STRUCTURE ELEMENTS
        ------------------------------      264   1                   |  |  |  |  @IF    EQ    *------------------------------ 7-
000062 15 11                                271   1                   |  |  |  |   CLR    1,1
        ------------------------------      272   1                   |  |  |  |  @THEN    *------------------------------ 7-
000064 47 70 A07C     0000007C              278   4                   |  |  |  |
        ------------------------------      281   1                   |  |  |  |   @IF    EQ    *------------------------------ 8-
000068 15 11                                288   1                   |  |  |  |    CLR    1,1
        ------------------------------      289   1                   |  |  |  |   @THEN    *------------------------------ 8-
00006A 47 70 A07C     0000007C              295   4                   |  |  |  |
        ------------------------------      298   1                   |  |  |  |    @IF    EQ    *------------------------------ 9-
00006E 15 11                                305   1                   |  |  |  |     CLR    1,1
        ------------------------------      306   1                   |  |  |  |    @THEN    *------------------------------ 9-
000070 47 70 A07A     0000007A              312   4                   |  |  |  |
000074 1A 11                                315   1                   |  |  |  |     AR     1,1
        ------------------------------      316   1                   |  |  |  |    @ELSE    *------------------------------ 9-
000076 47 F0 A07C     0000007C              320   2                   |  |  |  |
00007A 1A 12                                324   1                   |  |  |  |     AR     1,2
        ------------------------------      325   1                   |  |  |  |    @BEND    *------------------------------ 9-
        ------------------------------      332   1                   |  |  |  |   @BEND    *------------------------------ 8-
        ------------------------------      339   1                   |  |  |  |  @BEND    *------------------------------ 7-
                                            346                       |  |  |  |  PRINT  NOGEN,CODE
        ------------------------------      347                       |  |  |  @BEND    *------------------------------ 6-
00007C                                      353    DATAF1             |  |  |  @FREE  BASE=5
00007C 58 F0 A124     00000124              362   2                   |  |  |
000080 05 EF                                369   3                   |  |  |
000082 001C                                 370   2                   |  |  |
000084 5C                                   371   2                   |  |  |
00008A                                      372    DATAF2             |  |  |  @FREE  BASE=BASEREG
00008A 58 F0 A124     00000124              381   2                   |  |  |
00008E 05 EF                                388   3                   |  |  |
000090 0020                                 389   2                   |  |  |
000092 5C                                   390   2                   |  |  |
000098 ------------------------------       391    CASE2_E            |  |  @BEND    *------------------------------ 5-
                                            391    *                  |  |         END OF CASE 2
000098 47 F0 A0B8     000000B8              398   1                   |  |
00009C ------------------------------       399                       |  |  @BEGI    *------------------------------ 5-
                                            399    *                  |  |  |      CASE 3
                                            406                       |  |  |  PRINT  GEN
                                            407                       |  |  |  COLMAC ---- MACRO WITHOUT STRUCTURE ELEMENTS
                                            408   1*                  |  |  |  MACRO WITHOUT STRUCTURE ELEMENTS
00009C 1A 11                                409   1                   |  |  |  AR     1,1
00009E 18 11                                410   1                   |  |  |  LR     1,1
0000A0 1B 11                                411   1                   |  |  |  SR     1,1
0000A2 15 11                                412   1                   |  |  |  CLR    1,1
0000A4 41 10 0002                           413   1                   |  |  |  LA     1,2
                                            414                       |  |  |  PRINT  NOGEN,CODE
0000A8 ------------------------------       415                       |  |  @BEND    *------------------------------ 5-
                                            415    *                  |  |         END OF CASE 3
0000A8 47 F0 A0B8     000000B8              421   1                   |  |
0000AC ------------------------------       422                       |  @BEND    *------------------------------ 4-
```



```
*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0005
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                            422    *                    END OF ERROR HANDLING
 0000AC 47 F0 A0B8     000000B8             428   1
 0000B0 0003                                429   1
 0000B2 0004                                430   1
 0000B4 000A                                431   1
 0000B6 006C                                432   1
 0000B8 ------------------------------      434    RTC_END    @BEND    *------------------------------------------ 3-
                                            434    *                    END OF RTC
        ------------------------------      442              @BEND    *------------------------------------------ 2-
                                            448    *
        ------------------------------      449              @BEGI    *------------------------------------------ 2-
 0000B8                                     455    <----------@PASS  NAME=PROC2
 0000B8 58 F0 A128     00000128             461   1
 0000BC 05 EF                               468   2
        ------------------------------      469              @BEGI    *------------------------------------------ 3-
        ------------------------------      475                 @BEGI    *--------------------------------------- 4-
 0000BE 41 20 0001                          481    EXTREMELY_LONG_NAME_ENTRY
                                            481                    LA     2,1
        ------------------------------      482                    @BEGI    *------------------------------------ 5-
        ------------------------------      488                       @BEGI    *--------------------------------- 6-
        ------------------------------      494                          @BEGI    *------------------------------ 7-
        ------------------------------      500                             @BEGI    *--------------------------- 8-
        ------------------------------      506                                @BEGI    *------------------------ 9-
        ------------------------------      512                                   @BEGI    *-------------------10-
                                            518    *                               *
 0000C2 18 11                               519    NAME1                           LR     1,1
 0000C4 1A 22                               520    NAME2                           AR     2,2           COMMENT
 0000C6 D2 03 A0FCA0F8 000000FC 000000F8    521                                    MVC    LONG_TARGET_FIELD(L'LONG_TARGET_FIELD*
                                            521                                    ),LONG_SOURCE
                                            522    *                               *
 0000CC D2 03 A0F4A0F0 000000F4 000000F0    523    EXTREMELY_LONGNAME_ENTRY
                                            523                                    MVC    VERY_LONG_TARGET,VERY_LONG_SOURCE
                                            524    *                               *
 0000D2 41 20 0002                          525                                    LA     2,2 ASSEMBLER STATEMENT WITH LONG    *
                                            525    *                                   COMMENT
                                            526    *                               *
                                            527    *                               BLANK COMPRESSION IN COMMENT FIELD *
 0000D6 D2 03 A104A100 00000104 00000100    528                                    MVC    TARGET,SOURCE COMM.FIELD WITH SPACES
                                            529    *
                                            530    ** COMMENT IS NOT INDENTED                                          *
 0000DC 1A 11                               531                                    AR     1,1
                                            532    ************************************************************************
                                            533    **                                                                   *
                                            534    **    COMMENT BOX:                                                   *
                                            535    **                                                                   *
                                            536    **    THE STRUCTURE LINES IN A COMMENT BOX ARE                       *
                                            537    **    INTERRUPTED THROUGH TO THE END OF THE                          *
                                            538    **    COMMENT BOX                                                    *
                                            539    **                                                                   *
                                            540    **    THE TEXT IS EASY TO READ                                       *
                                            541    **                                                                   *
                                            542    ************************************************************************
 0000DE 18 22                               543                                    LR     2,2
                                            544    *
        ------------------------------      545                                    @BEND    *-------------------10-
```

```
*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0006
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 0000E0 ──────────────────────────────────  551                @IF    ZE    *──────────────────────────────10-
 0000E0 12 11                               558                 LTR    1,1
 0000E2 ──────────────────────────────────  559                @THEN    *──────────────────────────────10-
 0000E2 47 70 A0E8     000000E8             565   3
 0000E6 1A 11                               568                 AR     1,1
 0000E8 ──────────────────────────────────  569                @BEND    *──────────────────────────────10-
        ──────────────────────────────────  576               @BEND    *──────────────────────────────9-
        ──────────────────────────────────  582              @BEND    *──────────────────────────────8-
        ──────────────────────────────────  588             @BEND    *──────────────────────────────7-
        ──────────────────────────────────  594            @BEND    *──────────────────────────────6-
        ──────────────────────────────────  600           @BEND    *──────────────────────────────5-
        ──────────────────────────────────  606          @BEND    *──────────────────────────────4-
        ──────────────────────────────────  612         @BEND    *──────────────────────────────3-
        ──────────────────────────────────  618        @BEND    *──────────────────────────────2-
 0000E8                                     624    <─────────@EXIT
 0000E8 58 F0 A12C     0000012C             633   2
 0000EC 05 EF                               640   3
 0000EE F1F0                                641   2
 0000F0                                     642    VERY_LONG_SOURCE     DS      F      LONG SYMBOL NAME FOR TARGET ADDRESS
 0000F4                                     643    VERY_LONG_TARGET     DS      F      LONG NAME FOR SOURCE ADDRESS
 0000F8                                     644    LONG_SOURCE          DS      F
 0000FC                                     645    LONG_TARGET_FIELD    DS      F
 000100                                     646    SOURCE               DS      F
 000104                                     647    TARGET               DS      F
 000108                                     648    ADRINIT  DS    F
                       00000006             649    BASEREG  EQU   6
 000110                                     650    ENTR_END @END
 ──────────────────────────────────────────  651    *──────────────────────────────────────────────────────────
 000110 00000000                            654   1
 000114 00000130                            655   1
 000118 00000000                            656   1
 00011C 000003E8                            657   1
 000120 000007D0                            658   1
 000124 00000000                            659   1
 000128 00000168                            660   1
 00012C 00000000                            661   1
 ──────────────────────────────────────────  665    *──────────────────────────────────────────────────────────
 000130                                     665    PROC1    @ENTR  TYP=I
 000130 90 EC D00C     0000000C             672   1
 000134 18 AF                               673   1
 000136 58 F0 A028     00000158             678   2
 00013A 05 EF                               686   3
 00013C 00000060                            687   2
 000140 D7D9D6C3F1404040                    688   2
 000148                                     691    <─────────@PASS  NAME=PROC2
 000148 58 F0 A02C     0000015C             697   1
 00014C 05 EF                               704   2
 00014E                                     705    <─────────@EXIT
 00014E 58 F0 A030     00000160             714   2
 000152 05 EF                               721   3
 000154 F1F0                                722   2
 000158                                     723            @END
 ──────────────────────────────────────────  724    *──────────────────────────────────────────────────────────
 000158 00000000                            727   1
 00015C 00000168                            728   1
 000160 00000000                            729   1
 ──────────────────────────────────────────  733    *──────────────────────────────────────────────────────────
 000168                                     733    PROC2    @ENTR  TYP=L
 000168                                     741    <─────────@EXIT
```

```
*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0007
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 000168 07 FE                               747   1
 000170                                     748             @END
 ------------------------------------------ 749    *-------------------------------------------------------------
                                            755             END
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-01-13 AT 14:51:58



*** STRUCTURED LISTING ***                                                    14:53:06  1994-01-13   PAGE 0008
USED FILES AND LIBRARIES
SOURCE LIBRARY  :      :U:$ASS1.ESC.TSTLIB
SOURCE ELEMENT  :      STR.SOURCE
VERS/DATE       :      @/1993-08-24
MACRO-LIBRARIES     LINKNAME  LIBRARY-NAME
                              :U:$ASS1.ESC.TSTLIB
                              :U:$ASS1.VO.LIB
                              :U:$ASS1.ASS1.LIB
                              :H:$TSOS.SYSLIB.ASSEMBH.011
ASSEMBLY TIME :         16.175     SEC.
THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

6.6 Differences in lists where the module is output in LLM format

In the object module used up to now, the individual CSECTs were addressed contiguously and in ascending order (module-relative addressing).
Where modules are output in LLM format, CSECT-relative addressing is used, i.e. every CSECT in the module begins at location 0.
In this respect, a CSECT behaves as a DSECT. The same applies for the corresponding information in SAVLST (see section 6.4).

CSECT-relative addressing means that the contents of the address fields in the ESD and source program list and in the SYMBOL-XREF and LITERAL-XREF cross-reference lists are changed. All address values are offsets from the beginning of the corresponding CSECT, which always starts at location 0.

List	Field
ESD	ADDR
SOURCE	LOCTN ADDR1 ADDR2
SYMBOL-XREF, LITERAL-XREF	VALUE

The name field (SYMBOL) in the ESD list is extended to 32 characters.

The following examples show the lists in OM and LLM formats.

6.6.1 Lists in OM format

ESD list

```
         SYMBOL   TYPE ID  ADDR     LENGTH  A/R-MODE     EXTERNAL SYMBOL DICTIONARY
         C1        SD 0001 00000000 000020  24  24
         LONGER_N  ER 0002
         C2        SD 0003 00000020 000008  24  24
  (DUMMY) D1        DS 0004 00000000 000004
```

SOURCE LISTING

```
LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
000000                                       1     C1       CSECT
                                             2              EXTRN LONGER_NAME
000000 05 A0                                 3              BALR  10,0
000002                 00000002              4              USING *,10
000002                 00000020              5              USING C2,11
000002                 00000000              6              USING D1,12
000002 58 B0 A016      00000018              7              L     11,=A(C2)
000006 41 20 A012      00000014              8              LA    2,C1_AD1
00000A 41 20 B004      00000024              9              LA    2,C2_AD2
00000E 41 20 C000      00000000             10              LA    2,D1_AD1
000014                                      11   C1_AD1     DS    F
                                            12              LTORG
000018 00000020                             13                    =A(C2)
                                            14              SPACE ,
000020                                      15   C2         CSECT READ
000020                                      16   C2_AD1     DS    CL4
000024                                      17   C2_AD2     DS    F
                                            18              SPACE ,
000000                                      19   D1         DSECT
000000                                      20   D1_AD1     DS    F
000000                                      21              END   C1
```

Cross-reference list (SYMBOL-XREF)

```
SYMBOL    LEN  VALUE    DEFN    REFERENCES
C1       00032 00000000 000001  000021
C1_AD1   00004 00000014 000011  000008A
C2       00008 00000020 000015  000005  000007A 000015
C2_AD1   00004 00000020 000016
C2_AD2   00004 00000024 000017  000009A
D1       00004 00000000 000019  000006  000019
D1_AD1   00004 00000000 000020  0000010A
LONGER_NAME
         00000 00000000 000002
```

6.6.2 Lists in LLM format

Field contents not compatible with OM format are printed in bold.

ESD list

```
         SYMBOL                          TYPE ID  ADDR     LENGTH  A/R-MODE     EXTERNAL SYMBOL DICTIONARY
         C1                               SD 0001 00000000 000020  24  24
         LONGER_NAME                      ER 0002
         C2                               SD 0003 00000000 000008  24  24
  (DUMMY) D1                              DS 0004 00000000 000004
```

SOURCE LISTING

```
LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
000000                                       1     C1       CSECT
                                             2              EXTRN LONGER_NAME
000000 05 A0                                 3              BALR  10,0
000002                 00000002              4              USING *,10
000002                 00000000              5              USING C2,11
000002                 00000000              6              USING D1,12
000002 58 B0 A016      00000018              7              L     11,=A(C2)
000006 41 20 A012      00000014              8              LA    2,C1_AD1
00000A 41 20 B004      00000004              9              LA    2,C2_AD2
00000E 41 20 C000      00000000             10              LA    2,D1_AD1
000014                                      11   C1_AD1     DS    F
                                            12              LTORG
000018 00000000                             13                    =A(C2)
                                            14              SPACE ,
000000                                      15   C2         CSECT READ
000000                                      16   C2_AD1     DS    CL4
000004                                      17   C2_AD2     DS    F
                                            18              SPACE ,
000000                                      19   D1         DSECT
000000                                      20   D1_AD1     DS    F
000000                                      21              END   C1
```

Cross-reference list (SYMBOL-XREF)

```
SYMBOL    LEN  VALUE    DEFN    REFERENCES
C1       00032 00000000 000001  000021
C1_AD1   00004 00000014 000011  000008A
C2       00008 00000000 000015  000005  000007A 000015
C2_AD1   00004 00000000 000016
C2_AD2   00004 00000004 000017  000009A
D1       00004 00000000 000019  000006  000019
D1_AD1   00004 00000000 000020  0000010A
LONGER_NAME
         00000 00000000 000002
```

7 Language interfaces

7.1 Symbolic linking of assembler programs

The text of an assembler source program consists of one or more assembly units. An assembly unit usually begins with a START or CSECT instruction and is terminated with an END instruction. The assembly unit is often loosely designated as a "program".
Each assembly unit is assembled into a module.
An assembly unit may be made up of one or more control sections, which are assembled as parts of a module.
One or more modules can be linked into an executable program (see section 5.4).

Using appropriate instructions (see below), it is possible to

– branch from one program segment to another

– refer to data that is defined in another program segment.

Intercommunication between program segments must be established for this purpose.

– Every individual control section that is addressed must be symbolically addressable.

– The two or more assembly units which are to be linked must be linked symbolically.

Symbolic program linking enables symbols defined in one assembly unit to be accessed from another unit. To do this, the assembler requires appropriate information, which it passes on to the linkage editor via ESD entries. The linkage editor replaces these symbolic references with actual addresses prior to or during loading.
A symbol which is to be accessed from another assembly unit must be identified to the assembler and the linkage editor via the ENTRY instruction (see "ASSEMBH (BS2000) Reference Manual" [1]). This defines it as a symbol of an entry point.

In an assembly unit where symbols defined in another unit are used, these must be identified via the EXTRN or WXTRN instruction (see "ASSEMBH (BS2000) Reference Manual" [1]). In order to access the symbol, a base register must be provided in the assembly unit which uses the EXTRN address. The value of the address must be loaded into the base register via an A-type constant (see "ASSEMBH (BS2000) Reference Manual" [1], DC instruction).

Another method of symbolic linking is the use of V-type constants (see "ASSEMBH (BS2000) Reference Manual" [1], DC instruction). These constants are regarded as indirect linkage points, generated from an externally defined symbol. Here, the symbol must not be identified using the EXTRN instruction.
V-type constants may be used for branching into other assembly units, but not for references to data in other assembly units.

Data references are typically achieved via the COM, DXD or XDSEC instructions (see "ASSEMBH (BS2000) Reference Manual" [1]).

During a program run, the general-purpose registers 0-15 are only available collectively to all modules linked into a program. These registers represent the common communication level. The following requirement must therefore be met for program linking:
All general-purpose registers must be available to all subroutines.
This means that when a branch is made from one module to a subsequent module, the register contents of the calling module must be saved and then reloaded on returning from the called module.

For more information on linking assembler programs, refer to ASSEMBH (BS2000) Reference Manual [1], section 3.2, "Program sectioning and program linking".

7.1.1 Interfacing with other languages

- When assembler is called from some other language:
  This means that the assembler program must take the parameter passing conventions of the calling language into account and restore registers on the return accordingly.
- When assembler calls some other language:
  This is achieved via transfer routines, or the assembler program must take the parameter passing and register conventions of the called language into account. The language environment, i.e. the runtime system, of the called language must be initialized.

Interfacing with other languages such as COBOL, C, and FORTRAN is dealt with in the individual User Guide for each of these languages.

7.2 Linking structured assembler programs

The procedure and data principles of structured programming (see "ASSEMBH (BS2000) Reference Manual" [1]) allow several subroutines (i.e. procedures) to be linked to a main program (main procedure). A procedure starts with @ENTR and is terminated with @END (static procedure end). @PASS calls another procedure so that parameters can be passed to it. @EXIT terminates the called procedure (dynamic procedure end) and returns control to the calling procedure.

The following diagram illustrates the relationship between the static program structure and the dynamic linking of procedures.

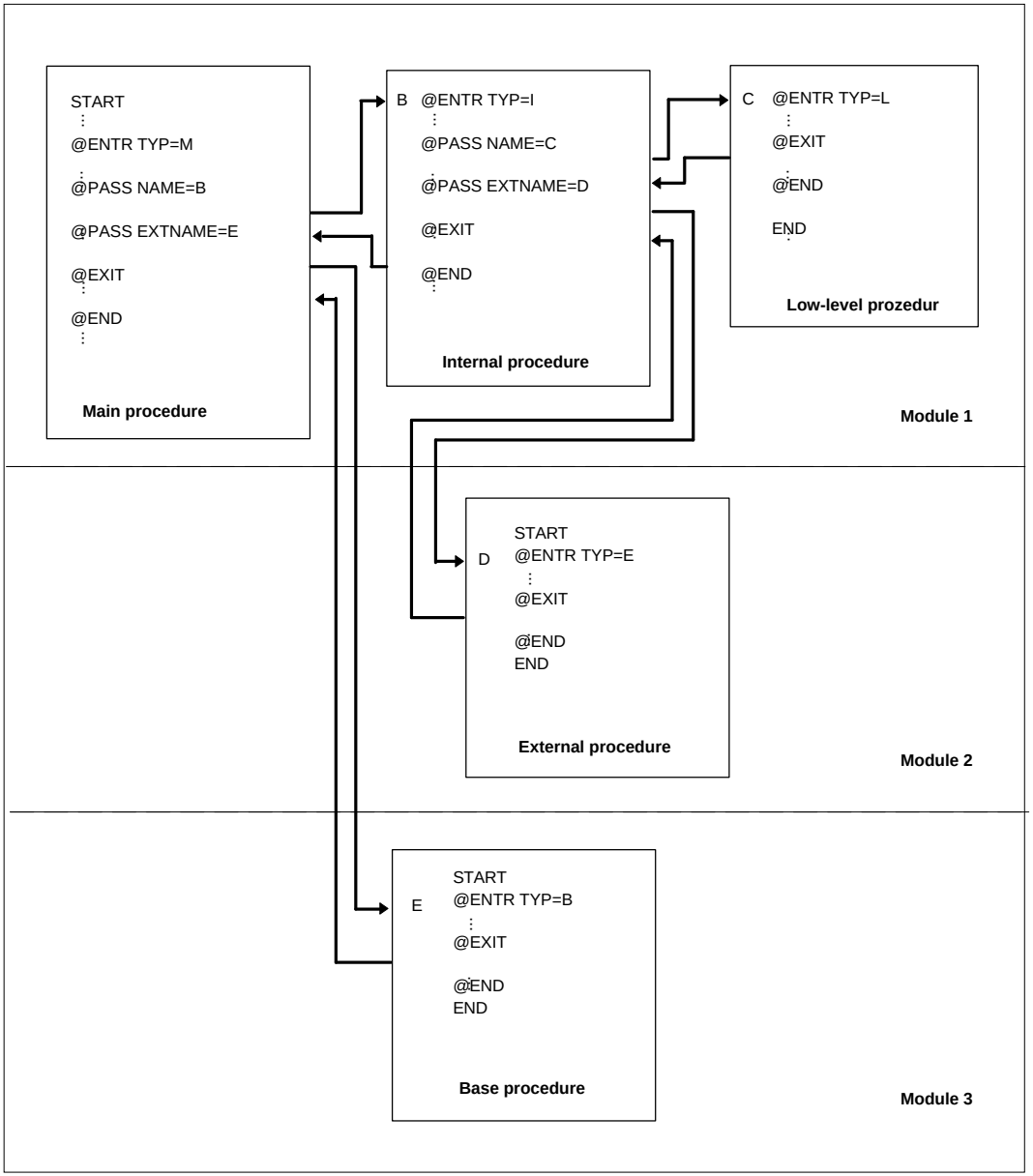


Fig. 7-1: Static program structure and dynamic procedure linkage

7.2.1 Interfacing structured assembler programs with C programs

In the case of C programs, there is an option of using structured assembler programs that behave like C programs, i.e. which comply with C conventions with regard to stack management and the supply of parameters.
The parameter &ENV=C of the @ENTR macro (see "ASSEMBH (BS2000) Reference Manual" [1]) generates code which calls the program manager for C programs to manage the save area and the stack.
The program manager for C programs is also called by the @EXIT and @END macros.
The @PASS and @PAR macros ensure that the supply of parameters is generated in accordance with C conventions.
The @DATA macro can only be used under restrictions.
If memory of storage class C (controlled) is used via the @DATA and @FREE macros, the C environment must be initialized, i.e. the main program must be a C program.

The following points must be additionally observed with regard to the use of registers, memory requests, and the passing of parameters:

- Register 12 must not be used.
  The parameter LOADR12=YES of the @ENTR macro loads the address of the program manager for C programs into register 12.
- Register 13 must not be used.
  The C program manager uses register 13 as the runtime stack register.
- Storage class A (automatic) must not be declared.
- Only type M and E procedures are permitted.
- Parameter passing may only be done in STANDARD form, i.e. register 1 holds the address of the parameter list.
  This form of passing is mandatory for C programs, i.e. the PASS parameter of the @ENTR or @PASS macro must not be changed to OPTIMAL form.

7.2.2 Interfacing structured assembler programs with COBOL and FORTRAN programs

If a structured assembler program that is not exclusively made up of type B, L and D procedures is to be called from a COBOL or FORTRAN main program, the entry IASSIN must be invoked once in order to initialize the assembler runtime system before the first call.

The initialization entry IASSIN performs the same functions as in the case of a structured assembler main program (@ENTR TYP=M).
The IASSIN call is normally made without parameters, and a standard size initial stack is created.

Notes

– To ensure compatibility with existing COBOL and FORTRAN objects with COLBIN calls, this entry is supported like IASSIN, where KL5SP specifications are ignored.

– If the size of the initial stack is to be defined, the IASSIN entry must be specified with a parameter:

  in COBOL, of type COMPUTATIONAL PIC 9(n) with $5 \leq n \leq 9$, and
  in FORTRAN: of type INTEGER.

  Like the STACK specification with @ENTR TYP=M, this parameter defines the size of the initial stack in bytes where:

$$\text{Size of initial stack} = \begin{cases} 1 \text{ page} & \text{parameter } n \leq 0 \\ n \text{ bytes} & \text{parameter } n > 0 \end{cases}$$

Parameter passing to structured assembler procedures

Since parameters are always passed in STANDARD form by COBOL and FORTRAN programs, they must also be accepted in STANDARD form (see "ASSEMBH (BS2000) Reference Manual" [1]).

Calling structured assembler procedures

Call from COLUMBUS-COBOL or COBOL

with parameters: @PASS name:TYP=E:USING parameters or
CALL "name" USING parameters.

without parameters: @PASS name.:TYP=E or
CALL "name".

Call from COLUMBUS-FORTRAN or FORTRAN

with parameters: @PASS name(parameters) or
CALL name(parameters)

without parameters: @PASS name or
CALL name

When type B, L, or D procedures are called, the user must ensure that the register contents are saved and restored by the called procedure.

Return to the calling COBOL or FORTRAN program

The structured assembler procedure returns control to the calling program with

```
@EXIT
```

The operands RC, RESTORE, and TO are not permitted. The operand PROG=FORTRAN must be specified on returning to the FORTRAN segment.

7.2.3 Interfacing structured assembler programs with assembler programs

Parameter passing to structured assembler procedures

– Acceptance form: OPTIMAL
  Registers 1 through 4 hold the parameters
  or register 1 holds the address of a parameter address list for the fourth and following parameters.
  Registers 2 through 4 hold parameters 1 through 3.

– Acceptance form: STANDARD
  Standard parameter interface: register 1 holds the address of the parameter address list.

Calling procedures without the runtime system (type B/L/D)

Structured assembler procedures are called via the standard interface: register 15 holds the procedure address, and register 14 holds the return address.

Structured assembler procedures can thus be called with

```
              ⎡A⎤
L     15,= ⎨V⎬ (procedure-name)
BALR 14,15    ⎣⎦
```

It is the user's responsibility to ensure that the register contents are saved and restored by the called procedure.

Calling procedures with the runtime system (type E/I)

These procedures are called like those without the runtime system (except that the register contents are saved and restored by the assembler runtime system). As with the interfacing of COBOL and FORTRAN programs with structured assembler programs, the runtime system must also be initialized in this case by calling the entry IASSIN once via a standard interface before the first call to a structured assembler procedure.

Interface for IASSIN = standard interface

Register 1	Points to a word containing either 0 (no parameter) or the address of the parameter (in word format). The meaning of the parameter entry is the same as when interfacing COBOL or FORTRAN programs (see section 7.2.2).
Register 15	Holds the address of the runtime entry IASSIN.
Register 14	Holds the return address.
Register 13	Points to the default register save area.

Register save area

Word 1	Internally used
Word 2	Points to predecessor, or holds 0 if no predecessor exists
Word 3	Points to successor
Words 4-18	Save area for registers 14,...,12
Word 19	reserved
Word 20	contains the pointer to the PCD (see section 7.3.2). The pointer is returned by the runtime system.

Return to the calling assembler program

The structured assembler procedure returns control to the calling program with

```
@EXIT
```

The operand TO is not permitted.

Example of a call to IASSIN and a structured assembler procedure

1. Without parameter passing and with creation of a standard initial stack

```
                    .
                    .
                    LA          13,SAVE
                    .
                    .
                    LA          1,PARAMLIS
                    L           15,=V(IASSIN)
                    BALR        14,15
                    .
                    .
                    L           15,=V(procedure-name)
                    BALR        14,15
                    .
                    .
PARAMLIS            DC          A(0)            No parameter passing
                    .
                    .
SAVE                DS          OD
                    DC          3F'0'
                    DS          17F
                    .
                    .
```

2. With parameter passing and an initial stack of 10000 bytes

```
                    .
                    .
STACK               EQU       10000
                    .
                    .
                    LA        13,SAVE
                    .
                    .
                    LA        1,PARAMLIS
                    L         15,=V(IASSIN)
                    BALR      14,15
                    .
                    .
                    L         15,=V(procedure-name)
                    BALR      14,15
                    .
                    .
PARAMLIS            DS        OF
                    DC        X'80'
                    DC        A(PAR1+X'80000000')
                    .
                    .
PAR1                DC        A(STACK)
                    .
                    .
SAVE                DS        OD
                    DC        3F'O'
                    DS        17F
                    .
                    .
```

Note

The address of the same register save area must be passed in register 13 to the called procedure and the runtime entry IASSIN !

7.2.4 Interfacing COBOL and FORTRAN program segments with structured assembler programs

Parameter passing from structured assembler procedures

COBOL and FORTRAN program segments accept parameter lists in STANDARD form only. If a COBOL segment is called without parameter passing, register 1 must be loaded before the call with the address of a word that contains 0. For a call to a FORTRAN segment without parameter passing, register 1 must be loaded with the value 1 before the call.

Call from structured assembler procedures

The structured assembler procedure passes control to the COBOL or FORTRAN segment with the instruction @PASS EXTNAME=.

When a call is made from type B, L, and D procedures, the user must ensure that register 13 contains the address of a save area.

Return to the calling (structured) assembler procedure

Return from	COLUMBUS-COBOL or COBOL @EXIT/@END or EXIT PROGRAM
Return from	COLUMBUS-FORTRAN or FORTRAN @EXIT/@END or RETURN/END

7.2.5 Interfacing assembler program segments with structured assembler programs

Parameter passing from structured assembler procedures

– Passing form OPTIMAL
  Registers 1 through 4 hold the parameters or register 1 holds the address of a parameter address list for the fourth and following parameters.
  Registers 2 through 4 hold parameters 1 through 3.

– Passing form STANDARD
  Standard parameter interface: register 1 holds the address of the parameter address list.

Call from structured assembler procedures

The structured assembler procedure passes control to the assembler segment with the instruction @PASS EXTNAME=. Register 14 then contains the return address; register 15 the address of the called program segment.

If the call is made from type B, L, and D procedures, the user must ensure that a register is loaded with the address of a save area. For a call from type M, E, and I procedures, register 13 is loaded with the address of a save area.

Return to the calling (structured) assembler procedure

Register 14, which was loaded with the return address by @PASS, must be used for the return.

7.3 The program communication interface ILCS

The program communication interface ILCS (Inter-Language Communication Services) standardizes communication between the main program and the external subprograms, and also between the various subprograms, in a language-independent fashion. It allows the user to write any program segment in any desired ILCS-compatible programming language without need for special precautions (such as activation of language initialization routines, connection modules etc.).

ILCS is a combination of software and interface convention:
Firstly, it contains runtime routines which are combined in a PLAM library. Secondly, it also guarantees the communication interface corresponding to the "standard linkage conventions in BS2000"; in other words, each module generated by a compiler with ILCS capability is prepared in accordance with the standard linkage conventions for interfacing with programs written in the same language and in different languages.

The library of ILCS runtime routines is supplied with every compiler that has ILCS capability - as an additional runtime system so to speak.

Specifically, ILCS offers the following functions:

- multilateral convention for interfacing programs written in different languages
- uniform guidelines for event handling
- storage management (stack and heap memories)
- handling of the program mask

The present section describes only the ILCS program interfacing function used by ASSEMBH structured programming, with the basic ILCS data structures.

Note

Programs translated by ILCS-compliant compilers must be linked by means of ILCS to form a program system. If a program system contains programs which do not behave in conformance with ILCS conventions, these programs may need to be restructured so as to conform to the ILCS conventions. If this is not done, there is a danger of incompatibility - at least when linking programs written in different languages.

7.3.1 ILCS register conventions

Register loading on program call

The following table gives an overview of the register loading performed by the calling program before the called program is entered.

Register number	Contents
0	Number of parameters
1	Start address of the parameter address list
2 - 12	Program data
13	Start address of the save area of the calling program
14	Address of the return point to the calling program
15	Address of the entry point in the called program
PM	Program mask: Value from PCD field "program mask"

Register loading on returning to calling program

The following table gives an overview of the register loading performed by the called program on returning to the calling program.

Register number	Contents
0 - 1	Return values of integer functions or undefined
2 - 14	Same as under loading on program call
15	Undefined
PM	Program mask: Value from PCD field "program mask"

7.3.2 ILCS data structures

Save area

The calling program provides the address of a save area in which the called program can place its current register values. The called program sets up a new save area and chains the two save areas.
The format of the save area is as follows:

Byte	Contents
1-4	Byte 1: Bit 1: activity bit (1: program active, 0: program inactive) Bits 2-7: reserved Bit 8 = normally 0 Byte 2: Version = X'01' Bytes 3 and 4: X'FEFF'
5-8	Start address of the save area of the calling program. In the **first** calling program, this field contains -1.
9-12	Start address of the next (chained) save area, if applicable.
13-16	Contents of register 14
17-20	Contents of register 15
21-24	Contents of register 0
25-28	Contents of register 1
29-32	Contents of register 2
.	
.	
69-72	Contents of register 12
73-76	Reserved for FOR1
77-80	Address of the PCD
81-84	Address of the EHL (Event Handler List): If no EHL is defined, the field contains the value -1.
85-128	Reserved

Prosys common data area (PCD)

The PCD is a common data area which is available to all programming languages. The size of the PCD is 4096 bytes.
The first part contains the data areas used by ILCS, including the "program mask" field (in byte 148), which is preset to the value X'0C'. The second part of the PCD contains the programming language areas, each 128 bytes long, which are available to the runtime systems of the different languages.

7.3.3 Initialization of the program system

The initialization of a program system takes place in two stages: First, the main program calls the appropriate ILCS initialization routine. This ILCS initialization routine then in turn calls all the requisite language-specific initializations so that the language environments required for the entire program system are set up prior to execution of the first program statement.

7.3.4 Program mask handling by ILCS

The program mask for program execution is set to the value of the PCD field "program mask" (preset to X'0C') during the course of initialization. If it is changed during program execution, it must be reset prior to the next program call or transfer of control to the value of the PCD field "program mask".

7.3.5 Parameter passing in ILCS program systems

There are significant differences in the semantics of the data types used in the various programming languages that can be interfaced by means of ILCS. The table below lists those data types which have the same form of data representation in the individual programming languages and can therefore be passed as parameters without problems. When using other data types as parameters, a precise knowledge of the relevant form of data storage is essential in order to ensure correct program execution.

Compiler	Data types			
	Binary Word	Floating-point Word	Floating-point Doubleword	String
COBOL85	PIC S9(i) COMP SYNCHRONIZED 5<=i<=9	COMP-1	COMP-2	USAGE DISPLAY
FOR1	INT*4	REAL*4	REAL*8	CHAR*i
Pascal-XT	long_integer	short_real	long_real	packed array [<range>]of char
PLI1	BIN FIXED(31) ALIGNED	BIN FLOAT(21) DEC FLOAT(6)	BIN FLOAT(53) DEC FLOAT(16)	CHAR(i)
C	long	float	double	char <var> [<size>]
ASSEMBH (@ macros)	F	E	D	C
RPG3	Binary array with 0 decimal places	——	——	Alphanum. array (fixed length)

The data must always be stored properly aligned; i.e. 32-bit integers in binary representation are aligned on a word boundary, floating-point numbers on a word or doubleword boundary, strings on a byte boundary. The lengths of strings are constant and known to the called program.

Parameters are passed "by reference", i.e. the address of the data item is transferred. The calling program creates a list of the transferred addresses. The number of parameters is transferred in register 0, the address of the list in register 1 (see section 7.3.1).

Passing function return values

Return values from integer functions are passed in registers 0 and 1, and return values from floating-point functions in floating-point register 0.
Passing return values using other data types in registers 0 and 1 is possible, but is not defined by ILCS. How they are represented is a matter for the various programming languages.

7.3.6 Notes on linking of ILCS program systems

Static linking

If a program system exclusively contains structured ASSEMBH programs, it is sufficient - as previously - to link in the ASSEMBH runtime library (SYSLIB.ASSEMBH.012) by means of the RESOLVE statement of TSOSLNK.

If a program system contains programs in different languages, the initialization routine IT0INITS must be linked in explicitly from the ILCS library.

Dynamic linking

Program systems exclusively containing programs written in the same language can - as previously - be dynamically linked to the runtime library by means of the TASKLIB assignment.

Program systems containing programs in different languages can be linked dynamically if the user ensures that the ILCS initialization routine IT0INITS is contained in the runtime library assigned with TASKLIB.

For dynamic linking using DBL (from BS2000 Version 10.0 on), the ILCS library can be assigned as a further library to be searched by means of the link name BLSLIBnn in RUN-MODE=ADVANCED.

Linking prelinked modules

When prelinked modules are linked, the ILCS routine IT0INITS may only be linked into the prelinked module that contains the main program. The entry points and external references of the prelinked modules must remain visible.

7.4 Program interfacing of structured assembler programs via the ILCS interface

In ASSEMBH Version 1.1A the macros for structured programming (@ macros) have been extended so that the user can now also write ILCS-compliant programs in assembler.
Structured programming is not supported by ASSEMBH-BC !

The standardized procedure interface has the following advantages for the user:

- assembler program segments can be used in any desired ILCS-compliant language environment
- passing of parameters and register conventions are standardized for all languages
- the initialization call COLBIN of ASSEMBH-RTS is not required
- the language-specific macros (such as CENT, CEX and CCALL of C), which previously supported programming in assembler in the particular language environment, can all be replaced by the @ macros, since an ILCS-compliant assembler program behaves the same with respect to all other languages in an ILCS environment.

Along with the conversion of the procedure interface to the ILCS conventions, the following new ILCS facilities are available to the assembler programmer through new @ macros in ASSEMBH V1.1A (see "ASSEMBH Reference Manual" [1]).

- uniform event handling
- uniform contingency handling
- uniform STXIT handling
- program mask handling
- setting of monitoring job variables
- language initialization for dynamically loaded modules
- activation of user-own routines for reserving and releasing memory for stack and heap
- activation of user-own termination routines
- specification of the minimum stack extent size

The new @ macros generate the calls to corresponding entries in the standard event handler (SEH), the standard contingency handler (SCH) and also in the standard STXIT handler (SSH) of the ILCS interface.

User-own handling routines can thus be activated and deactivated on the procedure level. The call for these procedures is effected in accordance with ILCS conventions.

7.4.1 Creating an ASSEMBH ILCS object

The following prerequisites must be satisfied before an executable ILCS object interfacing with the ASSEMBH-RTS (ASSEMBH runtime system) can be generated using ASSEMBH:

- If the object contains an assembler main procedure (TYP=M), this must be assembled with the @ENTR parameter ILCS=YES in order to set up the ILCS environment and initialize the ASSEMBH-RTS and the runtime systems of all other languages involved.
- If the ASSEMBH ILCS object is called by a non-assembler procedure, this procedure must first have initialized the ILCS environment and thus the ASSEMBH-RTS.
- Procedures of type E/I must be assembled with the parameter ILCS=YES.
- The ASSEMBH runtime library must be linked in (see section 7.3.6).

7.5 ILCS linkage combinations

7.5.1 ILCS object calls ASSEMBH ILCS object

Calling procedure:
The ILCS environment has been initialized by the ILCS non-assembler procedure. This means that language-specific initialization of the ASSEMBH runtime system (ASSEMBH-RTS) will also have been performed.

The COLBIN call previously required in the non-assembler procedure can be omitted.

Called procedure:
Procedure prolog:
By means of the macro @ENTR with ILCS=YES. The parameters are expected in STANDARD format using "call by reference".

Procedure epilog:
If the @ENTR parameter RETURNS=YES was set, the function value is copied from R1 to R0. Registers R2 through R14 are restored.

7.5.2 ASSEMBH ILCS object calls ASSEMBH ILCS object

Calling procedure:
Initialization of the ILCS environment and the ASSEMBH-RTS has taken place in the main procedure (type M) by means of @ENTR with ILCS=YES. Parameters are passed in STANDARD format using "call by reference".

Called procedure:
Procedure prolog:
By means of the macro @ENTR with ILCS=YES. The parameters are expected in STANDARD format using "call by reference".

Procedure epilog:
If the @ENTR parameter RETURNS=YES was set, the function value is copied from R1 to R0. Registers R2 through R14 are restored.

7.5.3 ASSEMBH ILCS object calls non-ILCS ASSEMBH object

Calling procedure:
Initialization of the ILCS environment and the ASSEMBH-RTS has taken place in the main procedure (type M) by means of @ENTR with ILCS=YES. Parameters are passed in STANDARD format using "call by reference".

Called procedure
Procedure prolog:
By means of the macro @ENTR with ILCS=NO (default). The parameters are expected in STANDARD format using "call by reference".

Procedure epilog:
The function value is not copied from R1 to R0.

Note

Called procedure:
- may contain no @EXIT with TO operand
- the number of parameters is passed in R0

Calling procedure:
- may not assume that the ILCS conventions are observed by the called procedure; for example, the function value is not copied from R1 to R0

7.5.4 Non-ILCS ASSEMBH object calls ASSEMBH ILCS object

Calling procedure:
Initialization of the ILCS environment and the ASSEMBH-RTS has taken place in the non-ILCS main procedure (type M) by means of @ENTR with ILCS=NO. Parameters are passed in STANDARD format using "call by reference".

Called procedure
Procedure prolog:
By means of the macro @ENTR with ILCS=YES. R0 generally does not contain the number of parameters. The parameters are expected in STANDARD format using "call by reference".

Procedure epilog:
If the @ENTR parameter RETURNS=YES was set, the function value is copied from R1 to R0. Registers R2 through R14 are restored.

Note

Called procedure:
- may not expect the number of parameters in R0

7.5.5 Non-ILCS object calls ASSEMBH ILCS object

Calling procedure
The COLBIN call for initializing the ILCS environment is required.
Parameters are passed in STANDARD format using "call by reference".

Called procedure same as section 7.5.4

Note
Called procedure:
PROG=FORTRAN may not be specified in the @EXIT macro.

7.5.6 Long-jump (@EXIT with parameter TO)

If a program contains ILCS objects and non-ILCS ASSEMBH objects, the user must ensure that no long-jump is present anywhere within the non-ILCS ASSEMBH objects. Otherwise, program errors are possible!

8 The ASSEMBH diagnostic routine ASSDIAG

Not supported by ASSEMBH-BC !

8.1 Application

The diagnostic routine ASSDIAG is interactively invoked by ASSEMBH within the framework of the assembler correction cycle. It fulfils the following basic functional requirements:

- Implicit initiation by ASSEMBH under SDF (or COMOPT) control as soon as a certain error weight is reached.
- Output of diagnostic information on a preceding assembly.
- Formatted I/O in interactive mode.
- Use of the file editor EDT to correct the source program in interactive mode.
- Restart of ASSEMBH with the set options.

ASSDIAG is dynamically loaded and activated by ASSEMBH from the library <userid>.SYSLNK.ASSEMBH.011.

Activation is prepared by specifying the following option (see section 2.4.7):

CORRECTION-CYCLE=YES(ACTIVATION-WEIGHT=<error-weight>)
(for COMOPT, with *COMOPT ADIAG=n)

This means that ASSDIAG will be started if the corresponding error weight is detected during the assembly.
If CORRECTION-CYCLE=YES(ACTIVATION-WEIGHT=ALWAYS)
(or *COMOPT ADIAG=0) is specified, ASSDIAG is started irrespective of the errors that occur.

Software requirements

To enable the correction of source lines, the diagnostic routine works with the file editor EDT. The appropriate version required for this purpose is given in the release notice.

8.2 Definition of terms

Diagnostic file

A temporary file that is created by ASSEMBH and deleted on termination of ASSDIAG.

ASSDIAG command

Instructions to the diagnostic routine to perform certain services.

Error class

Every error that is detected by ASSEMBH during the assembly of a program falls under one of the error classes described below:

Code	Description	Error classes	
		SDF	COMOPT
NOT	NOTE Successful program run possible	NOTE	-
WAR	WARNING Successful program run possible	WARNING	1
SIG	SIGNIFICANT ERROR Program run possible, but with errors	SIGNIFICANT	2
SER	SERIOUS ERROR Program run not possible	SERIOUS	3
FAT	FATAL ERROR Assembly aborted Diagnostic file incomplete	-	-
FAL	FAILURE - internal assembler error Assembly aborted Diagnostic file incomplete	-	-
MNO	MNOTE - message Message generated via variable symbols, with a severity code (MNOTE number) that can be used to create a corresponding error class. (Reactions as described for classes above)	-	-

Flag type

One of the letters A-Z, with which a flag code begins.

Flag code

Flag type followed by one or two digits with which an error is identified by a flag.

Message text

Verbal description of a flag code.

8.3 Starting the diagnostic routine

The option CORRECTION-CYCLE=YES(ACTIVATION-WEIGHT=<weight>)
(or the *COMOPT entry ADIAG=n) enables the user to have the diagnostic routine ASSDIAG started at the end of an assembly unit on the basis of the highest error class that occurs during the assembly.

The values <weight> and <n> have the following significance:

<weight>	<n>	ASSDIAG is started
ALWAYS	0	following an assembly, regardless of the assembly result
NOTE	-	at the occurrence of error class NOTE or higher
WARNING	1	at the occurrence of error class WAR or higher
SIGNIFICANT	2	at the occurrence of error class SIG or higher
SERIOUS	3	at the occurrence of error class SER

After the diagnostic file is opened, ASSDIAG evaluates the information that is beyond the scope of the assembler listing, outputs this information, and waits for the input of ASSDIAG commands.

The opening screen contains the following entries:
– name of the diagnostic file
– creation date
– name of the source program (source module)
– version number of the assembler
– number of flags that occurred, classified by error weight

Only the flags that remain detectable by the SDF specification MIN-MESSAGE-WEIGHT= (or the *COMOPT entry ERRPR=) are displayed and may be processed further.

8.4 Interrupting the program run

When the user is prompted for the input of an ASSDIAG command, he or she can press the BREAK key at the terminal and branch to system mode, where BS2000 commands may be entered (this can also be done with the ASSDIAG command SYSTEM; see section 8.5.9).
The interrupted ASSDIAG run can be subsequently continued by entering the command RESUME-PROGRAM.

8.5 ASSDIAG commands

ASSDIAG is controlled by means of commands. These commands, which must always be entered in the command line of the screen mask, are read with the aid of the WRTRD macro.
The commands can be shortened to any extent, down to a single character, provided uniqueness is ensured.

General format:

```
<command-name> [<parameter>[,<parameter>...]]
```

Scroll function
If the output extends over several screens, it is possible to scroll to the next screen by means of a null input.

Overview of ASSDIAG commands

Command	Function
CDT	Calls the file editor EDT for correcting source lines
CONTINUE-CDT	Continues correction processing at the interrupt point
DISPLAY	Outputs error causes and numbers of affected instructions on the display terminal
END	Terminates the diagnostic routine
HELP	Lists and explains ASSDIAG commands
LIST	Identical to DISPLAY, but with output to SYSLST
PRINT	Lists instructions
RERUN	Starts the assembler with the applicable COMOPTs or SDF options
SYSTEM	Executes a system command
TAGS	Lists all symbols that are undefined or multiply defined
XREF	Outputs cross-reference data

8.5.1 CDT command

Format

```
C[DT] [<parameter>]
```

Parameters

```
                ⌈ALL            ⌉
                |SO[URCE]       |
                |<error-class>  |
<parameter>::={                 }
                |<statement-no.>|
                |<flag-code>    |
                ⌊<flag-type>    ⌋
```

Function

The file editor EDT is started as a subroutine. The source program file or the source element of a PLAM library is opened and presented for correction.

1. Parameter ALL (default)

   The error information for all errors that have occurred is merged into the EDT work file, following the source statement causing the error, with the attributes ’reduced brightness’ and ’write-protected’. The information comprises flag code, error weight, message number and message text.
   The errored source line itself is set to ’normal brightness’ and ’overwritable’. The EDT window is positioned to 2 lines before the first error line in order to show the error in some context.
   In the case of errors in macro and COPY elements, the first-level call in the source is flagged as errored. The error information here is supplemented by the incorrectly generated line. It is provided with the statement number for subsequent actions.

2. Parameter SOURCE

   The source is presented for correction in the EDT work file without error information, and can be processed using the EDT facilities. It is the responsibility of the user to search for and correct errored source lines.
   Processing of library elements (macro, COPY) is not possible.

3. Parameter < >

   As described under point 1., but only for the specified error scope.

CDT command processing

Scroll function:

In functions 1. and 3., the <K1> key can be used to position the EDT window to the next errored source line.

Terminating CDT processing:

There are several ways of terminating EDT and saving the processed assembler sources (source/macro/COPY):

1. EDT command RETURN or
   termination of scroll function by last <K1> key

   The opened assembler sources are written back and EDT processing is terminated. With PLAM elements, the version designation is not changed.

2. EDT command HALT

   EDT is terminated. The processed elements are not written back.

3. By the ASSDIAG commands RERUN or END

Interruption:

1. ASSDIAG command other than CDT
2. CDT statement

Direct ASSDIAG command entry in the CDT correction screen

Entering ASSDIAG commands during CDT correction processing with EDT causes this processing to be interrupted and the corresponding ASSDIAG function is invoked. Continuation of the interrupted CDT correction processing at the point previously reached can only be effected by entering the ASSDIAG command C[ONTINUE]-C[DT], which has no parameters. The ASSDIAG command CDT is not permitted from within the CDT correction screen and will be rejected. Only the C-C command is permitted following an interruption in correction processing.
The commands LIST and SYSTEM are permitted in the correction screen as understood by EDT.

Special points concerning ASSDIAG commands issued from the CDT correction screen:

Abbreviations for ASSDIAG commands issued from the CDT correction screen differ from the possible abbreviations that otherwise apply. See the second line in the format description of the commands (section 8.5.3 ff).

– END command

Following a query and positive response, all files/library elements opened for correction are written back to the volume and closed. Otherwise, the program segments are closed unchanged. The compile correction cycle is terminated. The assembler listing is output if the parameter L is specified.
In order to maintain unambiguity, the command may not be entered in abbreviated form within correction mode.

– RERUN command

All files/library elements opened for correction are written back to the volume and closed without query. The compile correction cycle with the modified program segments is activated again.

8.5.1.1 CDT statements

CDT statements allow additional processing operations on the assembler program elements concerned to be included in the current source correction. These statements are permitted only during CDT correction processing and result in an error message if used outside the CDT command. The statements (1.-4.) may be specified from the CDT correction screen in the command line:

On opening the element, the following information is presented in the second window (applies to statements 2.-4.):
– library name of the opened element
– element name of the opened element
– version and type of the opened element
– element name of the element written back

CDT statement processing

Scrolling:	For SHOW-DEF in the bottom command line with EDT scroll function (+/-)
Terminate interruption:	<K1> key
Return point:	Last CDT command processing
Terminate:	RETURN, RERUN, END, HALT

CDT command processing or statement processing interrupted by ASSDIAG command:
Terminate interruption: CONTINUE-CDT
Return point: Last CDT command processing

1. Overlaying the definition line(s) on the source

Format

```
S[HOW]-D[EFINITION] <name>
```

Function

By means of this statement, all definition lines from the list having the symbol <name> are placed in the EDT work file. The EDT work file is displayed as a second window at the bottom of the screen. The list lines merged in are represented in compressed form and comprise:
Flag column, location counter, statement number, macro/COPY level and source statement.
EDT statements for scrolling within the EDT work file (+, -, ++, --) must be entered in the input line of the EDT work file; other CDT statements should be entered in the input line of the upper window (e.g. EDIT-DEFINITION).

2. Correcting a definition line

Format

```
E[DIT]-D[EFINITION] <name>
```

Function

This statement positions the EDT window on the first definition line of the specified symbol in the corresponding source element (source/macro/COPY). If the definition line is contained in a macro/COPY element, the element (if not already loaded) is read into a free EDT work file.

<name> Denotes the first occurrence of the symbol definition <name> in the source. If a definition line is to be processed and it is not the first definition line of the symbol <name>, this can be achieved by means of the CDT statements SHOW-DEFINITION <name> and EDIT <statement-no.>.

After EDIT-DEFINITION, processing is continued by means of the <K1> key.

3. Processing a source or macro/COPY statement with overlaid error information

Format

```
E[DIT] <statement-number>
```

Function

Statement number in the source:
If the specified statement number is located in the source part of the assembly unit, then the work file is positioned on the corresponding source line. The <K1> key then positions onto the next errored statement, or correction processing is terminated.

Statement number in the macro/COPY element:
The statement number is also output in the overlaid error information relating to the generated statement for the macro call. This information allows the relationship to the source element and the line number to be established. The statement causes the corresponding macro/COPY element to be read into a free work file; this is then positioned on the errored source line. With the aid of the overlaid error information the user can correct the line. Function key <K1> causes processing to be continued at the position reached prior to the EDIT statement.

4. Processing a macro/COPY element without overlaid error information

Format

```
       ⎧ M[ACRO] ⎫
E[DIT]-⎨         ⎬ <name>
       ⎩ C[OPY]  ⎭
```

Function

The element is processed without overlaid error information. The EDT window is positioned to the start of the element. The EDT work file is assigned in the same way as for processing with overlaid error information. The return to continued processing following element correction is effected with the <K1> key. Like-named macros are not supported.

8.5.2 CONTINUE-CDT command

Format

```
C[ONTINUE]-C[DT]
```

Function

This command allows CDT correction processing interrupted by an ASSDIAG command input to be resumed at the point of interruption.
If no CDT correction processing was interrupted, the command will be rejected as invalid.

8.5.3 DISPLAY command

Format

```
⎧D[ISPLAY]⎫
⎨         ⎬ [<parameter>]
⎩DIS[PLAY]⎭
```

The second line gives the minimum abbreviation allowed for command input from the CDT correction screen (see section 8.5.1).

Parameters

```
                ⎧NOT                                  ⎫
                ⎪WAR                                  ⎪
                ⎪SIG                                  ⎪
                ⎪SER                                  ⎪
<parameter>::=  ⎨FAT                                  ⎬
                ⎪FAL                                  ⎪
                ⎪MNO                                  ⎪
                ⎪<statement-no.> [-<statement-no.>]   ⎪
                ⎪<flag-type>                          ⎪
                ⎩<flag-code>                          ⎭
```

Function

The selected error causes are displayed on the terminal screen, provided the assembly was not aborted (see "Special case" below).

- Command without parameters
  Listing of all detected flags and MNOTEs, arranged by error class, flag code with message text, and references to the affected statement numbers.
- NOT:
  As above, but only for the error class NOTE.
- WAR:
  As above, but only for the error class WARNING.
- SIG:
  As above, but only for the error class SIGNIFICANT ERROR
- SER:
  As above, but only for the error class SERIOUS ERROR
- FAT:
  As above, but only for the error class FATAL ERROR
- FAL:
  As above, but only for the error class FAILURE

– MNO:
  Listing of all MNOTEs with assigned severity code and text, followed by the statement number.

– <statement-no.> [-<statement-no.>]
  The invalid statement or statements (no MNOTEs) detected within the specified number range are output on the terminal, followed by the respective flag code and message text in each case.

– <flag-type>
  Listing of the errors of a flag type, followed by the precise flag code and message text, and the affected statement numbers.

– <flag-code>
  As above, but only for the specified flag code.

Scroll function
If the output extends over several screens, it is possible to scroll to the next screen by means of a null input.

Special case: Assembly aborted

If the assembly is aborted, a complete diagnostic file cannot be generated. Two different outputs are then possible, depending on the cause of the abortion:

1. Abortion with error weight SERIOUS (i.e. continuation of assembler run not possible):

   The message text is output with notes concerning possible causes.
   All parameters in the DISPLAY command are ignored in this case.

2. Controlled abortion due to the entry of a maximum error weight:

   All errored statements thus far are output, including at least the statement that triggered the abortion (e.g. MNOTE with SEV-CODE=255).
   In such cases DISPLAY parameters are accepted with restrictions (statement references are always excluded).

8.5.4 END command

Format

```
⎡E[ND]⎤
⎨     ⎬ [L]
⎣END  ⎦
```

The second line gives the notation for command input from the CDT correction screen (see section 8.5.1).

Function

The diagnostic routine is terminated, and control is returned to ASSEMBH. If the supplement 'L' is added, the assembler listing is output in accordance with the options specified via SDF (or *COMOPT) control. Files that are open are closed.
Any further assembly unit in the same source will no longer be processed.

8.5.5 HELP command

Format

```
⎡H[ELP]⎤
⎨      ⎬ [<command>]
⎣HEL[P]⎦
```

The second line gives the minimum abbreviation permitted for command input from the CDT correction screen (see section 8.5.1).

Function

Listing of all ASSDIAG commands or description of selected ASSDIAG commands on the data display terminal.

8.5.6 LIST command

Format

```
L[IST] [<parameter>]
```

Parameters

See section 8.5.3, DISPLAY command

Function

Same as the DISPLAY command, but with output to SYSLST.

8.5.7 PRINT command

Format

```
P[RINT] [{<statement-no.>[-<statement-no.>]
         {                                  }][,L][,S]
         {<symbol>[-<symbol>]               }
```

The abbreviation also applies to command input from the CDT correction screen (see section 8.5.1).

Function

Lists a specific statement or range of statements as they would appear in the assembler listing. If no range is specified, all statements are output. However, the rightmost characters of any line extending beyond 80 columns are truncated on the display terminal.

Specifying ’S’ causes only the following information of a line to be output:
- location counter
- statement number
- and source statement (possibly with associated message text line)

Specifying ’L’ causes an additional complete output of the print lines to SYSLST. The flag codes and message texts are inserted after the instructions concerned.

Scroll function
If the output extends over several screens, it is possible to scroll to the next screen by means of a null input.

Note

This may not be possible if the assembly is aborted.

8.5.8 RERUN command

Format

```
R[ERUN]
RER[UN]
```

The second line gives the minimum possible abbreviation for command input from the CDT correction screen (see section 8.5.1).

Function

The RERUN command terminates ASSDIAG and causes ASSEMBH to reassemble the assembly unit with the options that were set for the preceding assembly.

Notes

- The RERUN command is rejected if the source was read in via SYSDTA.
- No RERUN is possible after the abortion of an assembly.
- It makes no sense to start a correction cycle including module output to *OMF, since the module generated by RERUN does not overwrite a module of the same name in *OMF.

8.5.9 SYSTEM command

Format

```
S[YSTEM]<parameter>
```

Parameters

```
<parameter>::='system-command'
```

Function

The system command enclosed in single quotes may be specified with or without a slash. It is executed immediately, and ASSDIAG is continued thereafter, provided this is allowed by the system command that was executed.

All commands that can be called via the CMD macro are permitted (see "BS2000 Executive Macros, Reference Manual" [12])

Note

ASSDIAG remains loaded, and open files are not closed during the execution of the command.

8.5.10 TAGS command

Format

```
⎧T[AGS]⎫
⎨      ⎬ [<type>[,<type>]][,X[REF]
⎩TAG[S]⎭
```

The second line gives the minimum possible abbreviation for command input from the CDT correction screen (see section 8.5.1).

Parameters

```
<type>::=⎧M⎫
         ⎩U⎭
```

Default value: M,U

Function

Displays all undefined (U) and/or multiply defined (M) symbols. If XREF is specified, cross-references are also indicated.

Scroll function
If the output extends over several screens, it is possible to scroll to the next screen by means of a null input.

Note

The function may not be possible if the assembly is aborted.

8.5.11 XREF command

Format

```
X[REF] <parameter>[,<parameter>[,<parameter>]]
```

The abbreviation also applies to command input from the CDT correction screen (see section 8.5.1).

Parameters

```
                ⎡<symbol>[-<specification>]⎤
<parameter>::=  ⎢*<macro-name>             ⎥
                ⎣<literal>                 ⎦

                      ⎡ ⎤
                      ⎢A⎥
                      ⎢R⎥
<specification>::=    ⎢ ⎥
                      ⎢W⎥
                      ⎢E⎥
                      ⎣O⎦
```

Function

The cross-references for the specified symbols, macro names or literals are displayed on the terminal.
If the output of references with attributes was specified for the assembly, a specific selection of cross-references may be requested for symbols:

A: Address accesses
R: Read-only accesses by instructions
W: Write accesses
E: Symbol of EQU/ORG instruction
O: Other accesses via assembly-language instructions

Scroll function
If the output extends over several screens, it is possible to scroll to the next screen by means of a null input.

Note

This may not be possible if the assembly was aborted or if no XREF was requested for the assembly.

8.6 Formatted screen I/O

8.6.1 Basic structure of ASSDIAG formats

```
CMD:                              A S S D I A G                          VERSION: V1.2A00
-------------------------------------------------------------------------------------------

   NAME OF SAVLST  :       :A:$ASSEMBH.TMP.SAVLST.ASSEMBH.4THG.104840
   CREATED         :       94-03-07 10:47:20


   SOURCE MODULNAME:       TESTXREF
   PROGRAM WAS ASSEMBLED BY ASSEMBH   V 1.2A00


   FLAGS WITH ERROR CLASS  MNO  : 1
                           NOT  : 0
                           WAR  : 0
                           SIG  : 16
                           SER  : 0
                           FAT  : 0
                           FAL  : 0




-------------------------------------------------------------------------------------------
CMD:
                                                                              PAGE:
```

8.6.2 Example: DISPLAY command

```
CMD: DISPLAY                      A S S D I A G                          VERSION: V1.2A00
-------------------------------------------------------------------------------------------
CLASS FLAG  MESSAGE AND STATEMENT NUMBERS
FAL          NONE
FAT          NONE
SER          NONE
SIG   B42   ASS0242 'COPY' MEMBER NOT FOUND
            000004
      D7    ASS0407 ALIGNMENT ERROR IN OPERAND
            000011
      U10   ASS2110 SYMBOL   IS UNDEFINED
            000006  000007  000009  000015  000016  000017  000018  000018
            000046  000047  000048  000049  000072  000073
WAR          NONE
NOT          NONE
MNO          MNOTE WITH SEVERITY CODE 0152
            000013



-------------------------------------------------------------------------------------------
CMD:
 END OF OUTPUT                                                            PAGE: 1
```

8.6.3 Example: TAGS command

```
CMD: TAGS                    A S S D I A G                    VERSION: V1.2A00

UNDEFND SYMBOL
 R14
 R15
 R5
 R6
 R7
 R8
 R9
 UNDEF

CMD:
 END OF OUTPUT                                                      PAGE: 1
```

9 The Advanced Interactive Debugger (AID)

9.1 Introduction

Even an assembler program that has no syntax errors may not run as required, as the program may still contain logical errors. The software product AID (**A**dvanced **I**nteractive **D**ebugger) is available to assembler programmers for detecting and eliminating such errors. AID is not subject to special programming requirements and allows the programmer to search a loaded program for errors during its execution and directly make corrections during this process.

Only the main features of AID are discussed in this User Guide. A detailed description of the debugger is provided in the manual "AID, Debugging of ASSEMBH Programs" [2].

AID is characterized by the following features:

1. AID supports "symbolic" debugging, which means that symbols from the source program can be specified in commands instead of hexadecimal addresses, assuming the requisite LSD information was generated during the assembly and subsequently passed to the loaded program (see section 9.2).

   It is not always necessary to load all such information for the entire program together with this program. AID permits LSD information to be dynamically loaded for each assembly unit if the associated modules (with the LSD information) are in a PLAM library. This provides for more efficient use of resources:

   – Less program memory is used, since the LSD information only needs to be loaded when it is required for debugging (memory requirements for a program increase by about five times if LSD information is loaded together with the program).

   – Programs that remain free of errors during debugging need not be reassembled (without LSD information) and linked before being put into productive use.

   – If a program needs to be debugged while it is in productive use, the prerequisite LSD information will already be available without having to assemble and link the program again.

2. AID provides functions which can be used to

   – interrupt the program run at predetermined locations or when defined events occur so that AID or BS2000 commands (subcommands) can be executed,

   – output the contents of fields on the basis of the data definition in the source program,

   – modify the contents of fields.

3. Besides the diagnosis of loaded programs, AID also supports the analysis of memory dumps in disk files.

9.2 Prerequisites for symbolic debugging

To permit debugging on the symbolic level, AID provides a means of addressing symbols which are defined in the source program and which refer to source program lines. Specific information on the symbols must be made available to AID for this purpose. This information consists of two parts:

- the LSD (List for Symbolic Debugging), which lists the symbols and instructions defined in the module, and
- the ESD (External Symbol Dictionary), in which external references of a module are registered.

The generation and forwarding of this information can be initiated or suppressed in each of the following steps:
- assembly with ASSEMBH,
- linking and loading with DLL (up to BS2000 V9.5), DBL as of BS2000 V10.0 or
- linking with TSOSLNK, and
- loading with ELDE or
- linking with BINDER (as of BS2000 V10.0) and loading with DBL

Whereas the ESD information is generated and passed on by default, the LSD information can be made available to AID in two ways:

If the LSD information was generated during assembly, it can either be
- loaded together with the entire program, or
- loaded dynamically for each assembly unit as required, provided the associated modules are in a PLAM library.

The table below shows an overview of both cases with the appropriate commands and operands that must be assigned in each program development step (for more detailed information, refer to the manual "AID, Debugging of ASSEMBH Programs" [2]):

Program development steps	Commands with operands	
	If the LSD information is to be loaded together with the program	If the LSD information is to be dynamically loaded by AID as needed *
Assembly with ASSEMBH	//COMPILE SOURCE=..., TEST-SUPPORT=AID	//COMPILE SOURCE=..., TEST-SUPPORT=AID, MODULE-LIBRARY=...
Linking and loading with DLL/DBL	/LOAD-PROGRAM..., or /START-PROGRAM..., TEST-OPTION=AID	/LOAD-PROGRAM..., or /START-PROGRAM..., TEST-OPTION=NONE
Linking with TSOSLNK	*PROGRAM...,SYMTEST=ALL	*PROGRAM...[,SYMTEST=MAP]
Linking with BINDER (and loading with DBL, see above)	START-LLM-CREATION..., INCLUSION-DEFAULTS= (TEST-SUPPORT=YES) SAVE-LLM LIB=..., TEST-SUPPORT=YES	Dynamic loading of LSD records is not possible for modules in LLM format
Loading with ELDE	/LOAD-PROGRAM..., or /START-PROGRAM..., TEST-OPTION=AID	/LOAD-PROGRAM..., or /START-PROGRAM..., TEST-OPTION=NONE

* This is possible only if the associated modules are in a PLAM library and have been assigned with %SYMLIB.

9.3 Example of a debugging run

This example demonstrates a debugging session with AID for a small assembler program. It is intentionally based on a relatively simple approach so that you can easily see the application and effect of a number of AID commands. The assembler program is listed in section 9.3.1; the debugging run is explained in section 9.3.2.
To enhance readability, inputs appear in bold print.

9.3.1 Assembler program

Objective

The program SUM is to read in up to 10 two-digit numbers and output the resulting total. Input of the number 00 serves as the end criterion.
If more than 10 numbers are entered, a message is issued together with the calculated total.

Source program listing

```
COMPUTE THE SUM OF N NUMBERS (N <= 10)                                                                    11:17:32  94-03-08
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 000000                                       1     SUM      START
                                              2              TITLE 'COMPUTE THE SUM OF N NUMBERS (N <= 10)'
                                              3              PRINT NOGEN
 000000                00000000               4     R0       EQU   0
 000000                00000001               5     R1       EQU   1
 000000                00000002               6     R2       EQU   2
 000000                00000003               7     R3       EQU   3
 000000                00000004               8     R4       EQU   4
 000000                00000005               9     R5       EQU   5
                                             10     SUM      AMODE ANY
                                             11     SUM      RMODE ANY
                                             12              GPARMOD 31
                                             14   2               *,VERSION 010
 000000 0D 20                                15              BASR  R2,R0
 000002                00000002              16              USING *,R2
                                             17     START    WROUT MESS1,END
                                             24   2               *,FHDR VERSION 105 / 1988-06-13
                                             48   2               *,@DCEO      952    900503   53531004
                                             51   1               *,WROUT      005    910215   53121058
 000026 58 50 2176     00000178              52              L     R5,=F'1'
 00002A 5A 50 2176     00000178              53     LOOP     A     R5,=F'1'
 00002E 49 50 2138     0000013A              54              CH    R5,TEN
 000032 47 20 20BE     000000C0              55              BH    ERROR
                                             56     READ     RDATA INPUT, END
                                             63   2               *,FHDR VERSION 105 / 1988-06-13
                                             92   2               *,@DCEI      920    881104   53531002
                                             95   1               *,RDATA      006    910215   53121057
 000062 D5 05 2121213A 00000123 0000013C     96     COMP     CLC   INPUT+4,ZERO
 000068 47 80 207A     0000007C              97              BE    FROM
 00006C F2 11 21232121 00000125 00000123     98     ADD      PACK  PACK,INPUT+4(2)
 000072 FA 31 213C2123 0000013E 00000125     99              AP    TOTAL,PACK
 000078 47 F0 2028     0000002A             100              B     LOOP
 00007C F3 63 2131213C 00000133 0000013E    101     FROM     UNPK  RESUL,TOTAL
 000082 D3 00 21372140 00000139 00000142    102              MVZ   RESUL+6(1),ZONE
                                            103              WROUT MESS2,END
                                            109   2               *,FHDR VERSION 105 / 1988-06-13
```

```
                                        133   2                *,@DCEO      952    900503   53531004
                                        136   1                *,WROUT      005    910215   53121058
                                        137     END      TERM  DUMP=Y
                                        140   2                *,VERSION 010
                                        152     ERROR    WROUT MESS3,END
                                        159   2                *,FHDR VERSION 105 / 1988-06-13
                                        183   2                *,@DCEO      952    900503   53531004
                                        186   1                *,WROUT      005    910215   53121058
 0000E2 47 F0 207A     0000007C         187              B     FROM
                                        188              EJECT
                                        189     *
                                        190     * DEFINITIONS
                                        191     *
 0000E6 0039                            192     MESS1    DC    Y(L'M1+5)
 0000E8 404001                          193              DC    X'404001'
 0000EB C2C9E3E3C540C2C9                194     M1       DC    C'PLEASE ENTER UP TO 10 2-DIGIT NUMBERS. END: 00'
 00011F 000000000000                    195     INPUT    DC    XL6'00'
 000125 000C                            196     PACK     DC    PL2'0'
                                        197     *
 000128 0012                            198     MESS2    DC    Y(L'M2+L'RESUL+5)
 00012A 404001                          199              DC    X'404001'
 00012D E2E4D4D4C57A                    200     M2       DC    C'SUM:'
 000133 40404040404040                  201     RESUL    DC    CL7' '
                                        202     *
 00013A 000A                            203     TEN      DC    H'10'
 00013C F0F0                            204     ZERO     DC    C'00'
 00013E 0000000C                        205     TOTAL    DC    PL4'0'
 000142 F0                              206     ZONE     DC    X'F0'
                                        207     *
 000144 0034                            208     MESS3    DC    Y(L'M3+5)
 000146 404001                          209              DC    X'404001'
 000149 C5E240D2D6C5D5D5                210     M3       DC    C'NO MORE THAN 10 NUMBERS CAN BE PROCESSED'
 000000                                 211              END   SUM
 000178 00000001                        212                    =F'1'
 00017C 9101221427002852                213                    =X'9101221427002852' CONSISTENCY CONSTANT FOR AID
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH                  V1.2A00     ON 1994-03-08 AT 11:15:54
```

9.3.2 Debugging run

Step 1

The Assembler source program SUM in the file SOURCE.TEST is assembled using ASSEMBH. The specified option TEST-SUPPORT=YES causes ASSEMBH to create LSD information and pass it to the object module. The source program is assembled without errors.

```
/DEL-SYS-FILE OMF
/START-PROG $ASSEMBH

%  BLS0500 PROGRAM 'ASSEMBH', VERSION '1.XXXX' OF 'yy-mm-dd' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991. ALL
   RIGHTS RESERVED
%  ASS6010 V 1.XXXX OF BS2000 ASSEMBH READY

//COMPILE SOURCE=SOURCE.TEST,
          TEST-SUPPORT=AID

%  ASS6011 ASSEMBLY TIME: 80 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 102 MSEC

//END

%  ASS6012 END OF ASSEMBH
```

Step 2

Program SUM is to be executed.

```
/START-PROG (*OMF)

%  BLS0517 MODULE 'SUM' LOADED

PLEASE ENTER UP TO 10 2-DIGIT NUMBERS. END: 00
*05
*16
*48
*00
*0
*00
*EN
/
```

The program always branches back to the input, so there must be a program error. The program is interrupted by pressing the K2 key.

Step 3

The program is reloaded with TEST-OPTION=AID so that it can be symbolically tested.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0517 MODULE 'SUM' LOADED

/%IN S'96' <%D INPUT;%STOP>
/%R
```

The %INSERT command is used to set a test point at the line with the statement number 96, i.e. the CLC instruction. Every time the program reaches this address, the contents of field INPUT are to be output.
Following output, the program is to be switched to the STOP status so that new commands can be entered.

The loaded program is started with %RESUME.

```
PLEASE ENTER UP TO 10 2-DIGIT NUMBERS. END: 00
*05

** ITN: #'004B012E' *** TSN: 2069 ****************************************
SRC_REF:    96 SOURCE: SUM       PROC: SUM   ********************************
INPUT           = 00060000 F0F5   ....05
STOPPED AT LABEL: COMP , SRC_REF:  96, SOURCE: SUM ,PROC: SUM

/%R
*48
INPUT           = 00060000 F4F8   ....48
STOPPED AT LABEL: COMP , SRC_REF:  96, SOURCE: SUM ,PROC: SUM

/%R
*16
INPUT           = 00060000 F1F6   ....16
STOPPED AT LABEL: COMP , SRC_REF:  96, SOURCE: SUM ,PROC: SUM

/%R
*00
INPUT           = 00060000 F0F0   ....00
STOPPED AT LABEL: COMP , SRC_REF:  96, SOURCE: SUM ,PROC: SUM
```

Field INPUT contains the correct value in each case. The program obviously does not recognize the end criterion.

Step 4

The %DISASSEMBLE command specifies that 5 lines are to be output in "disassembled" format starting at line 96, i.e. the CLC instruction.

```
/%DA 5 FROM S'96'

SUM+62          CLC   121(6,R2),13A(R2)      D5 05 2121 213A
SUM+68          BC    B'1000',7A(R0,R2)      47 80 207A
SUM+6C          PACK  123(2,R2),121(2,R2)    F2 11 2123 2121
SUM+72          AP    13C(4,R2),123(2,R2)    FA 31 213C 2123
SUM+78          BC    B'1111',28(R0,R2)      47 F0 2028
```

This shows that the length field of the CLC instruction contains '6' instead of '2'. This is why the end criterion is not recognized.

The correct assembler instruction reads:

```
COMP    CLC  INPUT+4(2),ZERO
```

Step 5

This error can be provisionally amended by means of the %SET command. The program is reloaded for this purpose.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0517 MODULE 'SUM' LOADED

/%SET X'D5012121213A' INTO COMP
/%DA 1 FROM COMP

SUM+62          CLC   121(2,R2),13A(R2)      D5 01 2121 213A

/%R
```

%SET changes the memory contents at address COMP. An AID literal with the same length as the CLC instruction and containing the length entry '01' instead of '05' is transferred. The CLC instruction is then checked using %DISASSEMBLE and the program restarted with %RESUME.

```
PLEASE ENTER UP TO 10 2-DIGIT NUMBERS. END: 00
*05
*16
*48
*12
*10
*15
*17
*19
*29
NO MORE THAN 10 NUMBERS CAN BE PROCESSED
SUM:0000171

%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '000000BE (SUM), (CDUMP), EC=90'
%  IDA0N45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP;Y,SYSTEM=SYSTEMDUMP;N=NO)?Y
%  IDA0N53 DUMP BEING PROCESSED. PLEASE HOLD ON
%  IDA0N54 USERDUMP WRITTEN TO FILE 'userid.DUMP.name.2069.00001'
%  IDA0N55 TITLE: 'TSN-2069  UID-userid  AC#-xxxxxxxx  USERDUMP
   PC-0000BE EC=90  VERS-110  DUMP-TIME  11:37:42  94-03-08'
```

Another program error exists, since the user has entered only 9 numbers. A dump for further diagnosis is therefore generated on program termination.

Step 6

The %DUMPFILE command opens the dump file and and assigns it the link name D1. The %BASE command switches the AID work area to the opened dump file. From now on, an address without its own base qualification will always cause the dump file data to be accessed.

```
/%DUMPFILE D1=DUMP.NAME.2069.00001
/%BASE E=D1

/%D INPUT
** D1: DUMP.NAME.2069.00001 ************************************************
INPUT           = 00060000 F2F9   ....29

/%D _R5
_R5             = 0000000B
```

The last number entered in the INPUT field is to be output. The output and log are identical.

As the number of inputs is counted in register 5, it is now queried.

Register 5 contains the value '11', although only 9 numbers were entered. A comparison with the assembly listing shows that register 5 has the initial value '1' and not '0'.

The correct assembler instruction reads: `L  R5,=F'0'`

Step 7

This error can be provisionally amended by means of the %SET command. The program is reloaded for this purpose.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0517 MODULE 'SUM' LOADED

/%BASE
/%SET X'D5012121213A' INTO COMP
/%IN LOOP <%SET #'0' INTO _R5; %REM %.>

/MOD-TEST-OPT DUMP=NO
/%R
```

First, %BASE must be issued to assign the loaded program to the AID work area.

Reloading the program causes the corrections that have been made to be deleted. To ensure an error-free program run, the %SET command from step 5 is issued again here.

%INSERT sets a *test point* to the assembler instruction with the name entry LOOP. This means AID is to execute the following *subcmd* prior to the add instruction.

The %SET command that gives register 5 the initial value '0' is contained in the *subcmd* of %INSERT. This *subcmd* is deleted with %REM after the first run (as no further subcommand has been entered for this *test-point*, the *test-point* is also deleted), and the program is then resumed.

As the TERM macro is defined in the source program with the DUMP=Y operand, a dump is offered every time the program terminates. This can be prevented before the program is started (%RESUME) with the following command: /MODIFY-TEST-OPTIONS DUMP=NO

```
PLEASE ENTER UP TO 10 2-DIGIT NUMBERS. END: 00
*05
*16
*48
*12
*10
*15
*17
*19
*29
*11
NO MORE THAN 10 NUMBERS CAN BE PROCESSED
SUM:0000182

%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '000000BE (SUM), (CDUMP), EC=90'
%  IDA0N47 DUMP PROHIBITED BY /OPTION COMMAND
/
```

After this correction the program executes without errors. The errors can now be definitively eliminated in the source program.

10 Utility routines for structured programming

Structured programming with ASSEMBH is supported by the following utility routines:

COLLIST to create structure lists
COLNAS to create Nassi-Shneiderman diagrams
COLINDA to indent structured source programs
COLNUMA to combine structured and assembler information in a list

Fig. 10-1 shows an overview of the function of COLLIST, COLNAS and COLNUMA with the help of a small example.

The format and meaning of the language elements for structured programming are described in detail in the "ASSEMBH Reference Manual" [1].
Structured programming is not supported by ASSEMBH-BC !

The COLLIST and COLNAS utilities are independent of the programming language, i.e. input to these programs may also consist of dummy code.

COLINDA creates an indented source program from the primary program, so that a clear and transparent listing is generated during assembly.

The COLNUMA utility optionally performs one of the following two functions:
- it extends a source program structure list created by COLLIST by adding to it information from the corresponding assembler listing;
- it enhances the assembler listing of a source program edited by COLINDA in order to highlight the program structure.

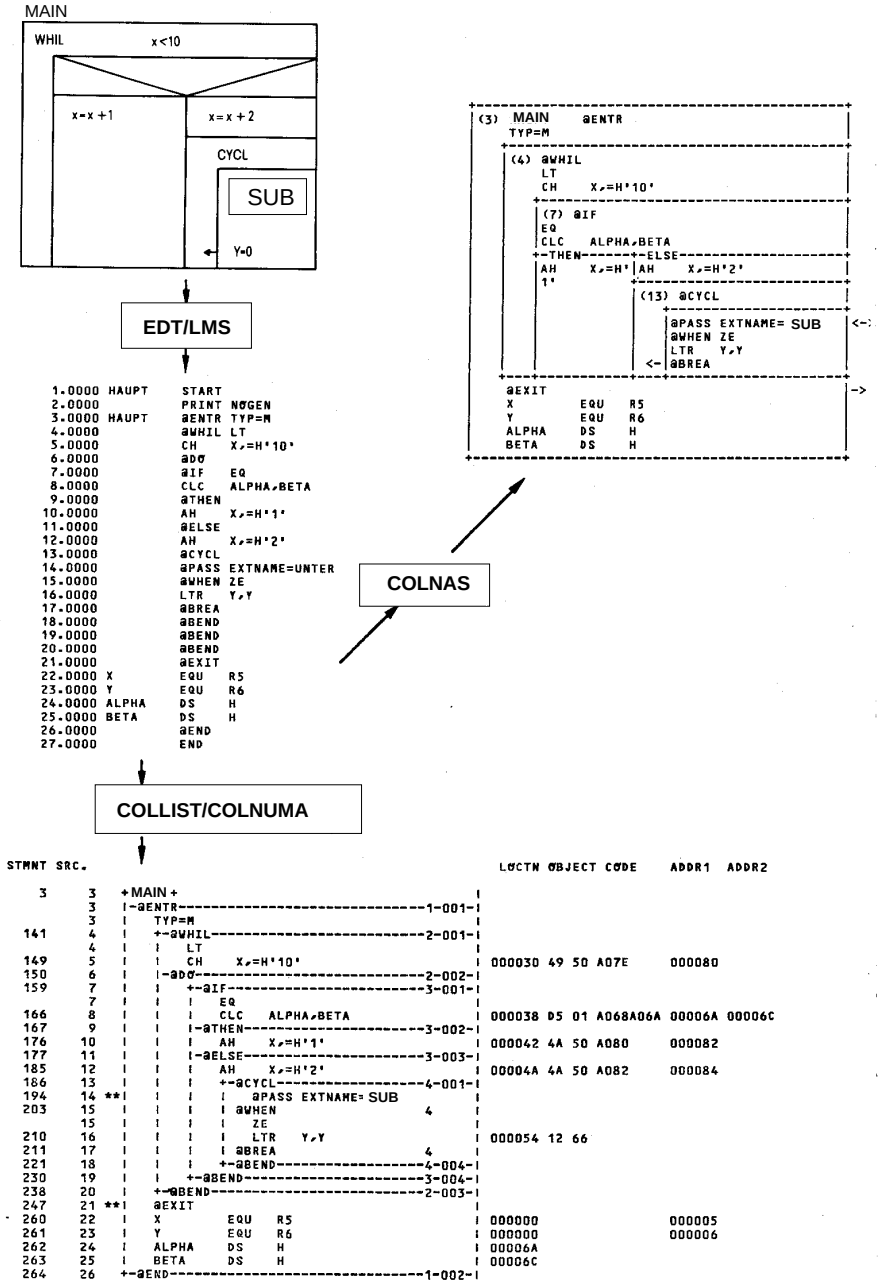


Fig. 10-1: Function of COLNAS and COLNUMA

10.1 Utilities which edit the structured source program

COLLIST, COLNAS, COLINDA

The COLLIST, COLNAS, and COLINDA utilities edit structured programming source programs. All three utilities perform syntax checks and indicate the detected syntax violations at the points where they occur.

10.1.1 COLLIST

The COLLIST utility performs two optional functions:

1. Creation of a structure list to show the nesting of structure blocks,
2. Creation of a procedure list showing the calling hierarchy, and a table of multiply used procedures.

10.1.1.1 Structure list

Block handling

– The structure words as well as the procedure header and procedure end are listed in separate lines, and a horizontal line and a four-digit number are added to the right end of the line. The first digit of the number indicates the nesting level; the remaining three digits form a sequence number within a nesting level and a procedure.

  If the maximum nesting level within a procedure is greater than 9, only sequence numbers starting with 1-000 are generated.

– The structure words within a structure are aligned vertically and linked by a vertical line.

– Subblocks belonging to structure words are indented relative to these words.

– The right margin of the list is terminated by a vertical line.

– The block name is placed in a separate line before its block.

Line handling

- Between the structure word lines, the basic instructions and comments from the source program are taken over and indented relative to the structure words. Leading blanks are removed.
- The basic statements @PASS and @EXIT are identified by means of asterisks in the left margin.
- If there is not enough space for a source line in the listing line, the source line is continued on one or more listing lines. If there is very little space in the listing line, i.e. less than 12 characters, the transfer of source lines is suppressed, and the message "LINE SUPPRESSED" is printed.
- For each listing line, the number of the corresponding source program line is output in the starting columns.
- A line identification number from the source program line (columns 73-80) or the key of the input record can optionally be output beyond the right margin. If no line identifier is desired, COLLIST pushes the vertical line to the extreme right edge.

Error messages

- Error messages and warnings resulting from syntax checks are printed in a line before the invalid keyword.

Page feed

- For each procedure there is a page feed and a header line with the name of the input file, the date, the time of day, and a page number. The procedure number is specified in a second header.

  In the same way, a page feed is generated, and a header line is output before all program sections which are external to procedures.

  If several small procedures are to be printed on one page, they must be separated in the source program by lines containing only an asterisk in column 1. The corresponding number of blank lines will then appear between the procedures in the structure list. Only one header with procedure number is created.
- To meaningfully divide structure lists which occupy more than one printed page, a page feed can be generated by entering a comments statement with *: in columns 1-2.
  This comments statement is not output in the structure list.

If the *: page feed character comes before the beginning of a procedure (@ENTRY), only one page feed takes place. This page feed character merely serves to enhance the clarity of the input file.
The output of the header can be controlled in such cases.

– The user can specify the maximum number of print lines per page.

Example

The following example shows the main features of a COLLIST structure list.

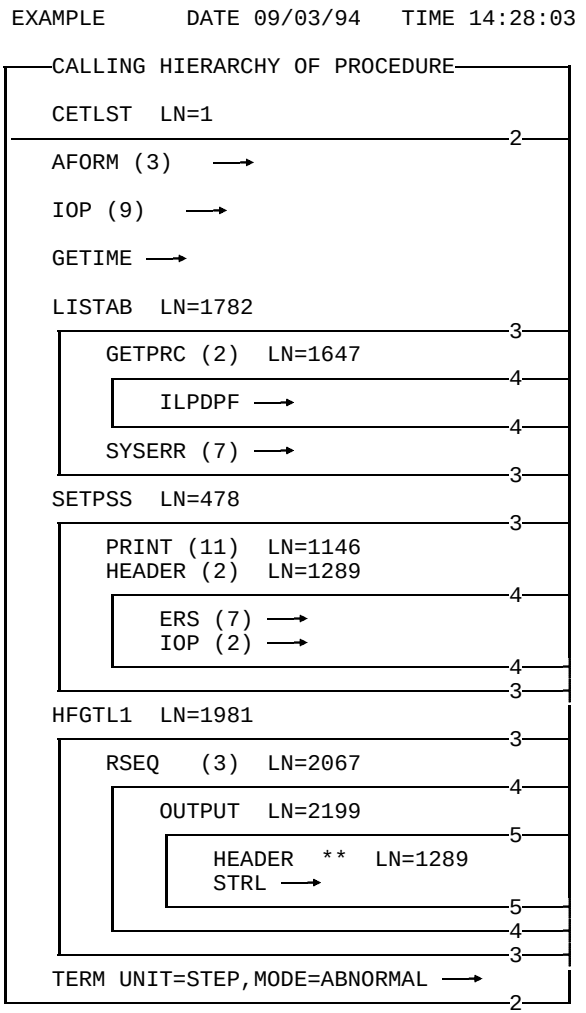
```
Input to COLLIST     Output from COLLIST

NAME  @ENTR TYP=X     1   +NAME+                                  |
      @WHIL CC        1   |-@ENTR----------------------------1-001-|
      .....           1   |   TYP=X                               |
      @DO             2   |   +-@WHIL------------------------2-001-|
      @IF   CC        2   |   |   CC                              |
      .....           3   |   |   .....                           |
      @THEN           4   |   -@DO--------------------------2-002-|
      .....           5   |       +-@IF----------------------3-001-|
      .....           5   |       |   CC                          |
      .....           6   |       |   .....                       |
      @ELSE           7   |       -@THEN--------------------3-002-|
      .....           8   |       |   .....                       |
      .....           9   |       |   .....                       |
      .....          10   |       |   .....                       |
      @CYCL          11   |       -@ELSE--------------------3-003-|
      .....          12   |       |   .....                       |
      @PASS NAME     13   |       |   .....                       |
      .....          14   |       |   .....                       |
      @WHEN CC       15   |       |   +-@CYCL-----------------4-001-|
      .....          16   |       |   |   .....                   |
      @BREA          17 ** |       |   |   @PASS NAME              |
      .....          18   |       |   |   .....                   |
      .....          19   |       |   |  @WHEN               4    |
      .....          19   |       |   |    CC                     |
      @BEND          20   |       |   |   .....                   |
      @BEND          21   |       |   |   @BREA              4    |
      @BEND          22   |       |   |   .....                   |
      @EXIT          23   |       |   |   .....                   |
      @END           24   |       |   |   .....                   |
                     25   |       |   +-@BEND-----------------4-004-|
                     26   |       +-@BEND---------------------3-004-|
                     27   |   +-@BEND------------------------2-003-|
                     28 ** |   @EXIT                               |
                     29   +-@END-----------------------------1-002-|
```

Fig. 10-2: COLLIST structure list

10.1.1.2 Procedure list

A procedure list, which is similar to a structure list, represents the calling hierarchy of a primary source program, where all @ENTR and @PASS statements are interpreted. Following the lists showing the calling hierarchy, there is a final table with information on multiply called procedures.
Further details on the procedure list are provided in the example.

The generation of procedure lists can be controlled so that if a procedure contained in the program is called repeatedly, its substructure is no longer output (identified by **).



```
EXAMPLE      DATE 09/03/94   TIME 14:28:03

MULTIPLE CALLED PROCEDURES
```

PROCEDURE	CALLED FROM
IOP ⟶	CETLST (9) HEADER (2)
HEADER (LN=1289)	SETPSS (2) OUTPUT

Fig. 10-3: Example of a procedure list

Explanatory notes on the example

- The file name of the procedure is EXAMPLE.
- The procedure AFORM is called three times from the procedure CETLST; IOP is called nine times, GETIME once, and so on. Procedures marked with - - → are not contained in the interpreted program.
- The procedure LISTAB, on the other hand, is part of the program EXAMPLE. The corresponding @ENTR statement is in line 1782. LISTAB calls the procedures GETPRC and SYSERR.
- Procedures marked with ** are contained in the interpreted program. The procedures subordinate to them have already appeared earlier on in the procedure list and are not printed out again here (parameter FULPCLST=NO).
- The left column of the final table contains the names of the multiply called procedures IOP and HEADER. If the called procedure is in the analyzed program, the start of the procedure is shown there as well.
- The right column of the table contains the names of the calling procedures and, in parentheses, the number of calls, if there are more than one. The IOP procedure is called nine times by the CETLIST procedure and twice by HEADER.

Notes

- The primary program may contain up to 300 different procedure names (in @ENTR and @PASS statements).
- The primary program may contain a maximum of 1000 @ENTR and @PASS statements (multiple calls to the same procedure from within a procedure are only counted once).

10.1.2 COLNAS

Structured programming procedures are represented as Nassi-Shneiderman diagrams in the lists output by COLNAS.

In particular, related THEN and ELSE subblocks or CASE subblocks are not shown one below the other, as in COLLIST, but next to each other. This emphasizes the flow of control more clearly than simple indentation.
A structure block is basically exited via the terminating horizontal line only, and this is reached via one of any adjacent subblocks.

Although placing subblocks next to one another has definite advantages (clarity, use of a second dimension), there are also some disadvantages, primarily due to the limited number of character positions in a line: in blocks with many levels of nesting there are sometimes so few characters available in the line for each subblock that it is not possible to meaningfully represent all the information in the source program.

If it is possible to show the information from the source program, however, the listing provides a good means of checking the program structure, particularly the nesting of selection structure blocks.

If errors are detected during the syntax check of the structure, the diagram for the procedure in question is suppressed.

10.1.2.1 Format of the list

Indentation amount

The user specifies the "indentation amount" in a control parameter (see INDAMT, section 10.3.5). The default value is 4 characters. The indentation amount determines the indentation of the loop subblocks within repetition structure blocks (@WHIL, @CYCL, @THRU) and that of the sequence subblock (@BEGI).

The indentation amount also plays a part in the decision construct (@IF).

Method of showing decisions

- The condition is placed in a rectangular box below the structure word IF, since diagonal lines cannot be drawn.
- The THEN and ELSE subblocks are indicated at the top edge of the relevant subblock, if there is still room for this in the line.
- If the ELSE subblock is missing, this is shown by an empty strip on the right-hand side, with a width equal to the indentation amount.

The way in which the available partial lines are divided up within subblocks primarily depends on the ratio of the number of lines between @THEN and @ELSE or @ELSE and @BEND. If the ratio is approximately 1 (0.8 - 1.2) or the part line width is relatively small (6 levels of indentation), this width is halved. Otherwise, a part line width is divided, according to the line ratio, into integral multiples of the indentation value. When the line is divided up according to line ratio, a minimum of three indentation values is made available for one subblock.

Information not transferred to the list

- References to source program lines (as output by COLLIST) are omitted. They are only generated for keywords which introduce a structure block.
- The contents of columns 73-80 or the record key are not transferred.
- The output of the contents of structure blocks or subblocks is suppressed if less than two indentation amounts are available for the block in the line.
- When the output of block contents is suppressed, the free space for the block in the diagram is filled with asterisks (*).
- The line numbers of the suppressed lines are output to the listing device (SYSOUT).

Page feed

- For each procedure, a page feed is generated in the list, and also a header line with the name of the input file, date, time of day, and page. The number of the procedure is output in a second header.
  There is also a page feed with output of a header line before all program segments external to procedures.
- The page feed characters *: and * (see section 10.1.1, COLLIST page feed) are taken into account by COLNAS outside procedures only.
- The user can control how many lines are output per page and whether there is a header on each page (even between procedures).

Error messages

- If errors are detected in a procedure when checking the syntax of the structure, no structure diagram is output for the procedure, just a list of structure error messages with the text of the corresponding lines.

Example

The following example shows the main features of a COLNAS listing.

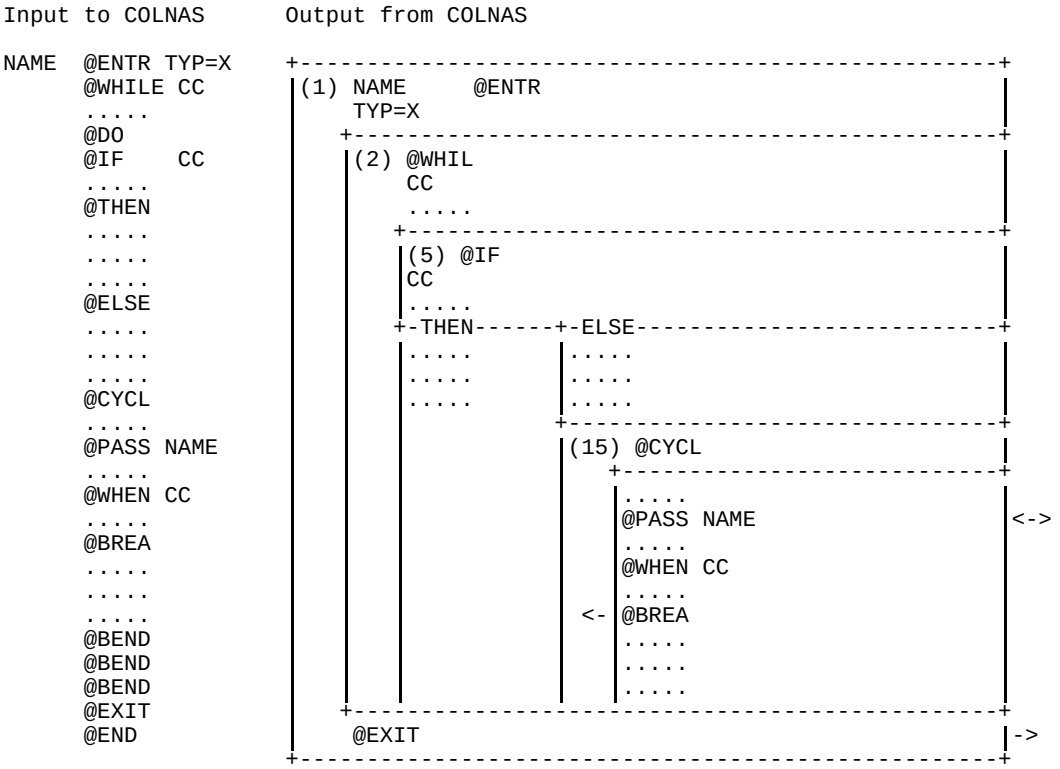


Fig. 10-4: COLNAS listing

10.1.3 COLINDA

The COLINDA utility takes a structured source program and generates from it one that is indented in accordance with structure block nesting.

10.1.3.1 Output from COLINDA

The COLINDA utility changes the format of a structured source program as follows: the operation, operand, and remarks parts of an instruction are indented in accordance with structure block nesting, and lines which begin with a structure word are terminated by means of a horizontal line, at the end of which the nesting level is specified. The name fields of the assembler format are recognized and retained at the left margin.

The generated indented source program serves as input to the assembler and is reflected in its assembler listing.

Format of the output source program and thus the generated assembler listing

- All structure words of a structure block are output, indented to the same character position.
- All structure words are emphasized by means of a line in the remarks section.
- All structure words contain a number at the end of the line, indicating the appropriate nesting level.
- Remarks concerning structure words are output in the next line in the form of a comments line.
- Names of structure blocks and subblocks are separated and placed before the start of the block with

```
DS OH
```

- Subblocks are indented in relation to the structure words to which they belong.
- No indentation is performed before the first and after the last structure block within a procedure (important for data definitions).
- Remarks in assembler instructions are also indented if there is sufficient space in the line; otherwise, they are placed in a separate comments line preceding the instruction.
- The significant part of comments statements (all columns from the first to the last non-blank column in the range 2 to 71) in subblocks is aligned according to the nesting level only if it does not need to extend beyond column 71 as a result of the indentation. If a comments statement in a subblock is to be left unaltered (e.g. boxed comments), this can be achieved by filling in column 2 and column 71.
- Columns 73-80 of the output line are numbered consecutively, so that columns 73-79 represent the sequential number of the input line within the input source program, and column 80 is always set to 0. A unique indication of the original structured source program can thus be found in the assembler listing. This numbering can be suppressed, in which case the line identification from columns 73-80 of the input is taken over.

Example

```
Input to COLINDA    Output from COLINDA

NAME  @ENTR TYP=X   *                       GENERATED BY COLINDA
      @WHIL CC      NAME     @ENTR TYP=X                                  00000010
      .....         *-------------------------------------------------------
      @DO                    @WHIL CC      *----------------------------------2- 00000020
      @IF   CC                      .....                                 00000030
      .....                  @DO           *----------------------------------2- 00000040
      @THEN                         @IF   CC      *--------------------------3- 00000050
      .....                           .....                                 00000060
      .....                         @THEN         *--------------------------3- 00000070
      .....                           .....                                 00000080
      @ELSE                           .....                                 00000090
      .....                           .....                                 00000100
      .....                         @ELSE         *--------------------------3- 00000110
      .....                           .....                                 00000120
      @CYCL                           .....                                 00000130
      .....                           .....                                 00000140
      @PASS NAME                      @CYCL ,       *------------------4- 00000150
      .....                                 .....                           00000160
      @WHEN CC                                @PASS NAME                    00000170
      .....                                   .....                         00000180
      @BREA                           @WHEN CC      *------------------4- 00000190
      .....                                   .....                         00000200
      .....                           @BREA         *------------------4- 00000210
      .....                                   .....                         00000220
      @BEND                                   .....                         00000230
      @BEND                                   .....                         00000240
      @BEND                           @BEND         *--------------------- 00000250
      @EXIT                         @BEND         *--------------------------3- 00000260
      @END                  @BEND         *----------------------------------2- 00000270
                            @EXIT                                         00000280
                  *-------------------------------------------------------
                            @END                                          00000290
```

Fig. 10-5: COLINDA output

10.1.3.2 Structure functions available in the TOM editor

The COLINDA utility function can be used directly from within the TOM editor TOM-TI. The two TOM-TI commands `COLINDA` and `COLA` are provided for this purpose.

COLINDA
: edits a structured assembler program in the TOM-TI work area such that it corresponds exactly to the output of the COLINDA utility.

COLA
: indents the program like COLINDA, but without the following: the line numbers in columns 73-80, the horizontal line to emphasize structure statements, the indication of the nesting level, and "name DS OH" before the start of structure blocks.

10.2 COLNUMA

COLNUMA is the utility that summarizes all information and thus enables the user to debug on the "structured programming level". Its functional scope is determined by the assigned input files.

10.2.1 Extending the structure list

If a structure list of the source program (COLLIST output) is assigned as the input file, it is enhanced with the addition of information from the assembler listing.

Prerequisites:

- The source program must be numbered in ascending order in positions 73-80. This can be achieved by using the following EDT command:

  ```
  @SEQ[UENCE]
  ```

- The assembly must be executed with the following assembler statement:

  ```
  PRINT NOGEN
  ```

- The COLLIST structure list must be generated with the following control parameter:

  ```
  LSTCOL=100
  ```

Input files for COLNUMA thus consist of the following lists, which originate from a source program with numbering in character positions 73-80.

– structure list (COLLIST output)

– assembler listing (assembler output)

The output from COLNUMA consists of a COLLIST structure list with the following insertions:

The corresponding hexadecimal addresses and the generated object code (left part of a line in the assembler listing) are inserted in the right margin of the structure list. The statement number of the assembler listing is inserted in the left margin.

Notes

– Input and output can be controlled via parameters or the link name CLIST, ASMLST or EWCLIST.

– Lines generated via macros are not included in the output list.

– If the source is not assembled with PRINT NOGEN, there will be no location entry in the macro instruction line.

– Assembler messages are not taken over.

10.2.2 Extending the assembler listing of a program edited by COLINDA

If the structure list is missing, i.e. if only an assembler listing is available, COLNUMA will check whether this listing is based on a program edited by COLINDA. If this is the case, COLNUMA will process this listing. If the assembler listing involved is not one of a program edited by COLINDA, the COLNUMA run is terminated with a message.

The following points describe the list that is generated by COLNUMA from the COLINDA assembler listing:

– The "@" characters of statements that introduce structure blocks (@BEGIN, @IF, @CASE, @CAS2, @WHILE, @CYCLE, @THRU) and of the corresponding statements that end them (@BEND) are connected by vertical strokes, without overwriting the "@" character and other non-blank characters (e.g. from name fields) in intervening lines. The parts of a procedure that lie outside structure blocks remain unaffected. COLINDA does not alter these parts either. The drawing of connecting strokes is limited to the area of the procedure in which the nesting level is entered in column 70. If there are structure errors (e.g. a missing @BEND), the strokes end at the horizontal line before the @END at the very latest. The COLINDA assembler listing enhanced by COLNUMA thus depicts the structure as clearly as a structure list created by COLLIST.

– A page feed is generated and a header line is output before each procedure.

– The procedure bodies are already clearly delineated by COLINDA with the horizontal lines after the @ENTR and before the @END.

– The operation code of structure blocks is output as of column 10. This creates a left margin for the name field without touching the vertical strokes separating the structure blocks.

  The example on the following page shows these vertical strokes.

Note

Input and output can be controlled via parameters or with the link names ASMLST and EWCLIST.

Example

```
 SOURCE STATEMENT                                          10:24:17  94-03-09

 BLD      START
 ** ERROR: STRUCTURE  201 IN @ENTRY   BLOCK
 *                       GENERATED BY COLINDA
 ONE      @ENTR TYP=B
*-------------------------------------------------------------------------------

DATA     @DATA CLASS=S, INIT=GLOBALS
         MNOTE 225, BASE MISSING
         MNOTE 225, FAILURES FOUND : ALL SKIPPED

FRAME    DS    OH
         @BEGIN     *-------------------------------------------------------2-
COND      |  DS      OH
          |  @IF     LE *-------------------------------------------------3-
          |  |  CR      R4, R5
          |  @THEN      *-------------------------------------------------3-
LOOP      |     DS      OH
          |     @CYCLE   (R7) *-------------------------------------------4-
          |     |  TRALALA
          |     |  LH      R4, X
OUTIS     |     |  DS      OH
          |     @WHEN ZE   *-------------------------------------------4-
          |     |  LTR     R4, R4
          |     @BREAK     *-------------------------------------------4-
LOOPEND   |     |  DS      OH
          |     @BEND      *-------------------------------------------4-
ENDCOND   |     DS      OH
          |  @BEND      *-------------------------------------------------3-
MVC X,Y   |
@BEND     |
ENDONE   @EXIT
RONE     EQU   R2
** ERROR |STRUCTURE  301 IN @BEGIN   BLOCK
*--------|----------------------------------------------------------------------
          @END
          MNOTE 250, SYNTAX ERROR : @END ON WRONG PLACE : SKIPPED
 DONE     EQU   1
          END
```

Fig. 10-6: Extract of a COLINDA assembler listing enhanced by COLINDA

10.3 Working with the COLLIST, COLNAS and COLINDA utilities

Data flow of structured assembler programs

The flowchart depicted on the opposite page shows the data flow of structured assembler programs.

The structured source program serves as the input file for the COLLIST, COLNAS, and COLINDA utilities. The lines can be numbered consecutively in EDT. The output from COLLIST and COLNAS is referred to as a "list"; the output from COLINDA is an "indented file" containing the source program.

The numbered source program file as well as the indented source program file serve as input to ASSEMBH; also assigned to it is the macro library for structured programming. The numbered source program file is also the input file for the COLLIST utility.

The structure list output by COLLIST and the assembler listing can be processed further by COLNUMA.

ASSEMBH places the module in a module library as specified in the MODULE-LIBRARY option (see section 2.4.2.2). The linkage editor TSOSLNK links the module with the assembler runtime system and generates an executable program ("load module"). This process is described in section 5.6, "Assembling and linking a structured assembler program".

The designations in the flowchart are also used occasionally when describing the operations below, especially when the various input and output files need to be differentiated.

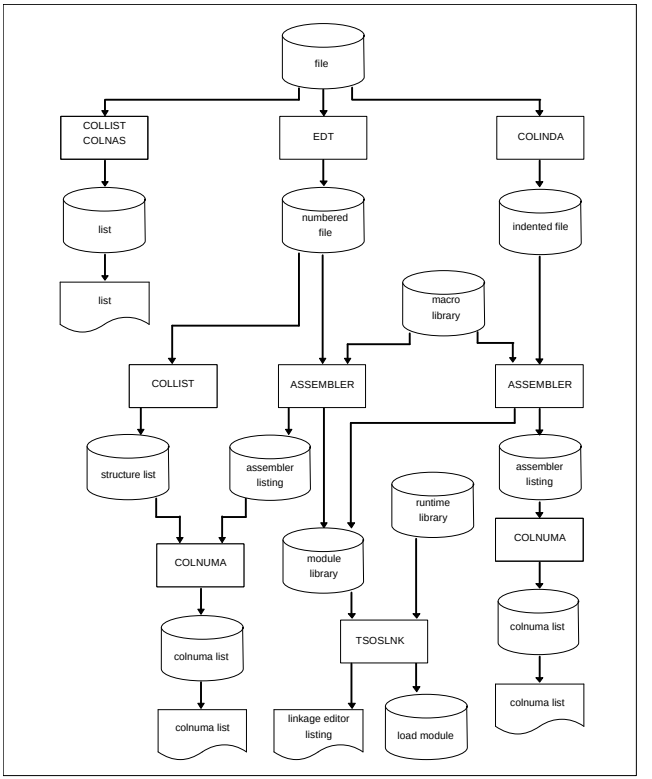


Fig. 10-7: Data flow of structured assembler programs

10.3.1 Input for COLLIST, COLNAS and COLINDA

The structured source programs used as input for the COLLIST, COLNAS and COLINDA utilities can be files of the following kinds:

- SAM files
- ISAM files
- Elements of a PLAM library

Note, however, that the ISAM files must have an 8-byte numeric key. The record length is variable.

Files and libraries are assigned either via the FILE command or via parameters. There are thus three possibilities:

1. With the LINK name for SAM and ISAM files according to the pattern:

```
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=file
```

2. If no FILE command was specified, the assignment can be made via the parameter CINPUT:

```
            ⌈file            ⌉
PAR CINPUT=⎨                 ⎬
            ⌊library(element)⌋
```

3. If the input is obtained from an element of a PLAM library, the assignment can be made via the LINK name SRCLIB and the parameter SRCELEM:

```
/SET-FILE-LINK LINK-NAME=SRCLIB,FILE-NAME=library
PAR SRCELEM=element
```

10.3.2 Output from COLLIST and COLNAS

- COLLIST and COLNAS lists are output to a SAM file that is identified with the suffix "CLIST" by default. Thus, if "file" is the structured source program file, the output is written to

```
file.CLIST
```

- The output file can be defined by using the FILE command and the LINK name CLIST as indicated below:

```
/SET-FILE-LINK LINK-NAME=clist,FILE-NAME=output-file
```

- The assignment of the output file can likewise be made using parameters:

```
PAR CLIST=output-file
```

– Output to a PLAM library is achieved via the parameter assignment:

```
PAR CLIST=library(element)
```

– The output lists can be printed with the command:

```
/PRINT-FILE FILE-NAME=output-file,DELETE-FILE=YES,-
LAYOUT-CONTROL=PAR(FORM-NAME=format, CHARACTER-SETS=chars,-
CONTROL-CHARACTERS=EBCDIC)
```

10.3.3 Output from COLINDA

– The indented source program generated by COLINDA is output by default to a SAM file that is identified with the suffix COUT. Thus, if "file" is the name of the structured source program, the output file will be:

```
file.COUT
```

– The output file can also be assigned by means of the FILE command and the LINK name:

```
/SET-FILE-LINK LINK-NAME=COUTPUT,FILE-NAME=output-file
```

– The output file can likewise be assigned via parameters:

```
PAR COUTPUT=output-file
```

– If the output is to be placed in an element of a PLAM library, the assignment is made via a parameter:

```
PAR COUTPUT=library(element)
```

– The output file serves as input for the subsequent assembler run. This produces an assembler listing in indented form. It is generally not advisable to print out the created file, since the information obtained from it can be more clearly represented by COLLIST or after processing it with COLNUMA. The created file can, however, be used as fresh input for COLINDA (possibly after corrections).

Summary

Based on the explanations in the preceding sections, the commands that are available for use with the COLLIST, COLNAS, and COLINDA utilities can be summarized as follows:

Input and output files are BS2000 files

```
/LOGON ...
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=file

[/SET-FILE-LINK LINK-NAME=COUTPUT,FILE-NAME=indented-file]            (1)
[/SET-FILE-LINK LINK-NAME=CLIST,FILE-NAME=list]                       (2)
[/ASSIGN-SYSDTA TO-FILE=parameter]                                    (3)

            ⌈ASSEMBH.COLLIST⌉
/START-PROG {ASSEMBH.COLNAS  }
            ⌊ASSEMBH.COLINDA⌋
.
.
.
Parameter input
.
.
.
[/ASSIGN-SYSDTA=*PRIMARY]                                             (3)

/PRINT-FILE FILE-NAME=output-file,DELETE-FILE=YES,-
LAYOUT-CONTROL=PAR(FORM-NAME=format, CHARACTER-SETS=chars,-
CONTROL-CHARACTERS=EBCDIC)
/LOGOFF
```

(1) for COLINDA
(2) for COLLIST and COLNAS
(3) optional if parameters are entered via SYSDTA (see section 10.3.4 for details)

Input and output files are elements of a PLAM library

```
/LOGON ...
[/SET-FILE-LINK LINK-NAME=SRCLIB,FILE-NAME=library]
[/ASSIGN-SYSDTA TO-FILE=parameter]                                    (3)
.
.
.
            ⌈ASSEMBH.COLLIST⌉
/START-PROG {ASSEMBH.COLNAS  }
            ⌊ASSEMBH.COLINDA⌋
.
.
.
[PAR SRCELEM=input-element]
[PAR CINPUT=input-library(element)]
[PAR CLIST=output-library(element)]                                   (1)
[PAR COUTPUT=output-library(element)]                                 (2)
.
.
Additional parameter input
.
.
[/ASSIGN-SYSDTA=*PRIMARY]                                             (3)

/PRINT-FILE FILE-NAME=output-file,DELETE-FILE=YES,-                   (4)
LAYOUT-CONTROL=PAR(FORM-NAME=format, CHARACTER-SETS=chars,-
CONTROL-CHARACTERS=EBCDIC)
/LOGOFF
```

(1) for COLLIST and COLNAS

(2) for COLINDA

(3) optional if parameters are entered via SYSDTA (see section 10.3.4 for details)

(4) If the output is placed in a library, the element must be made available in a file before printing

10.3.4 Control of COLLIST, COLNAS and COLINDA

The COLLIST, COLNAS, and COLINDA utilities provide the user with the following control options, all of which can be specified via parameters:

- assignment of files, PLAM or LMS elements for input (CINPUT).
- assignment of PLAM or LMS elements for input (SRCELEM).
- assignment of files or PLAM elements for output (COUTPUT).
- assignment of files or PLAM elements for output of listings (CLIST).
- flexible internal memory management to suit the maximum procedure size (PROCSIZE).
- replacement of the default syntax characters

  ```
  @  :  *  ,
  ```

  by others (DELIM).
- definition of the indentation amount (INDAMT).

For COLLIST and COLNAS only

- The output of target language statements can be fully or partially suppressed (STATEMENT).
- The output of comments lines is suppressed. Only the target language statements in structure blocks are output (COMMENT).
- If target language statements as well as comments lines are suppressed, only the structure statements are output in their order and nesting.
- The output of structure blocks can be suppressed as of a specified nesting level (LEVLIM).
- Line length; narrow and wide pages can be generated, e.g. DIN formats (LSTCOL).
- The maximum length of input records can be defined (RECLEN).
- Variable line identifier (LINEID).
- Page feed control (LINELIM).
- Control over output of page header (HEADLINE).

For COLLIST only

– Request for structure and/or procedure lists (LIST).

– Repeated output of substructures in the procedure list (FULPCLST).

For COLNAS only

– The output of structure blocks can be suppressed up to a specified nesting level (LEVBEG).

For COLINDA only

– Consecutive numbering of output lines (RENUM).

Input and output from and to files can also be controlled by using link names (see sections 10.3.1 and 10.3.3).

10.3.5 Parameters

The utilities are controlled via parameters; however, it is only necessary to specify them in exceptional cases. Normally, the default values for the parameters apply.

Parameters may be entered either in interactive mode or in batch mode. In interactive mode, the user controls the input of parameters by responding to the terminal prompt as follows:

```
PARAMS? (STANDARD/SYSDTA/DIALOG)
```

Meaning of each response:

STA[NDARD] No further parameter input. Default values are assumed for all parameters.

[SYS[DTA]] In interactive mode parameter statements in the form

$\texttt{PAR param}_1\texttt{=value}_1\texttt{,param}_2\texttt{=value}_2\texttt{...}$

are read in from SYSDTA until the END statement. Instead of responding with SYS[DTA], parameter statements may also be entered directly.

If default values are to be used for all parameters, it is sufficient to enter END.

The SYS[DTA] response is intended for cases when the parameters are in a file, and were assigned with /ASSIGN-SYSDTA=parameter before the program call.

DIA[LOG] Individual parameters are interactively requested via the terminal (with the WRTRD macro) with the query:

$param_i$? (brief description)

Response:

$value_i$

Besides the desired values, STANDARD (default values for the queried parameter) and END (default value for the queried parameter and for all that follow) may also be specified as responses to "param?". If END is not entered as a response, all parameters will be queried.

In batch mode (ENTER tasks), parameter statements in the form

PAR $param_1$=$value_1$,$param_2$=$value_2$,.....

terminated by

END

must be supplied via SYSDTA, i.e. usually immediately after the /START-PROG command. If default values are to be used, only an END statement must be specified.

Description of parameters

The following table shows which parameters are applicable to the individual utilities (or functions). The meanings of the parameters are explained thereafter.

Parameters	Value range	COLINDA	COLLIST STR	COLLIST PRC	COLNAS
CINPUT={file / library(element)}		X	X	X	X
CLIST={file / library(element)}			X	X	X
COMMENT={YES / NO}			X		X
COUTPUT={file / library(element)}		X			
DELIM={'abcd' / '@:*,'}		X	X	X	X
FULPCLST={YES / NO}				X	
HEADLINE={PROC / PAGE}			X		X
INDAMT=n	1≤n≤8	X(n=3)	X(n=4)X		X(n=4)
LEVBEG={n / 1}	1≤n≤20				X
LEVLIM={n / 20}	1≤n≤20		X		X

Parameters	Value range	COLINDA	COLLIST STR	COLLIST PRC	COLNAS
LINELIM={n / 64}	0≤n≤144		X	X	X
LINEID={YES / NO / KEY}			X	X	X
LIST={STR / PRC / ALL}			X	X	
LSTCOL={n / 100}	52≤n≤240		X	X	X
PROCSIZE={n / 250}	10≤n≤4000	X	X	X	X
RECLEN={n / 80}	80≤n≤255		X	X	X
RENUM={YES / NO}		X			
SRCELEM=element		X	X	X	X
STATMENT={YES / NO / CON}			X		X

Alternative values for parameters are enclosed within braces. Default values are underlined.

Meaning of each parameter

CINPUT — The CINPUT parameter is used to assign files or elements of a PLAM or LMS library after calling a utility.

CLIST — File or PLAM library element in which output from COLLIST and COLNAS is to be placed.

COMMENT — When COMMENT=YES, remarks are included in the output.

COUTPUT — File or PLAM library element in which output from COLINDA is to be placed.

DELIM — 4 printing characters to replace the delimiters in the order @ : * ,

where the characters have the following meanings:

@ Prefix for structure words.

: Form feed control for COLLIST in combination with the comment character.

* Comment identifier.

, Separator for parameters.

The characters "@", "*", and "," may only be changed in dummy code, not in the structured assembler program.

Even if all 4 delimiters are not being replaced, all 4 characters must be specified.

The 4 characters must be enclosed in single quotes, unless the value for DELIM is queried interactively.

FULPCLST — Controls the procedure list output of COLLIST:

=YES — Full-size procedure list (default).

=NO — In the case of repeated calls to a procedure contained in the program, the output of the procedure substructure is suppressed (identified by **).

HEADLINE — Controls the output of the page header.

=PROC — The page header appears on the first page and on every page on which a new procedure begins.

=PAGE — The header appears on every page, regardless of how the page feed was triggered.

A page feed is effected when

- a new procedure or a control section external to procedures begins,
- a procedure or a control section has more lines than are predefined in the LINELIM parameter,
- the user forces a page feed with *:

INDAMT — Numeric value between 1 and 8 for the indentation amount.

LEVBEG — Numeric value between 1 and 20, specifying the first nesting level as of which printing is to begin. The LEVBEG parameter only applies to COLNAS.

LEVLIM — Numeric value between 1 and 20 (default value 20) which defines the last nesting level to be printed.

LINEID

=YES — Columns 73-80 of the input constitute the line identifier. COLLIST enters this identifier at the right margin of the list. No line identifier is output by COLNAS.

The value of the RECLEN parameter must be 80; otherwise, a warning will be issued, and LINEID will be set to NO by COLLIST and COLNAS.

=NO — Columns 73-80 of the input are not interpreted as a line identifier, but as a part of the program text. The COLLIST and COLNAS utilities process the entire line.

The vertical line at the right edge in the structure list is pushed to the extreme right by COLLIST, so that more space is available for text.

=KEY — The key of the relevant input record is treated as the line identifier. In the case of ISAM files, this is the ISAM key (only keypos=5 and keylen=8 are allowed). In the case of SAM files, the key consists of the first 8 characters of the record (can be generated in EDT with "@WRITE'input'KEY"). The key is checked for numeric contents. If it is not numeric, an error message is issued.

COLLIST and COLNAS process the entire contents of the line (after the key). COLLIST prints the key in the structure list next to the vertical stroke at the right margin.

LINELIM

LINELIM=0 is the lower limit.
LINELIM=64 is the default value.
LINELIM=144 is the upper limit for the value of nn.

Defines the number of lines after which a new page is to begin in the structure and procedure list (output of COLLIST) and in the structogram (output of COLNAS), provided a page feed was not forced earlier by the start or end of a procedure or, for COLLIST, by the entry of "*:" in columns 1-2.

The automatic page feed after nn lines is suppressed if LINELIM is assigned the value 0.

LIST

Controls the list function of COLLIST:

=STR

The structure list is created (default).

=PRC

The procedure list is created (if there are no structure errors).

=ALL

Both lists are created (if structure errors exist: only the structure list).

LSTCOL

Numeric value between 52 and 240 (default value 100).
Last print position in the line.

A value above 132 should only be specified if an appropriate printer is available.

PROCSIZE

Numeric value between 10 and 4000 (default value 250). The parameter determines the size of the memory area for internal listings.
PROCSIZE is rounded up in steps of 200, and one page (4 KB) of virtual memory is requested for each multiple of 200.
The determining factor for the size of the memory area required for the internal listings is the number of structure statements in a procedure. The value X of the PROCSIZE parameter can be roughly estimated according to the formula:

```
X = 3 * S
```

where S is the number of structure statements.

With structograms, the value of PROCSIZE is therefore approximately equal to the number of source program lines in the largest procedure (@ENTR to @END).

The procedure with the highest value for X determines the value of the PROCSIZE parameter. The default value of 250 should therefore suffice in most cases, especially since memory is always requested in units of 4 Kbytes. If there is not enough memory, COLLIST issues a message recommending a new value for PROCSIZE.

RECLEN — RECLEN=80 is the lower limit and default value.
RECLEN=255 is the upper limit.

The RECLEN parameter can be used to control the maximum permissible length for input records, so that records with more than 80 characters can be used when working with dummy code.

If the value of RECLEN is greater than 80, the contents of columns 73-80 will not be interpreted as the line identifier but as part of the program text.
In this case the LINEID parameter may only have the values NO and KEY.

RENUM — Numbering of COLINDA output in columns 73-80.

=YES — Sequential numbering of lines in steps of 10.

=NO — The contents of columns 73-80 from the input line are carried over to the output line.

SRCELEM — This parameter is used to define the library element for the input after the library has been assigned with the link name SCRLIB.

STATMENT — Transfer of target language statements to the output.
With STATMENT=CON, only the conditions will be output, i.e. the target language texts between @IF and @THEN, @WHILE and @DO, @THRU and @DO, @WHEN and @BREAK, as well as @CASE2 and the first @OF.

Examples

Interactive mode

Working with default values (for all parameters):

```
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=TEST-PROGRAMM
/START-PROG ASSEMBH.COLLIST
%  BLS0500 PROGRAM 'COLLIST', VERSION '41B11' OF '1991-05-16' LOADED
%  BLS0551 COPYRIGHT (C) SNI 1991. ALL RIGHTS RESERVED
COLLIST VERSION 41B11 - 01.12.91 STARTED
PARAMS?(STANDARD/SYSDTA/DIALOG)
*STA
COLLIST COMPLETED
```

Input of current parameters from a file read via SYSDTA:

```
/START-PROG $EDT
%  BLS0500 PROGRAM 'EDT', VERSION '16.4A' OF '1992-06-24' LOADED
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG 1992. ALL
   RIGHTS RESERVED
@EDT
   1.
PAR INDAMT=6
   2.
PAR RENUM=NO
   3.
END
   4.
@W'TEST.PARAMETER'
   4.
@H
EDT NORMAL END
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=TEST-PROGRAMM
/ASSIGN-SYDTA TO-FILE=TEST.PARAMETER
/START-PROG ASSEMBH.COLINDA
%  BLS0500 PROGRAM 'COLINDA', VERSION '22F11' OF '1991-02-11' LOADED
%  BLS0551 COPYRIGHT (C) SNI 1991. ALL RIGHTS RESERVED
COLINDA VERSION 2.2F11 - 01.12.91 STARTED
PARAMS?(STANDARD/SYSDTA/DIALOG)
COLINDA COMPLETED
/ASSIGN-SYSDTA TO-FILE=*PRIMARY
```

Input of current parameters in interactive mode:

```
/START-PROG ASSEMBH.COLLIST
%  BLS0500 PROGRAM 'COLLIST', VERSION '41B11' OF '1991-05-16' LOADED
%  BLS0551 COPYRIGHT (C) SNI 1991. ALL RIGHTS RESERVED
COLLIST VERSION 41B11 - 01.12.91 STARTED
PARAMS?(STANDARD/SYSDTA/DIALOG)
*DIA
CINPUT?(NAME OF COLUMBUS-INPUT)
TEST.PROGRAMM
COMMENT?(YES/NO)
YES
STATMENT?(YES/NO/CON)
CON
PROCSIZE?(MAX SIZE OF PROCEDURES IN NO OF STMTS)
STA
LSTCOL?(LAST COLUMN IN LISTING)
60
LINELIM?(LIMIT OF LINES PER PAGE)
END
COLLIST COMPLETED
```

Input of current parameters via PAR statements:

```
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=TEST-PROGRAMM
/START-PROG ASSEMBH.COLNAS
%  BLS0500 PROGRAM 'COLNAS', VERSION '41B11' OF '1991-05-17' LOADED
%  BLS0551 COPYRIGHT (C) SNI 1991. ALL RIGHTS RESERVED
COLNAS VERSION 41B11 - 01.12.91 STARTED
PARAMS?(STANDARD/SYSDTA/DIALOG)
*PAR LSTCOL=80,INDAMT=6
*PAR STATMENT=CON
*END
COLNAS COMPLETED
```

Batch mode

The following options are available to the user in batch mode (ENTER task):

– Working with default values:
  This is done by entering the "END" statement immediately after the utility is loaded and started with the EXEC command.

– Assignment of current parameter values:
  To do this, the user must supply the current values via SYSDTA, which normally means immediately after the EXEC command. These values are specified in the form of PAR statements, and must be terminated with the statement "END" (the format is described earlier in this section).

The following example shows the setup of an ENTER file where current values are assigned to specific parameters:

```
/LOGON ...
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=ERB.CON
/START-PROG ASSEMBH.COLLIST
PAR LSTCOL=80,INDAMT=6
PAR STATMENT=CON
END
/RELEASE CINPUT
/LOGOFF
```

10.4 Working with the COLNUMA utility

The function to be performed by COLNUMA is determined by the input files. COLNUMA is started with the command:

```
/START-PROG ASSEMBH.COLNUMA
```

The files and PLAM library elements for input and output are assigned via link names or parameters.

10.4.1 Extending the structure list

Input

A structure list created by COLLIST and an assembler listing generated by ASSEMBH must be provided as the input.

The structure list can be assigned in the following ways:

Link name:

```
SET-FILE-LINK LINK-NAME=CLIST,FILE-NAME=file
```

Parameters:

```
          ⌈file            ⌉
PAR CLIST=⎨                ⎬
          ⌊library(element)⌋
```

The following methods can be used to assign the assembler listing:

Link name:

```
SET-FILE-LINK LINK-NAME=ASMLST,FILE-NAME=file
```

Parameters:

```
            ⌈file            ⌉
PAR ASMLST=⎨                 ⎬
            ⌊library(element)⌋
```

Notes

– The input assembler listing must be one that was created with the option:
  LISTING=PAR(LAYOUT=PAR(FORMAT=F-ASSEMB-COMPATIBLE))

  If the input assembler listing is entered from a library, this library element must be of type P.

– The structured assembler source program that is the source of the two inputs to COLNUMA (i.e. the structure list and assembler listing) must be numbered in columns 73-80 by means of the program $EDT before being processed by COLLIST or the assembler. These numbers are carried over into the structure list as well as the assembler listing and establish the reference between the two listings.

– The COLLIST structure list must be a list that was created with the parameter LSTCOL=100 (default value).

Output

The extended structure list can be output to a SAM file or an element of a PLAM library by means of the following assignment:

Link name:

```
SET-FILE-LINK LINK-NAME=EWCLIST,FILE-NAME=output-file
```

Parameters:

```
                ⌈file            ⌉
PAR EWCLIST=    {                }
                ⌊library(element)⌋
```

The input of parameters is terminated with END.

Summary

The following commands can be used to create the input files for COLNUMA and to run the COLNUMA utility:

```
/LOGON
/START-PROG $EDT
@READ'file'
@SEQ
@W'numbered-file'
@H
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=numbered-file
/SET-FILE-LINK LINK-NAME=CLIST,FILE-NAME=structure-list
/START-PROG ASSEMBH.COLLIST
STA
/DELETE-SYSTEM-FILE OMF
/START-PROG $ASSEMBH
//COMPILE SOURCE=numbered-file,LISTING=PAR(LAYOUT=PAR(FORMAT=F-ASSEMB-COMPATIBLE
//        PAR(OUTPUT=assembler-listing)),-
//        MACRO-LIBRARY=macro-library
//END
/SET-FILE-LINK LINK-NAME=CLIST,FILE-NAME=structure-list
/SET-FILE-LINK LINK-NAME=ASMLST,FILE-NAME=assembler-listing
/SET-FILE-LINK LINK-NAME=EWCLIST,FILE-NAME=colnuma-list
/START-PROG ASSEMBH.COLNUMA
[PAR CLIST=structure-list]
[PAR ASMLST=assembler-listing]
[PAR EWCLIST=colnuma-list]
[END]
/DELETE-FILE numbered-file
/DELETE-FILE assembler-listing
/DELETE-FILE structure-list
/PRINT-FILE FILE-NAME=output-file,DELETE-FILE=YES,-
LAYOUT-CONTROL=PAR(FORM-NAME=format, CHARACTER-SETS=chars,-
CONTROL-CHARACTERS=EBCDIC)
```

10.4.2 Enhancing the assembler listing of a program edited by COLINDA

Input

The structured assembler source program must first be edited by COLNUMA (see sections 10.3.1 and 10.3.3), and the edited program must then be assembled. The assembler listing that is generated by ASSEMBH and output to SYSLST serves as the input to COLNUMA.
The input file can be assigned via a link name or parameters:

Link name:

```
SET-FILE-LINK LINK-NAME=ASMLST,FILE-NAME=file
```

Parameters:

```
                ⌈file            ⌉
PAR ASMLST=     |                |
                ⌊library(element)⌋
```

Notes

– The input assembler listing must have been created with the following option:
  LISTING=PAR(LAYOUT=PAR(FORMAT=F-ASSEMB-COMPATIBLE))

  If the input assembler listing is input from a library, then this must be a type P library element.

Output

The assembler listing is enhanced with the addition of vertical strokes and page headers. The assignment of a file or a PLAM library element for output can be made either via a link name or via parameters:

Link name:

```
/SET-FILE-LINK LINK-NAME=EWCLIST,FILE-NAME=output-file
```

Parameters:

```
                ⌈file            ⌉
PAR EWCLIST=    |                |
                ⌊library(element)⌋
```

The input of paramters is terminated with END.

Summary

The following commands can be used to prepare the input and to run COLNUMA:

```
/LOGON
/SET-FILE-LINK LINK-NAME=CINPUT,FILE-NAME=file
/SET-FILE-LINK LINK-NAME=COUTPUT,FILE-NAME=indented-file
/START-PROG ASSEMBH.COLINDA
*PAR param=value,...
*END
/DELETE-SYS-FILE OMF
/START-PROG $ASSEMBH
//COMPILE SOURCE=indented-file,LISTING=PAR(LAYOUT=PAR(FORMAT=F-ASSEMB-COMPATIBLE
//        PAR(OUTPUT=assembler-listing)),-
//        MACRO-LIBRARY=macro-library
//END
[/SET-FILE-LINK LINK-NAME=ASMLST,FILE-NAME=assembler-listing]
[/SET-FILE-LINK LINK-NAME=EWCLIST,FILE-NAME=colnuma-list]
/START-PROG ASSEMBH.COLNUMA
[PAR ASMLST=assembler-listing]
[PAR EWCLIST=indented-file]
[END]
/DELETE-FILE indented-file
/DELETE-FILE assembler-listing
/PRINT-FILE FILE-NAME=colnuma-list,DELETE-FILE=YES,-
LAYOUT-CONTROL=PAR(FORM-NAME=format, CHARACTER-SETS=chars)
```

10.4.3 Parameters

The following table shows which parameters are applicable to COLNUMA. The meanings of the parameters are explained thereafter.

Parameters
ASMLST=⎧file⎫ ⎩library(element)⎭
EWCLST=⎧file⎫ ⎩library(element)⎭
CLIST=⎧file⎫ ⎩library(element)⎭

ASMLST — File or element of a PLAM library which contains the assembler listing.

EWCLST — File or element of a PLAM library which is to take the output from COLNUMA.

CLIST — File or element of a PLAM library which contains the structure list created by COLLIST.

10.5 Messages from the utilities

There are three classes of error messages within the scope of structured programming. These are:

– operator error messages and system messages

– error messages relating to structured programming syntax violations, and

– error messages that may appear during execution of a structured program.

10.5.1 Operator error messages and system messages

Operator error messages and system messages are output to SYSOUT.

COLLIST, COLNAS and COLINDA

The table below lists the error numbers for the individual components, to the extent that they are relevant for structured programming.

The following codes are used:

PR Parameter handling
IO Primary input and output
RQ Memory request
IL Intermediate language
PL Procedure list
SH String handling

Message nnn iiii cn xxx	Meaning	Effect
001 - RQ	The memory area defined by the PROCSIZE parameter is not available. The program is terminated.	Termination

Message nnn iiii cn xxx	Meaning	Effect
003 - IL	The memory area estimated by the PROCSIZE parameter is not sufficient to process a procedure. The program is terminated.	Termination
005 - PR	System error affecting RDATA macro.	Termination
006 - PR	System error affecting WRTRD macro.	
007 - PR	System error affecting WROUT macro.	
011 - PR parameter statement	Error in parameter statement.	The program assumes default values for the parameters.
014 - PR parameter statement	Error in parameter statement RECLEN > 80 and LINEID = YES	Parameter LINEID is set to NO by the COLLIST and COLNAS utility routines.
001 u) IO y) name of library	Error on opening the library	Termination
003 u) IO y) name of library	Error on closing the library	
004 u) IO y) name of element and library	Error on reading a record from a library element	

Message nnn iiii cn xxx	Meaning	Effect
005 u) IO y) name of element	Error on writing a record to a library element	
007 - IO	System error affecting WROUT macro.	Termination
008 - IO	Invalid key in the input record.	
009 x1) IO	DMS error.	
010 x1) IO	DMS error. Error in FILE macro.	
014 - IO	Invalid file name generated for the output file.	
023 u) IO y) name of library and element	Error on opening a library element.	Termination
024 u) IO y) name of element	Error on closing a library element.	
015 - PR parameter line	Wrong format for file or library name.	Termination
016 - PR parameter line	Wrong format for element name.	
017 - PR	CINPUT and SRCELEM are specified.	The most recently made entry applies.

Message nnn iiii cn xxx	Meaning	Effect
016 - IO Input record	Input record longer that preset in RECLEN parameter.	The record is truncated.
020 - PL	Too many procedure names (more than 300).	Reduce size of primary program.
021 - PL	Too many @ENTR and @PASS statements.	Reduce size of primary program.
022 - PL	Recursive procedure call and parameter FULPCLST=YES (A procedure list is generated with FULPCLST=NO).	
023 - PL	At least one @PASS statement is outside a procedure.	Check the primary program, since the procedure list(s) may contain errors.
024 - PL	At least 2 @ENTR statements with the same procedure name have occurred. Identifier in the procedure list: <proc.-name> LN= **nn where nn = line number of 1st @ENTR statement.	Check the primary program, since the procedure list(s) may contain errors.

Message nnn iiii cn xxx	Meaning	Effect
025 - PL	No procedure on "highest" level present (recursive call). A procedure list that begins with the first @ENTR statement is generated.	
002 - SH	Error in string handling.	Inform customer service department.
004 - SH	Insufficient string memory.	Inform customer service department.

Explanation

```
         ERROR: nnn [iiii] COMPONENT: cn
        [PROGRAM IS TERMINATED]
        [xxxxxxxxxxx.........x]

nnn     Error number
iiii    Additional error designation of the system
cn      Abbreviated component name
xxx     Supplementary text line

x1)  DMS code (see "BS2000 System Messages, Reference Manual")
u)   PLAM/ILAM return code
y)   Library identifier
        P   PLAM
        M   MLU, LMS (Version 1.0)
        C   COBLUR
        F   FMS
        U   Undefined
```

Note

Program termination messages which also indicate whether structure errors have occurred in the source are output to SYSOUT like the operator error messages and system messages.

COLNUMA

Message	Meaning
NO CLIST PROGRAM IS TERMINATED	No structure list was assigned, and the program was not edited with COLINDA.
NO ASMLST PROGRAM IS TERMINATED	No assembler listing was assigned.
WRONG NUMBERS IN COL. 73-80 INPUT FILE OR COLLIST PARAM LSTCOL NOT = 100 (STANDARD)	The input file is incorrectly numbered or the structure list is too wide or too narrow.
END OF ASSEMBLER LISTING	The structure list and assembler listing are not of the same program.

10.5.2 Syntax error messages

Handling of messages for the individual utility routines

- COLLIST
  When COLLIST is used, messages concerning structure errors are inserted into the list at the places where they occur.
- COLNAS
  In the case of structure errors within a procedure, COLNAS outputs no structure diagram for this procedure, but merely an error list. Warnings are output only if there is also a structure error in the current procedure.
- COLINDA
  The program COLINDA outputs structure errors and warnings in the form of comment lines in the generated indented structured program at the positions where they occur.

Format of the syntax error messages

```
⎡W⎤
⎣E⎦  aabb  zz...zz
```

W	Warning
E	Error
aa	Max. two-digit number (leading zero suppressed), representing a structural status in which only certain keywords are permitted (see section 10.5.3 for meanings).
bb	Two-digit number representing an invalid or missing keyword: 01 A terminating @BEND or @END is missing in the status defined by aa (hierarchy level not properly closed). 09-17 An invalid keyword has occurred in the status defined by aa.
zz....zz	Indication of procedure error or structure block error.

Warnings

Waabb	Status of the hierarchy level	Expected keywords on the same hierarchy level

Example

W614	Status after @CYCL @CYCL is followed by a @BEND without an intervening @WHEN-@BREAK (termination conditions missing): count loop or continuous loop	@WHEN

10.5.3 Meaning of aabb in syntax error messages

aa..	Status of hierarchy level	Permissible keywords
1..	Initial status or status between @BEND and start of next structure block	@BEGIN, @IF, @WHILE, @CASE, @CYCLE, @EXIT @PASS
2..	Status after @ENTR	@END
3..	Status after @BEGIN	@BEND, @WHEN
4..	Status after @IF	@THEN
5..	Status after @WHILE	@DO
6..	Status after @CYCLE	@WHEN
7..	Status after @CASE	@OF
8..	Status after @THRU	@DO
9..	Status after @ON	@DO
10..	Status after @THEN	@ELSE, @BEND, @WHEN
11..	Status after @ELSE	@BEND, @WHEN
12..	Status after @DO	@BEND, @WHEN
13..	Status after @WHEN	@BREAK
14..	Status after @BREAK	@BEND, @WHEN
15..	Status after @OF	@OF, @OFREST, @BEND @WHEN
16..	Status after @OFREST	@BEND, @WHEN

..bb	Keyword missing or invalid
..01	Terminating @BEND or @END missing
	Invalid keywords (09-17)
..09	@THEN
..10	@ELSE
..11	@OF
..12	@OFREST
..13	@END
..14	@BEND
..15	@WHEN
..16	@BREAK
..17	@DO

10.6 Support for monitoring job variables

If a utility is called with

/START-PROG ASSEMBH.COL... ,MONJV=jvname

where jvname is the name of a job variable defined by the user, program execution can be monitored, since the utilities place a return code in bytes 4-7 of the job variable.

The user can thus use job variables to control interactive procedures or ENTER tasks. The following table shows the relationship between error weight and return code in the job variable. Both operator errors and errors in the presource (e.g. structure errors) are evaluated.

Job variable values:

Error class	Termination	Return code in the job variable
No error	Normal	0000
Warning	Normal	1003
Soft error	Normal	2004
Serious error	Immediate	2005

11 Appendix

11.1 ASSEMBH messages

The messages are arranged as follows:

```
Message number  Flag Weight       Line 1
Message number  Text English      Line 2
Message number  Text German       Line 3
-------------------------------------------------------------------------
ASS0110  A10  SIGNIFICANT ERROR
ASS0110  RELOCATABLE TERM IN PRODUCT OR DIVISION
ASS0110  PRODUKT ODER QUOTIENT ENTHAELT RELATIVEN ELEMENTARAUSDRUCK

ASS0111  ......
ASS0111  ......
ASS0111  ......

ASS0112  ......
  ...    ......
  ...    ......

   .
   .
   .
```

```
ASS0110    A10 - SIGNIFICANT ERROR
ASS0110    RELOCATABLE TERM NOT ALLOWED IN MULTIPLICATION OR DIVISION
ASS0110    RELATIVER AUSDRUCK IN MULTIPLIKATION ODER DIVISION UNZULAESSIG
```

Meaning
The argument used in multiplication/division is a relocatable value.

```
ASS0111    A11 - SIGNIFICANT ERROR
ASS0111    'EQU' EXPRESSION CANNOT BE EVALUATED
ASS0111    'EQU'-AUSDRUCK NICHT BERECHENBAR

ASS0112    A12 - SIGNIFICANT ERROR
ASS0112    'EQU' INSTRUCTION WITHIN XDSEC ILLEGAL
ASS0112    'EQU'-ANWEISUNG INNERHALB 'XDSEC' UNZULAESSIG

ASS0113    A13 - SIGNIFICANT ERROR
ASS0113    NEGATIVE RELOCATABLE ADDRESS
ASS0113    RELATIVE ADRESSE NEGATIV

ASS0114    A14 - SIGNIFICANT ERROR
ASS0114    ADDRESS OF A COMPLEX RELOCATABLE EXPRESSION CANNOT BE FOUND
ASS0114    ADRESSE EINES ZUSAMMENGESETZTEN RELATIVIERBAREN AUSDRUCKS NICHT AUFFINDBAR

ASS0115    A15 SIGNIFICANT ERROR
ASS0115    UNRESOLVABLE EXPRESSION
ASS0115    AUSDRUCK UNAUFLOESBAR

ASS0116    A16 - SIGNIFICANT ERROR
ASS0116    EXPRESSION CANNOT BE EVALUATED
ASS0116    AUSDRUCK NICHT BERECHENBAR

ASS0117    A17 - SIGNIFICANT ERROR
ASS0117    EXPRESSION IS NOT RELOCATABLE
ASS0117    AUSDRUCK NICHT RELATIV

ASS0120    A20 - SIGNIFICANT ERROR
ASS0120    VALUE OF EXPRESSION GREATER THAN 2**31 - 1
ASS0120    WERT DES AUSDRUCKS GROESSER ALS 2**31 - 1

ASS0121    A21 - SIGNIFICANT ERROR
ASS0121    ILLEGAL NEGATIVE ADDRESS
ASS0121    NEGATIVE ADRESSE IST UNZULAESSIG

ASS0210    B10 - SIGNIFICANT ERROR
ASS0210    ILLEGAL OPERAND IN 'ICTL' OR 'ISEQ' INSTRUCTION
ASS0210    OPERAND IN 'ICTL'- ODER 'ISEQ'-ANWEISUNG UNZULAESSIG
```

```
ASS0211    B11 - SIGNIFICANT ERROR
ASS0211    'ICTL' MUST BE THE FIRST INSTRUCTION STATEMENT IN PROGRAM
ASS0211    'ICTL' MUSS ERSTE ANWEISUNG IM PROGRAMM SEIN

ASS0212    B12 - SIGNIFICANT ERROR
ASS0212    PRIMARY COLUMN IN 'ICTL' OPERAND MISSING
ASS0212    ANFANGSSPALTE IN 'ICTL'-OPERAND FEHLT

ASS0213    B13 - SIGNIFICANT ERROR
ASS0213    PRIMARY COLUMN IN 'ICTL' OPERAND IS NO DIRECT VALUE
ASS0213    ANFANGSSPALTE IN 'ICTL'-OPERAND KEIN DIREKTWERT

ASS0214    B14 - SIGNIFICANT ERROR
ASS0214    PRIMARY COLUMN IN 'ICTL' OPERAND IS WRONG
ASS0214    ANFANGSSPALTE IN 'ICTL'-OPERAND FEHLERHAFT

ASS0215    B15 - SIGNIFICANT ERROR
ASS0215    LAST COLUMN IN 'ICTL' OPERAND IS NO DIRECT VALUE
ASS0215    END-SPALTE IN 'ICTL'-OPERAND KEIN DIREKTWERT

ASS0216    B16 - SIGNIFICANT ERROR
ASS0216    LAST COLUMN IN 'ICTL' OPERAND IS WRONG
ASS0216    END-SPALTE IN 'ICTL'-OPERAND FEHLERHAFT

ASS0217    B17 - SIGNIFICANT ERROR
ASS0217    CONTINUE COLUMN IN 'ICTL' OPERAND IS WRONG
ASS0217    FORTSETZUNGSSPALTE IN 'ICTL'-OPERAND FEHLERHAFT

ASS0218    B18 - SIGNIFICANT ERROR
ASS0218    MAINTENANCE OPTION 'MONSYS-RECORDS' NOT GIVEN
ASS0218    MAINTENANCE-OPTION 'MONSYS-RECORDS' NICHT GESETZT

ASS0220    B20 - WARNING
ASS0220    ILLEGAL 'START' INSTRUCTION
ASS0220    'START'-ANWEISUNG UNZULAESSIG

ASS0221    B21 SERIOUS ERROR
ASS0221    SECTION (&00) DOES NOT EXIST
ASS0221    SECTION (&00) NICHT VORHANDEN

ASS0230    B30 - SIGNIFICANT ERROR
ASS0230    ILLEGAL 'START' VALUE
ASS0230    'START'-WERT UNGUELTIG

ASS0231    B31 - SIGNIFICANT ERROR
ASS0231    ILLEGAL ATTRIBUTE (&00) IN 'CSECT' OR 'START' INSTRUCTION
ASS0231    MERKMAL (&00) IN 'CSECT'- ODER 'START'-ANWEISUNG UNZULAESSIG
```

```
ASS0233    B33 - SIGNIFICANT ERROR
ASS0233    ILLEGAL OPERAND IN 'END' INSTRUCTION
ASS0233    OPERAND IN 'END'-ANWEISUNG UNGUELTIG

ASS0234    B34 - WARNING
ASS0234    LENGTH OF ATTRIBUTED 'CSECT' (&00) IS ZERO; LINK PROBLEMS ARE POSSIBLE
ASS0234    'CSECT' (&00) MIT MERKMAL-ANGABE HAT LAENGE NULL; BINDERPROBLEME MOEGLICH
```

Meaning
A subsequent CSECT may receive the attributes of CSECT (&00) during loading.

Response
Remove the CSECT or attributes.

```
ASS0240    B40 - SIGNIFICANT ERROR
ASS0240    ILLEGAL OPCODE IN NAME OR OPERAND FIELD OF 'OPSYN' INSTRUCTION
ASS0240    OPERATIONS-CODE IM NAMENS- ODER OPERANDENFELD EINER 'OPSYN'-ANWEISUNG UNGUELTIG

ASS0241    B41 - SIGNIFICANT ERROR
ASS0241    'OPSYN' INSTRUCTION NOT ALLOWED IN MACROS
ASS0241    'OPSYN'-ANWEISUNG INNERHALB VON MAKROS UNZULAESSIG

ASS0242    B42 - SIGNIFICANT ERROR
ASS0242    'COPY' MEMBER NOT FOUND
ASS0242    'COPY'-ELEMENT NICHT GEFUNDEN
```

Response
Possible responses:
- Specify the COPY library in the assembler options;
- Correct the element ("member") name in the COPY operand.

```
ASS0243    B43 - SIGNIFICANT ERROR
ASS0243    NAME OF 'COPY' MEMBER INVALID
ASS0243    NAME DES 'COPY'-ELEMENTES FEHLERHAFT
```

Meaning
The first character in the name of a COPY element ("member") must be alphabetic; the remaining characters may be either letters or digits. The maximum length for the name of a COPY element is 64 characters.

```
ASS0244    B44 - WARNING
ASS0244    MACRO NAME IN PROTOTYPE STATEMENT AND LIBRARY MEMBER NAME DIFFER
ASS0244    MAKRONAME IN MUSTERANWEISUNG UND BIBLIOTHEKSELEMENTNAME UNTERSCHIEDLICH
```

```
ASS0245   B45 - SIGNIFICANT ERROR
ASS0245   ILLEGAL OPERAND IN 'COPY' INSTRUCTION
ASS0245   'COPY'-OPERAND FEHLERHAFT
```

Meaning
There is no operand, more than one operand, or an operand with an illegal syntax in the COPY instruction. The COPY element ("member") was not inserted.

Response
Correct the operand.

```
ASS0246   B46 - SIGNIFICANT ERROR
ASS0246   MAXIMUM 'COPY-LEVEL' (&00) EXCEEDED
ASS0246   MAXIMALER 'COPY-LEVEL' (&00) UEBERSCHRITTEN
```

```
ASS0247   B47 - FATAL ERROR
ASS0247   THE MAXIMUM MACRO LEVEL OF (&00) HAS BEEN REACHED
ASS0247   MAXIMALE MAKRO-VERSCHACHTELUNGSTIEFE VON (&00) ERREICHT
```

Meaning
The maximum macro nesting level specified in the assembler option (MAX-MACRO-NEST-LEVEL) has been reached (default value: 255).

Response
Correct the assembler option or check the macro calls in the program for an endless loop.

```
ASS0248   B48 - NOTE
ASS0248   ATTENTION: SOURCE CONTAINS 'OPSYN' INSTRUCTIONS
ASS0248   VORSICHT: QUELLPROGRAMM ENTHAELT 'OPSYN'-ANWEISUNGEN
```

Meaning
The effectiveness of 'OPSYN' instructions regarding domain and duration, is to be considered especially in conjunction with (library) macros.
See also ASSEMBH-Beschreibung: Unterschiede ASSEMBH und ASSEMB V30.0A .

```
ASS0249   B49 - NOTE
ASS0249   'OPSYN' INACTIVATED
ASS0249   'OPSYN' INAKTIVIERT
```

```
ASS0250    B50 - WARNING
ASS0250    UNEXPECTED EOF BEFORE 'END' INSTRUCTION
ASS0250    EOF VOR 'END'-ANWEISUNG AUFGETRETEN
```

Meaning
EOF was encountered before the END instruction when reading the source. The ASSEMBH generates an END instruction statement and continues the assembly.

```
ASS0251    B51 - NOTE
ASS0251    'MEND' INSTRUCTION MISSING
ASS0251    'MEND'-ANWEISUNG FEHLT
```

Meaning
MEND instruction missing in library macros. Sequence errors cannot occur, as the MEND instruction statement is generated.

Response
Insert the MEND instruction statement.

```
ASS0252    B52 - WARNING
ASS0252    INPUT RECORD TOO LONG; MAXIMUM LENGTH = 256
ASS0252    EINGABESATZ ZU LANG; MAXIMALLAENGE = 256
```

```
ASS0254    B54 - WARNING
ASS0254    UNEXPECTED EOF
ASS0254    UNERWARTETES EOF
```

Meaning
EOF was encountered before the END or MEND instruction when reading a file or a library element.

Response
Insert the missing END or MEND instruction statement.

```
ASS0255    B55 - SIGNIFICANT ERROR
ASS0255    'MEND' INSTRUCTION MISSING
ASS0255    'MEND'-ANWEISUNG FEHLT
```

Meaning
MEND instruction missing in source deck macros or in library macros with inner macro definitions. This may prevent the macro from being generated or cause it to be generated incorrectly.

Response
Insert the MEND instruction statement.

```
ASS0256    B56 - NOTE
ASS0256    'END' INSTRUCTION IS GENERATED
ASS0256    'END'-ANWEISUNG GENERIERT

ASS0257    B57 - NOTE
ASS0257    'END' INSTRUCTION GENERATED BY MACRO EXPANSION
ASS0257    'END'-ANWEISUNG DURCH MAKRO-GENERIERUNG ERZEUGT

ASS0258    B58 - NOTE
ASS0258    THIS STATEMENT IS NO LONGER SUPPORTED
ASS0258    ANWEISUNG NICHT MEHR UNTERSTUETZT

ASS0259    B59 - NOTE
ASS0259    'CSECT' WITH NO NAME IS GENERATED
ASS0259    NAMENLOSE 'CSECT' WIRD GENERIERT

ASS0260    B60 - NOTE
ASS0260    THE 'MCALL'/'GSEQ' INSTRUCTIONS IN MACRO (&00) ARE NO LONGER NEEDED IN ASSEMBH
ASS0260    IM MAKRO (&00) AUFGETRETENE 'MCALL'/'GSEQ'-ANWEISUNG IM ASSEMBH NICHT MEHR
           BENOETIGT
```

Meaning

The instructions are no longer required and will be ignored.

```
ASS0261    B61 - NOTE
ASS0261    INCOMPLETE PROGRAM; NO OBJECT GENERATION
ASS0261    PROGRAMM UNVOLLSTAENDIG; KEINE OBJEKTERZEUGUNG

ASS0262    B62 - NOTE
ASS0262    THE 'MCALL' OR 'GSEQ' INSTRUCTION IS NO LONGER NEEDED IN ASSEMBH
ASS0262    'MCALL'/'GSEQ'-ANWEISUNG IN ASSEMBH NICHT MEHR BENOETIGT

ASS0270    B70 - SIGNIFICANT ERROR
ASS0270    PROCEDURE NAME IN '$LSDL' STATEMENT MISSING
ASS0270    PROZEDUR-NAME BEI '$LSDL'-ANWEISUNG FEHLT

ASS0271    B71 - SIGNIFICANT ERROR
ASS0271    WRONG PROCEDURE NAME IN '$LSDLSAVE' STATEMENT
ASS0271    PROZEDUR-NAME BEI '$LSDLSAVE'-ANWEISUNG FEHLERHAFT

ASS0272    B72 - SIGNIFICANT ERROR
ASS0272    TYPE AND NAME IN '$LSDL' STATEMENT DO NOT CORRESPOND
ASS0272    TYP UND NAME BEI '$LSDL'-ANWEISUNG PASSEN NICHT ZUEINANDER

ASS0273    B73 - SIGNIFICANT ERROR
ASS0273    MORE THAN 3 OPERANDS IN '$LSDL' STATEMENT
ASS0273    MEHR ALS 3 OPERANDEN IN '$LSDL'-ANWEISUNG
```

```
ASS0274    B74 - SIGNIFICANT ERROR
ASS0274    TYPE IN '$LSDL' STATEMENT MISSING
ASS0274    TYP BEI '$LSDL'-ANWEISUNG FEHLT

ASS0275    B75 - WARNING
ASS0275    'CCW' FLAG BYTE WAS NOT CHECKED
ASS0275    'CCW'-FLAGBYTE NICHT GEPRUEFT

ASS0276    B76 - WARNING
ASS0276    CONSISTENCY CONSTANT IS GENERATED FOR EMPTY 'CSECT' SECTION
ASS0276    KONSISTENZ-KONSTANTE FUER LEERE 'CSECT' GENERIERT

ASS0311    C11 - SIGNIFICANT ERROR
ASS0311    ILLEGAL CONCATENATION
ASS0311    KONKATENIERUNG UNZULAESSIG
```

Meaning
Only variable symbols (including generated or subscripted symbols) are permitted in the operand field of an LCL/GBL instruction and in the name field of a SET instruction. Concatenation is not legal.

```
ASS0312    C12 - SIGNIFICANT ERROR
ASS0312    ILLEGAL DIMENSION SPECIFIED
ASS0312    DIMENSIONSANGABE FEHLERHAFT
```

Meaning
The dimension in the operand field of a LCL or GBL instruction must be an unsigned decimal number.

```
ASS0313    C13 - SIGNIFICANT ERROR
ASS0313    SYNTAX ERROR IN THE SUBSCRIPT OF A VARIABLE SYMBOL
ASS0313    SYNTAX-FEHLER IM INDEX EINES VARIABLEN PARAMETERS
```

Meaning
The subscript of a variable symbol must be a SETA expression.

```
ASS0321    C21 - SIGNIFICANT ERROR
ASS0321    OPERAND (&00) IS A SYMBOLIC PARAMETER IN A MACRO PROTOTYPE STATEMENT
ASS0321    OPERAND (&00) IST SYMBOLISCHER PARAMETER IN MUSTERANWEISUNG
```

Meaning
A variable symbol cannot be a symbolic parameter and a SET symbol at the same time.

```
ASS0322    C22 - SIGNIFICANT ERROR
ASS0322    ILLEGAL SYMBOLIC PARAMETER IN OPERAND FIELD OF 'LCL' OR 'GBL' INSTRUCTION
ASS0322    SYMBOLISCHE PARAMETER IM OPERANDENFELD EINER 'LCL'/'GBL'-ANWEISUNG UNZULAESSIG
```

```
ASS0335   C35 - SERIOUS ERROR
ASS0335   SERIOUS ERROR(S) FOR 'MACRO' OR PROTOTYPE STATEMENT OF A LIBRARY MACRO
ASS0335   SERIOUS ERROR(S) ZU 'MACRO' ODER MUSTERANWEISUNG EINES BIBLIOTHEKSMAKROS
```

Meaning
Some of the SERIOUS ERRORS for this macro instruction pertain to the associated macro definition header statement (MACRO) or the macro instruction prototype statement.

Response
Check MACRO and prototype statement.

```
ASS0336   C36 - NOTE
ASS0336   MACRO (&00) MULTIPLY DEFINED IN SOURCE
ASS0336   MAKRO (&00) IN DER SOURCE MEHRFACH DEFINIERT
```

Meaning
Note concerning incompatibility: A macro instruction will always generate the macro whose definition was processed last.

```
ASS0337   C37 - NOTE
ASS0337   NOTE(S) FOR 'MACRO' OR PROTOTYPE STATEMENT OF A LIBRARY MACRO
ASS0337   NOTE(S) ZU 'MACRO'- ODER MUSTERANWEISUNG EINES BIBLIOTHEKSMAKROS
```

Meaning
Some of the NOTES for this macro instruction pertain to the associated macro definition header statement (MACRO) or the macro instruction prototype statement.

Response
Check MACRO and prototype statement.

```
ASS0338   C38 - WARNING
ASS0338   WARNING(S) FOR 'MACRO' OR PROTOTYPE STATEMENT OF A LIBRARY MACRO
ASS0338   WARNING(S) ZU 'MACRO'- ODER MUSTERANWEISUNG EINES BIBLIOTHEKSMAKROS
```

Meaning
Some of the WARNINGS for this macro instruction pertain to the associated macro definition header statement (MACRO) or the macro instruction prototype statement.

Response
Check MACRO and prototype statement.

```
ASS0339   C39 - SIGNIFICANT ERROR
ASS0339   SIGNIFICANT ERROR(S) FOR 'MACRO' OR PROTOTYPE STATEMENT OF A LIBRARY MACRO
ASS0339   SIGNIFICANT ERROR(S) ZU 'MACRO'- ODER MUSTERANWEISUNG EINES BIBLIOTHEKSMAKROS
```

Meaning
Some of the SIGNIFICANT ERRORS for this macro instruction pertain to the associated macro definition header statement (MACRO) or the macro instruction prototype statement.

Response
Check MACRO and prototype statement.

```
ASS0340   C40 - SIGNIFICANT ERROR
ASS0340   MACRO PROTOTYPE STATEMENT HAS INVALID OPCODE
ASS0340   OPERATIONSCODE DER MUSTERANWEISUNG UNGUELTIG
```

Meaning
The macro name is longer than 64 characters or contains illegal characters. The macro is not generated.

```
ASS0341   C41 - SIGNIFICANT ERROR
ASS0341   MISSING OPCODE IN MACRO PROTOTYPE STATEMENT
ASS0341   OPERATIONSCODE IN MUSTERANWEISUNG FEHLT
```

Meaning
The operation code (=macro name) is missing in a macro prototype statement. The macro is not generated.

```
ASS0342   C42 - SIGNIFICANT ERROR
ASS0342   WRONG OPCODE IN FIRST STATEMENT OF LIBRARY MACRO
ASS0342   OPERATIONSCODE IN 1.ANWEISUNG EINES BIBLIOTHEKSMAKROS FEHLERHAFT
```

Meaning
The opcode in the first statement of a library macro (excluding blank lines, comments, or macro remarks) contains syntax errors or variable symbols.

Response
Correct/insert the MACRO statement.

```
ASS0343   C43 - SIGNIFICANT ERROR
ASS0343   MACRO DOES NOT CONTAIN ANY PROTOTYPE STATEMENT
ASS0343   MAKRO ENTHAELT KEINE MUSTERANWEISUNG
```

Meaning
No macro prototype statement (the first statement after the macro instruction, excluding blank lines, comments, or macro remarks) was found when expanding a macro.

Response
Insert or correct the macro prototype statement.

```
ASS0344   C44 - SIGNIFICANT ERROR
ASS0344   LIBRARY MACRO DOES NOT BEGIN WITH A 'MACRO' STATEMENT
ASS0344   BIBLIOTHEKSMAKRO BEGINNT NICHT MIT 'MACRO'-ANWEISUNG

ASS0346   C46 - SIGNIFICANT ERROR
ASS0346   MISSING OPCODE IN FIRST STATEMENT OF A LIBRARY MACRO
ASS0346   OPERATIONSCODE IN 1.ANWEISUNG EINES BIBLIOTHEKS-MAKROS FEHLT
```

Meaning
Missing opcode in the first statement of a library macro (excluding blank lines, comments, or macro remarks). This must always be the macro definition header statement MACRO.

Response
Insert or correct the macro prototype statement.

```
ASS0347   C47 - SIGNIFICANT ERROR
ASS0347   ERROR IN OPCODE OR OPERAND FIELD OF THE CORRESPONDING PROTOTYPE STATEMENT;
          MACRO WILL NOT BE GENERATED
ASS0347   OPCODE- ODER OPERANDENFELD DER ZUGEHOERIGEN MUSTERANWEISUNG FEHLERHAFT; MAKRO
          WIRD NICHT GENERIERT

ASS0348   C48 - SIGNIFICANT ERROR
ASS0348   MEND INSTRUCTION IS GENERATED
ASS0348   'MEND'-ANWEISUNG WURDE GENERIERT
```

```
ASS0349     C49 - NOTE
ASS0349     OPERAND FIELD OF THE PROTOTYPE STATEMENT ENDS WITH A COMMA
ASS0349     OPERANDENFELD DER MUSTERANWEISUNG ENDET MIT KOMMA
```

Meaning
The terminating comma could indicate that further operands follow. If this is the case, they will begin in the wrong column of the continuation line and thus be treated as comments.

Response
Check whether the continuation line(s), if any, begin at the correct starting column, or remove the comma.

```
ASS0351     C51 - SIGNIFICANT ERROR
ASS0351     SYMBOLIC PARAMETER (&00) OCCURS MORE THAN ONCE IN PROTOTYPE STATEMENT
ASS0351     SYMBOLISCHER PARAMETER (&00) TRITT IN MUSTERANWEISUNG MEHRFACH AUF

ASS0352     C52 - SIGNIFICANT ERROR
ASS0352     PRIMARY VALUE OF KEYWORD PARAMETER IN PROTOTYPE STATEMENT CANNOT BE GENERATED
ASS0352     GENERIERUNG DES ANFANGSWERTES EINES KENNWORT-OPERANDEN IN MUSTERANWEISUNG
            UNZULAESSIG

ASS0356     C56 - SIGNIFICANT ERROR
ASS0356     EMPTY PARAMETER IN PROTOTYPE STATEMENT
ASS0356     OPERAND IN MUSTERANWEISUNG LEER
```

Meaning
An empty parameter is not permitted in the prototype statement.

Response
Correct/insert the empty or missing parameter.

```
ASS0381     C81 - WARNING
ASS0381     UNDEFINED KEYWORD PARAMETER (&00); OPERAND WAS INTERPRETED AS A POSITIONAL
            OPERAND
ASS0381     KENNWORT-OPERAND (&00) UNDEFINIERT; OPERAND WIRD ALS STELLUNGS-OPERAND
            INTERPRETIERT

ASS0401     D01 - SIGNIFICANT ERROR
ASS0401     INVALID CONSTANT TYPE
ASS0401     KONSTANTENTYP UNGUELTIG
```

Meaning
An invalid constant type was specified in a DC or DS instruction or in a literal.

Response
Correct the constant type in the statement.

```
ASS0402    D02 - SIGNIFICANT ERROR
ASS0402    LENGTH MODIFIER ERROR
ASS0402    LAENGENFAKTOR FEHLERHAFT
```

Meaning
The length modifier of a DC/DS instruction or a literal has a syntax error, or its value lies outside the permissible range.

Response
Correct the syntax or value of the length modifier.

```
ASS0403    D03 - SIGNIFICANT ERROR
ASS0403    CONSTANT OF TYPE S ILLEGAL IN A LITERAL STRING
ASS0403    S-KONSTANTE IN LITERALEN UNZULAESSIG

ASS0404    D04 - SIGNIFICANT ERROR
ASS0404    QUOTES NOT PAIRED OR ILLEGAL TERMINATION OF A QUOTED STRING
ASS0404    HOCHKOMMATA NICHT PAARWEISE ODER UNERLAUBTE BEENDIGUNG EINER ZEICHENKETTE

ASS0405    D05 - SIGNIFICANT ERROR
ASS0405    EMPTY OPERAND
ASS0405    OPERAND IST LEER

ASS0407    D07 - SIGNIFICANT ERROR
ASS0407    ALIGNMENT ERROR IN OPERAND (&00)
ASS0407    AUSRICHTUNGSFEHLER IN OPERAND (&00)
```

Meaning
The operand must be aligned on a halfword, fullword, or doubleword boundary.

```
ASS0408    D08 - SIGNIFICANT ERROR
ASS0408    UNPAIRED '&' IN CONSTANT VALUE OF A DC/DS OPERAND OR LITERAL
ASS0408    UNGEPAARTES '&' IM KONSTANTENWERT EINES DC/DS-OPERANDEN ODER LITERALS

ASS0409    D09 - SIGNIFICANT ERROR
ASS0409    DISPLACEMENT IN OPERAND (&00) NOT IN THE RANGE 0 TO 4095
ASS0409    DISTANZANGABE IN OPERAND (&00) NICHT 0 BIS 4095
```

Meaning
The displacement does not lie in the range 0 to 4095 (inclusive).

```
ASS0411    D11 - SIGNIFICANT ERROR
ASS0411    LENGTH SPECIFICATION IN OPERAND (&00) NOT IN THE RANGE 1 TO 16
ASS0411    LAENGENANGABE IN OPERAND (&00) NICHT 1 BIS 16
```

```
ASS0412    D12 - SIGNIFICANT ERROR
ASS0412    DUPLICATION FACTOR ERROR
ASS0412    WIEDERHOLUNGSFAKTOR FEHLERHAFT
```

Meaning
The duplication factor of a DC/DS operand or literal has a syntax error, or its value lies outside the legal range.
Permissible range: 0 to 2**24 - 1

Response
Correct the syntax or value of the duplication factor.

```
ASS0413    D13 - SIGNIFICANT ERROR
ASS0413    SCALE MODIFIER ERROR
ASS0413    SKALENFAKTOR FEHLERHAFT
```

Meaning
The scale modifier of a DC/DS operand or literal has a syntax error, or its value lies outside the legal range. The permissible range depends on the type of constant.

Response
Correct the syntax or value of the scale modifier.

```
ASS0414    D14 - SIGNIFICANT ERROR
ASS0414    EXPONENT MODIFIER ERROR
ASS0414    EXPONENTENFAKTOR FEHLERHAFT
```

Meaning
The exponent modifier of a DC/DS operand or literal has a syntax error, or its value lies outside the legal range.
Permissible range: -85 to +75

Response
Correct the syntax or value of the exponent modifier.

```
ASS0415    D15 - SIGNIFICANT ERROR
ASS0415    PRECISION LOST IN DC CONSTANT
ASS0415    GENAUIGKEITSVERLUST IN DC-KONSTANTE
```

Meaning
When positions are lost in the constant due to the specification of a scale modifier, the precision of the constant is reduced.

Response
Specify an appropriate scale modifier.

```
ASS0416   D16 - SIGNIFICANT ERROR
ASS0416   SELFDEFINING TERM (&00) TOO LARGE
ASS0416   SELBSTDEFINIERENDER WERT (&00) ZU GROSS

ASS0417   D17 - SIGNIFICANT ERROR
ASS0417   ARITHMETIC OVERFLOW
ASS0417   ARITHMETISCHER UEBERLAUF
```

Meaning
The final result or an intermediate result obtained when evaluating an arithmetic expression does not lie within the range 2**31-1 and -2**31.

Response
Change the arithmetic expression so that an overflow no longer occurs.

```
ASS0418   D18 - SIGNIFICANT ERROR
ASS0418   FLOATING-POINT CHARACTERISTIC OUT OF RANGE
ASS0418   GLEITPUNKTCHARAKTERISTIK AUSSERHALB DES ZULAESSIGEN BEREICHS
```

Meaning
The characteristic portion, i.e. the hexadecimal exponent, of a floating-point number
(type E, D, or L) is less than -64 or greater than 64 and thus lies outside the permissible range.

Response
Correct the constant.

```
ASS0419   D19 - SIGNIFICANT ERROR
ASS0419   INVALID CHARACTER IN CONSTANT VALUE OF A DC/DS OPERAND OR LITERAL
ASS0419   UNGUELTIGES ZEICHEN IM KONSTANTEN-WERT EINES DC-/DS-OPERANDEN ODER LITERALS
```

Meaning
The constant contains characters that are not permitted for this type of constant.

Response
Correct the constant.

```
ASS0421   D21 - SIGNIFICANT ERROR
ASS0421   SYNTAX ERROR IN 'EQU' OPERAND
ASS0421   SYNTAX-FEHLER IN 'EQU'-OPERAND
```

```
ASS0422    D22 - SIGNIFICANT ERROR
ASS0422    INVALID LENGTH ATTRIBUTE IN 'EQU' OPERAND
ASS0422    LAENGENMERKMAL IN 'EQU'-OPERAND FEHLERHAFT
```

Meaning
The value of the length attribute must be between 0 and 2**24 - 1.

Response
Correct the explicitly specified length attribute.

```
ASS0423    D23 - SIGNIFICANT ERROR
ASS0423    INVALID TYPE ATTRIBUTE IN 'EQU' OPERAND
ASS0423    TYPENMERKMAL IN 'EQU'-OPERAND FEHLERHAFT
```

Meaning
The type attribute must be a self-defining term (max. 1 byte long).

Response
Correct the explicitly specified type attribute.

```
ASS0424    D24 - SIGNIFICANT ERROR
ASS0424    LIMIT VALUES OF EXPONENT OUT OF RANGE
ASS0424    GRENZWERTE DER EXPONENTEN AUSSERHALB DES ZULAESSIGEN BEREICHS
```

Meaning
The sum of the internal and external exponents of a DC constant exceeds or lies below the prescribed limit values.

Response
Correct the exponent entry.

```
ASS0425    D25 - SIGNIFICANT ERROR
ASS0425    STRING VALUE (&00) CANNOT BE CONVERTED IN ARITHMETIC VALUE
ASS0425    ZEICHENWERT (&00) IN ARITHMETISCHEN WERT NICHT KONVERTIERBAR

ASS0427    D27 - SIGNIFICANT ERROR
ASS0427    ADDRESS CONSTANT CANNOT BE EVALUATED; NO GENERATION
ASS0427    ADRESSKONSTANTE NICHT BERECHENBAR. KEINE GENERIERUNG
```

```
ASS0428   D28 - SIGNIFICANT ERROR
ASS0428   CONSTANT VALUE OR EXPONENT OF A DC/DS OPERAND OR LITERAL OUT OF RANGE. DEFAULT
          VALUE 0 IS INSERTED
ASS0428   KONSTANTENWERT ODER EXPONENT EINES DC-/DS-OPERANDEN BZW. LITERALS AUSSERHALB
          DES ZULAESSIGEN BEREICHS; ERSATZWERT '0' WIRD VERWENDET
```

Meaning

Possible causes:

– The value of the constant exceeds the value range defined for the type of constant.
– The sum of the exponent and the exponent modifier lies outside the permissible range.

Permissible range: -85 to +75.

```
ASS0429   D29 - SIGNIFICANT ERROR
ASS0429   ARITHMETIC OVERFLOW AFTER CONVERSION OF (&00)
ASS0429   ARITHMETISCHER UEBERLAUF NACH (&00) KONVERTIERUNG
```

Meaning

The self-defining term cannot be converted into the internal (binary) representation, since the conversion would produce a binary value of more than 32 bits.
(&00): string to be converted.

Response

Correct the self-defining term so that its internal representation can be stored in one word (32 bits).

```
ASS0430   D30 - SERIOUS ERROR
ASS0430   INVALID REGISTER SPECIFICATION IN OPERAND (&00); EVEN NUMBERED VALUE BETWEEN 0
          AND 14 REQUIRED
ASS0430   REGISTER-ANGABE IN OPERAND (&00) UNGUELTIG; NUR GERADZAHLIGE NR. ZWISCHEN 0 UND
          14 ZULAESSIG

ASS0431   D31 - SERIOUS ERROR
ASS0431   INVALID FLOATING-POINT REGISTER SPECIFICATION IN OPERAND (&00); ONLY 0, 2, 4,
          OR 6 ALLOWED
ASS0431   GLEITPUNKTREGISTER-ANGABE IN OPERAND (&00) UNGUELTIG; NUR 0, 2, 4 ODER 6
          ZULAESSIG

ASS0432   D32 - SERIOUS ERROR
ASS0432   INVALID REGISTER SPECIFICATION; DIRECT VALUE (0 TO 15) EXPECTED
ASS0432   REGISTER-ANGABE UNGUELTIG; DIREKTWERT (0 BIS 15) WIRD ERWARTET
```

```
ASS0433    D33 - SIGNIFICANT ERROR
ASS0433    RELOCATABLE VALUE INVALID AS BASE REGISTER; VALUE MUST BE ABSOLUTE AND BETWEEN
           0 AND 15
ASS0433    RELATIVWERT ALS BASISREGISTER-ANGABE UNZULAESSIG. ABSOLUTWERT ZWISCHEN 0 UND 15
           ZULAESSIG

ASS0434    D34 - SERIOUS ERROR
ASS0434    INVALID PAIR NUMBER OF FLOATING-POINT REGISTER IN OPERAND (&00); VALUE 0 OR 4
           REQUIRED
ASS0434    GLEITPUNKTREGISTER-PAARNUMMER IN OPERAND (&00) UNGUELTIG; NUR 0 ODER 4
           ZULAESSIG

ASS0435    D35 - SIGNIFICANT ERROR
ASS0435    ILLEGAL SPECIFICATION OF A BASE REGISTER
ASS0435    BASISREGISTER-ANGABE UNZULAESSIG
```

Meaning
An operand of the instruction has the wrong format. Instead of a register or direct value,
a displacement and base register was specified.

Response
Correct the format of the operand.

```
ASS0436    D36 - SIGNIFICANT ERROR
ASS0436    ILLEGAL SPECIFICATION OF A BASE REGISTER AND INDEX REGISTER OR LENGTH
ASS0436    BASISREGISTER-ANGABE UND INDEXREGISTER- BZW. LAENGENANGABE UNZULAESSIG
```

Meaning
An operand of the instruction has the wrong format. Instead of a register or direct value,
a displacement, base register, and index register or length was specified.

Response
Correct the format of the operand.

```
ASS0437    D37 - SIGNIFICANT ERROR
ASS0437    ILLEGAL INDEX REGISTER OR LENGTH SPECIFICATION
ASS0437    INDEXREGISTER- BZW. LAENGENANGABE UNZULAESSIG
```

Meaning
An operand of the instruction has the wrong format. An index register or a length was specified in addition to a displacement and base register.

Response
Correct the format of the operand.

```
ASS0438    D38 - SERIOUS ERROR
ASS0438    ILLEGAL REGISTER SPECIFICATION IN OPERAND (&00)
ASS0438    REGISTERANGABE IN OPERAND (&00) UNGUELTIG
```

Meaning
Register numbers may only assume specific values for DUET instructions.

```
ASS0439    D39 - SIGNIFICANT ERROR
ASS0439    ADDRESS VALUE IN OPERAND (&00) OUT OF RANGE
ASS0439    ADRESSWERT IN OPERAND (&00) AUSSERHALB DES ZULAESSIGEN BEREICHS
```

Meaning
The values of the addresses exceed the permissible limits for DUET instructions.

```
ASS0441    D41 - SIGNIFICANT ERROR
ASS0441    QUOTES NOT PAIRED
ASS0441    UNGEPAARTE APOSTROPHE
```

```
ASS0442    D42 - SIGNIFICANT ERROR
ASS0442    SYNTAX ERROR IN DC/DS INSTRUCTION OR LITERAL
ASS0442    SYNTAX-FEHLER IN DC/DS-ANWEISUNG ODER LITERAL
```

```
ASS0443    D43 - SIGNIFICANT ERROR
ASS0443    SCALE OR EXPONENT MODIFIER ILLEGAL
ASS0443    SKALEN- BZW. EXPONENTENFAKTOR UNZULAESSIG
```

Meaning
A scale and exponent modifier is only permitted for fixed and floating-point constants in DC instructions and literals.

Response
Omit the scale or exponent modifier, or change the type of constant.

```
ASS0445    D45 - SIGNIFICANT ERROR
ASS0445    INVALID LENGTH SPECIFIED IN OPERAND (&00); LENGTH MUST BE WITHIN THE RANGE 1 TO
           256
ASS0445    LAENGENANGABE IN OPERAND (&00) UNGUELTIG; LAENGE 1 BIS 256 ZULAESSIG
```

```
ASS0446    D46 - SIGNIFICANT ERROR
ASS0446    ILLEGAL SPECIFICATION OF ADDRESS; DISPLACEMENT WILL BE IGNORED
ASS0446    ADRESSANGABE FEHLERHAFT; DISTANZ WIRD IGNORIERT
```

Meaning
The specification of an address in the form of a relocatable value in relation to a base address and the explicit specification of a base register are illegal.

```
ASS0447    D47 - SIGNIFICANT ERROR
ASS0447    ILLEGAL RELOCATABLE VALUE FOR LENGTH; LENGTH 0 WILL BE INSERTED
ASS0447    RELATIVWERT ALS LAENGENANGABE UNZULAESSIG. LAENGE '0' WIRD VERWENDET
```

Meaning
The specified length must be an absolute value.

```
ASS0448    D48 - SIGNIFICANT ERROR
ASS0448    SYNTAX ERROR IN CONSTANT OF A DC/DS OPERAND OR LITERAL
ASS0448    SYNTAX-FEHLER IM KONSTANTEN-WERT EINES DC/DS-OPERANDEN ODER LITERALS

ASS0449    D49 - NOTE
ASS0449    BASE REGISTER '0' IS USED
ASS0449    BASISREGISTER '0' WIRD VERWENDET

ASS0450    D50 - NOTE
ASS0450    NO OPERAND FIELD ENTRY ALLOWED
ASS0450    OPERANDENFELD-EINTRAG UNZULAESSIG

ASS0451    D51 - SIGNIFICANT ERROR
ASS0451    REQUIRED OPERAND FIELD ENTRY MISSING
ASS0451    ERFODERLICHER OPERANDENFELD-EINTRAG FEHLT

ASS0452    D52 - SIGNIFICANT ERROR
ASS0452    WRONG OPERAND TYPE IN OPERAND (&00). VALUE '0' IS INSERTED
ASS0452    FEHLERHAFTER OPERANDENTYP IN OPERAND (&00); ERSATZWERT '0' WIRD VERWENDET
```

Meaning
An absolute value is expected (e.g. specification of a register).

```
ASS0453    D53 - WARNING
ASS0453    EMPTY OPERAND (&00)
ASS0453    OPERAND (&00) IST LEER
```

Meaning
The named operand is empty, but this does not make sense for the processed instruction.

```
ASS0454    D54 - SIGNIFICANT ERROR
ASS0454    REQUIRED NUMBER (&00) OF OPERANDS EXCEEDED; EXCESS OPERANDS WILL BE IGNORED
ASS0454    ERFORDERLICHE OPERANDENANZAHL (&00) UEBERSCHRITTEN; UEBERZAEHLIGE OPERANDEN
           IGNORIERT

ASS0455    D55 - SERIOUS ERROR
ASS0455    REQUIRED OPERAND (&00) MISSING
ASS0455    ERFORDERLICHER OPERAND (&00) FEHLT
```

```
ASS0456    D56 - SIGNIFICANT ERROR
ASS0456    TOO MANY OPERANDS
ASS0456    ZU VIELE OPERANDEN

ASS0457    D57 - SIGNIFICANT ERROR
ASS0457    TOO FEW OPERANDS
ASS0457    ZU WENIG OPERANDEN

ASS0459    D59 - NOTE
ASS0459    OPERAND (&00) HAS NO EFFECT, SINCE THE OPTION 'PREFIX=EXCEPT' IS SET.
ASS0459    OPERAND (&00) UNWIRKSAM, DA OPTION 'PREFIX=EXCEPT' GESETZT

ASS0460    D60 - NOTE
ASS0460    DIVISION BY ZERO
ASS0460    DIVISION DURCH NULL

ASS0461    D61 - SIGNIFICANT ERROR
ASS0461    STRING DOES NOT BEGIN WITH A QUOTE
ASS0461    STRING BEGINNT NICHT MIT HOCHKOMMA

ASS0462    D62 - SIGNIFICANT ERROR
ASS0462    STRING IN '$DSDDI' OR '$DSDDR' OPERAND IS TOO LONG
ASS0462    STRING IN '$DSDDI'- BZW. '$DSDDR'-OPERAND ZU LANG
```

Meaning

The string in the first operand of the $DSDDI or $DSDDR instruction must not exceed 51 characters;
a maximum of 240 characters is permitted in the second operand.

```
ASS0463    D63 - SIGNIFICANT ERROR
ASS0463    REPRO OPERAND MISSING
ASS0463    'REPRO'-OPERAND FEHLT

ASS0464    D64 - NOTE
ASS0464    OPERAND (&00) IS NO LONGER SUPPORTED
ASS0464    OPERAND (&00) NICHT MEHR UNTERSTUETZT

ASS0504    E04 - SIGNIFICANT ERROR
ASS0504    WRONG CCW0/CCW1 OPCODE
ASS0504    CCW0/CCW1-OPERATIONSCODE FEHLERHAFT

ASS0505    E05 - SIGNIFICANT ERROR
ASS0505    WRONG CCW OPCODE
ASS0505    CCW-OPERATIONSCODE FEHLERHAFT

ASS0506    E06 - SIGNIFICANT ERROR
ASS0506    WRONG CCW BYTE COUNTER
ASS0506    CCW-BYTE-ZAEHLER FEHLERHAFT
```

```
ASS0507    E07 - SIGNIFICANT ERROR
ASS0507    WRONG CCW FLAG BYTE
ASS0507    CCW-FLAGBYTE FEHLERHAFT

ASS0508    E08 - SIGNIFICANT ERROR
ASS0508    WRONG CCW1 ADDRESS; A VALUE FROM 0 TO 2**31 - 1 IS PERMITTED
ASS0508    CCW1-ADRESSE UNGUELTIG; WERT 0 BIS 2**31 - 1 ZULAESSIG

ASS0509    E09 - SIGNIFICANT ERROR
ASS0509    WRONG CCW/CCW0 ADDRESS;                A VALUE FROM 0 TO 2**24 - 1 IS PERMITTED
ASS0509    CCW/CCW0-ADRESSE UNGUELTIG; WERT 0 BIS 2**24 - 1 ZULAESSIG

ASS0510    E10 - SIGNIFICANT ERROR
ASS0510    ILLEGAL CONTINUATION LINE
ASS0510    FORTSETZUNGSZEILE UNZULAESSIG

ASS0511    E11 - SIGNIFICANT ERROR
ASS0511    EOF WAS REACHED BEFORE CONTINUATION LINE
ASS0511    EOF VOR FORTSETZUNGSZEILE

ASS0512    E12 - SIGNIFICANT ERROR
ASS0512    LAST QUOTE MISSING IN OPERAND (&00)
ASS0512    ABSCHLIESSENDES HOCHKOMMA IN OPERAND (&00) FEHLT

ASS0513    E13 - SIGNIFICANT ERROR
ASS0513    NULL STRING ILLEGAL AS CONSTANT OF A DC/DS OPERAND OR LITERAL
ASS0513    NULLSTRING ALS KONSTANTENWERT EINES DC/DS-OPERANDEN ODER LITERALS UNZULAESSIG

ASS0517    E17 - SIGNIFICANT ERROR
ASS0517    SURPLUS 'STACK'/'UNSTK' INSTRUCTION (&00) (&01)
ASS0517    UEBERZAEHLIGE 'STACK'/'UNSTK'-ANWEISUNG (&00) (&01)
```

Meaning
(&01): name of the incorrectly stored or released instruction.

```
ASS0518    E18 - SIGNIFICANT ERROR
ASS0518    ILLEGAL CHARACTER BEFORE CONTINUATION LINE
ASS0518    UNZULAESSIGE(S) ZEICHEN VOR FORTSETZUNGSSPALTE

ASS0521    E21 - SIGNIFICANT ERROR
ASS0521    REQUIRED NAME FIELD ENTRY MISSING
ASS0521    ERFORDERLICHER NAMENSFELD-EINTRAG FEHLT

ASS0524    E24 - SIGNIFICANT ERROR
ASS0524    SYMBOL NOT ALLOWED IN NAME FIELD
ASS0524    SYMBOL IM NAMENSFELD UNZULAESSIG
```

```
ASS0525    E25 - SIGNIFICANT ERROR
ASS0525    INVALID SYMBOL IN NAME FIELD
ASS0525    SYMBOL IM NAMENSFELD FEHLERHAFT

ASS0526    E26 - SIGNIFICANT ERROR
ASS0526    FIRST OPERAND IN 'AGO' INSTRUCTION IS EMPTY
ASS0526    ERSTER OPERAND IN 'AGO'-ANWEISUNG IST LEER
```

Meaning
An empty operand was encountered as the first operand in an AGO instruction. A sequence symbol or arithmetic expression and sequence symbol (computed AGO) are permitted.

```
ASS0527    E27 - SIGNIFICANT ERROR
ASS0527    INVALID SEQUENCE SYMBOL (&00) IN OPERAND (&01): NO BRANCH
ASS0527    FOLGESYMBOL (&00) IN OPERAND (&01) UNGUELTIG: KEIN SPRUNG

ASS0528    E28 - SIGNIFICANT ERROR
ASS0528    NAME OF SEQUENCE SYMBOL (&00) IS TOO LONG; MAXIMUM LENGTH = 64
ASS0528    FOLGESYMBOL (&00) ZU LANG; MAXIMALLAENGE = 64

ASS0529    E29 - SIGNIFICANT ERROR
ASS0529    OPERAND IS NOT A SEQUENCE SYMBOL
ASS0529    OPERAND IST KEIN FOLGESYMBOL

ASS0530    E30 - SIGNIFICANT ERROR
ASS0530    SYNTAX ERROR IN OPERAND (&00)
ASS0530    SYNTAX-FEHLER IN OPERAND (&00)

ASS0531    E31 - SIGNIFICANT ERROR
ASS0531    SEMANTIC ERROR IN OPERAND (&00)
ASS0531    SEMANTISCHER FEHLER IN OPERAND (&00)

ASS0532    E32 - NOTE
ASS0532    SYNTAX ERROR IN OPERAND (&00) OF THE 'SPACE' INSTRUCTION
ASS0532    SYNTAXFEHLER IN OPERAND (&00) DER 'SPACE'-ANWEISUNG
```

Meaning
The operand is incorrect, or a comment was interpreted as the operand.

```
ASS0533    E33 - SIGNIFICANT ERROR
ASS0533    ERROR IN OPERAND (&00)
ASS0533    FEHLER IN OPERAND (&00)
```

Meaning
A syntax or semantic error has occurred in the named operand. More details are usually provided in a supplementary error message.

Response
Correct the operand.

```
ASS0534    E34 - NOTE
ASS0534    DROP ISSUED FOR A RELEASED REGISTER OR ONE NOT ASSIGNED IN A 'USING'
           INSTRUCTION
ASS0534    ZU SPERRENDES REGISTER SCHON GESPERRT ODER NOCH NICHT DURCH 'USING'-ANWEISUNG
           ZUGEWIESEN

ASS0535    E35 - SIGNIFICANT ERROR
ASS0535    SEMANTIC ERROR
ASS0535    SEMANTISCHER FEHLER

ASS0538    E38 - SIGNIFICANT ERROR
ASS0538    ATTRIBUTE OF A SYMBOL CANNOT BE EVALUATED
ASS0538    MERKMAL EINES SYMBOLS NICHT BESTIMMBAR
```

Meaning
The referenced symbol is undefined or cannot be evaluated.

```
ASS0539    E39 - SIGNIFICANT ERROR
ASS0539    SYNTAX ERROR IN 'SETC' OPERAND OR IN TEXT REPLACEMENT
ASS0539    SYNTAXFEHLER IN 'SETC'-OPERAND ODER BEI TEXTERSETZUNG

ASS0540    E40 - SIGNIFICANT ERROR
ASS0540    LOGICAL EXPRESSION WRONG OR MISSING
ASS0540    LOGISCHER AUSDRUCK FEHLERHAFT BZW. NICHT VORHANDEN
```

Meaning
The logical expression in the AIF operand is missing or not enclosed in parentheses.

```
ASS0543    E43 - SIGNIFICANT ERROR
ASS0543    NAME OF THE SEQUENCE SYMBOL (&00) IS ILLEGAL
ASS0543    NAME DES FOLGESYMBOLS (&00) UNZULAESSIG
```

Meaning

The first character in a sequence symbol must be alphabetic; the remaining characters may be either letters or digits. The maximum length for a sequence symbol is 64 characters.

```
ASS0546    E46 - SIGNIFICANT ERROR
ASS0546    INVALID SYMBOL REFERENCE
ASS0546    SYMBOLZUGRIFF FEHLERHAFT
```

```
ASS0550    E50 - SIGNIFICANT ERROR
ASS0550    PARENTHESIS ERROR
ASS0550    KLAMMERUNG FEHLERHAFT
```

Meaning

A right parenthesis may be missing:

- after the base register specification in the operand
- in a parenthesized term in the operand
- in the case of a parenthesized duplication factor or modifier in a DC or DS constant or a literal.

```
ASS0552    E52 - SIGNIFICANT ERROR
ASS0552    PARENTHESIS ERROR IN OPERAND (&00)
ASS0552    KLAMMERUNG IN OPERAND (&00) FEHLERHAFT
```

```
ASS0553    E53 - SIGNIFICANT ERROR
ASS0553    ILLEGAL CHARACTER
ASS0553    UNZULAESSIGES ZEICHEN
```

```
ASS0554    E54 - SIGNIFICANT ERROR
ASS0554    ILLEGAL CHARACTER(S) IN OPERAND (&00)
ASS0554    OPERAND (&00) ENTHAELT UNZULAESSIGE(S) ZEICHEN
```

```
ASS0555    E55 - SIGNIFICANT ERROR
ASS0555    VALUE OF THE SECOND OPERAND IN SRP IS INVALID; VALUE '0' IS INSERTED
ASS0555    WERT DES 2. SRP-OPERANDEN UNGUELTIG; '0' WIRD VERWENDET
```

Meaning

The information cannot be shifted within the possible limits.

```
ASS0556    E56 - SIGNIFICANT ERROR
ASS0556    SYNTAX ERROR IN OPERAND OF A 'SETA', 'AGO', OR 'SETC' INSTRUCTION
ASS0556    SYNTAX-FEHLER IN OPERAND EINER 'SETA'-, 'AGO'- ODER 'SETC'-ANWEISUNG
```

Meaning
Syntax error in the arithmetic expression:
for SETA - entire operand field;
for 'computed AGO' - number of sequence symbol;
for SETC - duplication factor and arguments of the substring function (initial position and length).

```
ASS0557    E57 - SIGNIFICANT ERROR
ASS0557    SYNTAX ERROR IN LOGICAL EXPRESSION
ASS0557    SYNTAX-FEHLER IN LOGISCHEM AUSDRUCK

ASS0566    E66 - SIGNIFICANT ERROR
ASS0566    OPERAND (&00) IS NOT A SYMBOLIC ADDRESS
ASS0566    OPERAND (&00) IST KEINE SYMBOLISCHE ADRESSE

ASS0571    E71 - SERIOUS ERROR
ASS0571    ILLEGAL LITERAL USE IN OPERAND (&00)
ASS0571    LITERAL IN OPERAND (&00) UNZULAESSIG

ASS0572    E72 - SIGNIFICANT ERROR
ASS0572    CONSTANT OF TYPE Q NOT ALLOWED WITHIN LITERALS
ASS0572    Q-KONSTANTE IN LITERALEN UNZULAESSIG

ASS0580    E80 - NOTE
ASS0580    'TITLE' TEXT EXCEEDS 97 CHARACTERS
ASS0580    'TITLE'-TEXT LAENGER ALS 97 ZEICHEN

ASS0581    E81 - SIGNIFICANT ERROR
ASS0581    'TITLE' TEXT MISSING
ASS0581    'TITLE'-TEXT FEHLT

ASS0582    E82 - NOTE
ASS0582    EXTERNAL SYMBOL IN OPERAND FIELD IS TRUNCATED TO 8 CHARACTERS
ASS0582    EXTERNES SYMBOL IM OPERANDENFELD AUF ZULAESSIGE 8 ZEICHEN GEKUERZT
```

Meaning
The symbolic name of the external start address in the END record of the object is limited to 8 characters. Only the first 8 characters of the specified name are used.

```
ASS0593    E93 - SIGNIFICANT ERROR
ASS0593    ILLEGAL SEQUENCE SYMBOL IN NAME FIELD
ASS0593    FOLGESYMBOL IM NAMENSFELD UNZULAESSIG

ASS0594    E94 - NOTE
ASS0594    SYMBOL IN NAME FIELD IS TRUNCATED TO THE ALLOWED 8 CHARACTERS
ASS0594    SYMBOL IM NAMENSFELD AUF ZULAESSIGE 8 ZEICHEN GEKUERZT

ASS0595    E95 - SIGNIFICANT ERROR
ASS0595    SEQUENCE SYMBOL IS MISSING OR HAS A SYNTAX ERROR
ASS0595    FOLGESYMBOL FEHLT BZW. SYNTAKTISCH FALSCH

ASS0597    E97 - NOTE
ASS0597    NAME FOR OUTPUT TO ESD RECORD IS TRUNCATED TO 8 CHARACTERS
ASS0597    NAME FUER AUSGABE IN ESD-SATZ AUF 8 ZEICHEN GEKUERZT
```

Meaning
The name for entries in the ESD record of the object is limited to 8 characters. Only the first 8 characters of the name are used.

```
ASS0711    G11 - SIGNIFICANT ERROR
ASS0711    ILLEGAL 'MEND' OR 'MEXIT' INSTRUCTION
ASS0711    'MEND'- ODER 'MEXIT'-ANWEISUNG UNZULAESSIG
```

Meaning
The macro instructions MEND and MEXIT are only allowed within a macro definition.

```
ASS0712    G12 - WARNING
ASS0712    '.*' COMMENT IS ILLEGAL OUTSIDE OF MACRO DEFINITION
ASS0712    '.*'-KOMMENTAR AUSSERHALB VON MAKRODEFINITIONEN UNZULAESSIG

ASS0713    G13 - SIGNIFICANT ERROR
ASS0713    GENERATION OF A MACRO INSTRUCTION IS ILLEGAL
ASS0713    GENERIEREN EINER MAKRO-ANWEISUNG UNZULAESSIG

ASS0714    G14 - SIGNIFICANT ERROR
ASS0714    'MEND' AND 'MEXIT' INSTRUCTIONS ARE ONLY ALLOWED WITHIN MACRO DEFINITIONS
ASS0714    'MEND'- ODER 'MEXIT'-ANWEISUNG NUR IN MAKRODEFINITIONEN ZULAESSIG
```

Meaning
A MEND or MEXIT instruction was encountered in the source. They are only permitted in macro definitions.

Response
Remove the statement and check the nesting level of inner macro definitions if required.

```
ASS0724    G24 - SIGNIFICANT ERROR
ASS0724    OVERFLOW OF MAXIMUM COPY LEVEL (&00)
ASS0724    MAXIMALER COPY-LEVEL (&00) UEBERSCHRITTEN

ASS0730    G30 - SIGNIFICANT ERROR
ASS0730    GENERATED OPCODE (&00) IS NOT ALLOWED OR MUST NOT BE GENERATED
ASS0730    GENERIERTER OPERATIONS-CODE (&00) UNZULAESSIG BZW. DARF NICHT GENERIERT WERDEN
```

Meaning
The text replacement produced an incorrect operation code or one which must not be generated.

```
ASS0732    G32 - SIGNIFICANT ERROR
ASS0732    GENERATED OPCODE CONSISTS OF BLANKS
ASS0732    GENERIERTER OPERATIONS-CODE BESTEHT AUS LEERZEICHEN

ASS0734    G34 - WARNING
ASS0734    GENERATION OF '.*' COMMENTS IS NOT ALLOWED
ASS0734    GENERIEREN VON '.*'-KOMMENTAREN UNZULAESSIG
```

Meaning
A '.*' comment was to be generated.

```
ASS0736    G36 - SIGNIFICANT ERROR
ASS0736    ILLEGAL USE OF A NULL STRING WHEN A VARIABLE SYMBOL IS GENERATED
ASS0736    NULLSTRING BEI GENERIERUNG EINES VARIABLEN PARAMETERS UNZULAESSIG

ASS0737    G37 - SIGNIFICANT ERROR
ASS0737    GENERATED NAME/OPCODE FIELD CONSISTS OF MORE THAN ONE STRING
ASS0737    GENERIERTES NAMENS-/OPCODE-FELD BESTEHT AUS MEHR ALS EINEM STRING

ASS0740    G40 - SIGNIFICANT ERROR
ASS0740    ILLEGAL GENERATION OF A NULL STRING IN OPCODE FIELD
ASS0740    GENERIERUNG EINES NULLSTRINGS IN OPCODE-FELD UNZULAESSIG

ASS0811    H11 - SIGNIFICANT ERROR
ASS0811    DUMMY REGISTER EXCEEDS 4095 BYTES
ASS0811    PSEUDOREGISTER GROESSER ALS 4095 BYTES
```

Meaning
The maximum length of a DXD operand (duplication factor * length modifier) must not exceed 4095 bytes.

```
ASS0910    I10 - SIGNIFICANT ERROR
ASS0910    INVALID DIRECT VALUE IN OPERAND (&00); VALUE MUST BE FROM 0 TO 255
ASS0910    DIREKTWERT IN OPERAND (&00) UNGUELTIG; WERT 0 BIS 255 ZULAESSIG
```

```
ASS0911    I11 - SIGNIFICANT ERROR
ASS0911    INVALID DIRECT VALUE IN OPERAND (&00); VALUE MUST BE FROM 0 TO 15
ASS0911    DIREKTWERT IN OPERAND (&00) UNGUELTIG; WERT 0 BIS 15 ZULAESSIG

ASS0912    I12 - SIGNIFICANT ERROR
ASS0912    INVALID ROUNDED VALUE IN OPERAND (&00); VALUE MUST BE FROM 0 TO 9
ASS0912    RUNDUNGSWERT IN OPERAND (&00) UNGUELTIG; WERT 0 BIS 9 ZULAESSIG

ASS0913    I13 - SIGNIFICANT ERROR
ASS0913    INVALID MASK SPECIFICATION IN OPERAND (&00); VALUE MUST BE FROM 0 TO 15
ASS0913    MASKENANGABE IN OPERAND (&00) UNGUELTIG; WERT 0 BIS 15 ZULAESSIG

ASS0920    I20 - SIGNIFICANT ERROR
ASS0920    INVALID SELF-DEFINING TERM
ASS0920    SELBSTDEFINIERENDER WERT FEHLERHAFT

ASS0921    I21 - SIGNIFICANT ERROR
ASS0921    ARITHMETIC VALUE (&00) CONTAINS ILLEGAL CHARACTERS
ASS0921    ARITHMETISCHER WERT (&00) ENTHAELT UNZULAESSIGE ZEICHEN

ASS1110    K10 - WARNING
ASS1110    SEQUENCE SYMBOL (&00) ALREADY DEFINED
ASS1110    FOLGESYMBOL (&00) BEREITS DEFINIERT

ASS1230    L30 - SIGNIFICANT ERROR
ASS1230    RELOCATABLE ADDRESS CONSTANT CONTAINS NAME FROM 'DSECT'
ASS1230    ZU RELATIVIERENDE ADRESSKONSTANTE ENTHAELT NAME AUS 'DSECT'
```

Meaning

No RLD information can be generated for an entity from a DSECT.

```
ASS1250    L50 - SERIOUS ERROR
ASS1250    OPERAND 2 OF THE 'CNOP' INSTRUCTION MUST BE '4' OR '8'
ASS1250    OPERAND 2 DER 'CNOP'-ANWEISUNG MUSS '4' ODER '8' SEIN

ASS1251    L51 - SERIOUS ERROR
ASS1251    OPERAND 1 OF THE 'CNOP' INSTRUCTION MUST BE '0', '2', '4' OR '6'
ASS1251    OPERAND 1 DER 'CNOP'-ANWEISUNG MUSS '0', '2', '4' ODER '6' SEIN

ASS1252    L52 - SERIOUS ERROR
ASS1252    OPERAND 1 OF THE 'CNOP' INSTRUCTION MUST BE '0' OR '2'
ASS1252    OPERAND 1 DER 'CNOP'-ANWEISUNG MUSS '0' ODER '2' SEIN

ASS1253    L53 - SERIOUS ERROR
ASS1253    'CNOP' OPERAND (&00) IS RELOCATABLE; IT MUST BE ABSOLUTE
ASS1253    'CNOP'-OPERAND (&00) IST RELATIV, MUSS JEDOCH ABSOLUT SEIN
```

```
ASS1254    L54 - SIGNIFICANT ERROR
ASS1254    MISSING 'CNOP' OPERAND
ASS1254    'CNOP'-OPERAND FEHLT

ASS1310    M10 - SIGNIFICANT ERROR
ASS1310    SYMBOL (&00) IS MULTIPLE DEFINED
ASS1310    SYMBOL (&00) MEHRFACH DEFINIERT

ASS1320    M20 - SIGNIFICANT ERROR
ASS1320    'ENTRY' NOT ALLOWED IN 'DSECT', 'COM', 'XDSEC', OR 'DXD'
ASS1320    'ENTRY' IN 'DSECT','COM','XDSEC','DXD' UNZULAESSIG

ASS1321    M21 - SIGNIFICANT ERROR
ASS1321    ENTRY (&00) IS IN 'DSECT' OR 'XDSEC'
ASS1321    'ENTRY' (&00) LIEGT IN 'DSECT'/'XDSEC'

ASS1324    M24 - SIGNIFICANT ERROR
ASS1324    'AMODE'/'RMODE' INCONSISTENCY
ASS1324    'AMODE'/'RMODE'-UNVERTRAEGLICHKEIT

ASS1325    M25 - SIGNIFICANT ERROR
ASS1325    'AMODE'/'RMODE' IS ILLEGAL FOR AN UNNAMED 'COMMON' CONTROL SECTION
ASS1325    'AMODE'/'RMODE' FUER UNBENANNTEN 'COM'-ABSCHNITT UNZULAESSIG

ASS1330    M30 - SIGNIFICANT ERROR
ASS1330    ADDRESS OR DIRECT VALUE NOT WITHIN THE RANGE 0 TO 2**31 - 1
ASS1330    ADRESSE ODER DIREKTWERT NICHT IM BEREICH ZWISCHEN 0 UND 2**31 - 1

ASS1332    M32 - SIGNIFICANT ERROR
ASS1332    VALUE IN THE OPERAND EXCEEDS 2**24-1
ASS1332    OPERANDENWERT UEBERSCHREITET 2**24-1

ASS1350    M50 - SIGNIFICANT ERROR
ASS1350    SYMBOL (&00) ALREADY DEFINED AS A 'CSECT', 'START', 'DSECT', 'COM', 'XDSEC' OR
           'DXD' NAME
ASS1350    SYMBOL (&00) BEREITS ALS 'CSECT'-, 'START'-, 'DSECT'-, 'COM'-, 'XDSEC'- ODER
           'DXD'-NAME DEFINIERT

ASS1351    M51 - SIGNIFICANT ERROR
ASS1351    SYMBOL (&00) ALREADY DEFINED AS NAME OF 'EXTRN' OR 'WXTRN'
ASS1351    SYMBOL (&00) BEREITS ALS 'EXTRN'- ODER 'WXTRN'-NAME DEFINIERT

ASS1352    M52 - SIGNIFICANT ERROR
ASS1352    'AMODE'/'RMODE' ALREADY PRESENT
ASS1352    'AMODE'/'RMODE' BEREITS VORHANDEN
```

```
ASS1353    M53 - SIGNIFICANT ERROR
ASS1353    SYMBOL (&00) WAS ALREADY REFERENCED AS A Q-CONSTANT
ASS1353    SYMBOL (&00) BEREITS ALS Q-KONSTANTE REFERENZIERT
```

Meaning
The new symbol to be defined is ignored and can produce sequence errors in following references.

```
ASS1354    M54 - SIGNIFICANT ERROR
ASS1354    'XDSEC' ALREADY DEFINED. OPERAND DOES NOT EQUAL DEFINITION/REFERENCE
ASS1354    'XDSEC' BEREITS DEFINIERT. OPERAND NICHT GLEICH DEFINITION/REFERENZ

ASS1356    FAILURE
ASS1356    MISSING 'START' OR 'CSECT' INSTRUCTION
ASS1356    'START' BZW. 'CSECT'-ANWEISUNG FEHLT

ASS1357    M57 - SIGNIFICANT ERROR
ASS1357    'DSDD' INFORMATION ALREADY PRESENT
ASS1357    'DSDD'-INFORMATION BEREITS VORHANDEN

ASS1410    N10 - SIGNIFICANT ERROR
ASS1410    VARIABLE SYMBOL (&00) UNDEFINED AT TIME OF GENERATION
ASS1410    VARIABLER PARAMETER (&00) ZUM GENERIERUNGSZEITPUNKT UNDEFINIERT

ASS1420    N20 - SIGNIFICANT ERROR
ASS1420    UNDEFINED SEQUENCE SYMBOL IN OPERAND (&00): NO BRANCH
ASS1420    FOLGESYMBOL IN OPERAND (&00) UNDEFINIERT: KEIN SPRUNG

ASS1502    O02 - SIGNIFICANT ERROR
ASS1502    INVALID OPCODE
ASS1502    OPERATIONS-CODE UNGUELTIG

ASS1503    O03 - SIGNIFICANT ERROR
ASS1503    ERROR IN 'MACRO' OR MACRO PROTOTYPE STATEMENT: NO MACRO GENERATED.
ASS1503    FEHLER IN 'MACRO'- ODER MUSTERANWEISUNG: MAKRO WIRD NICHT GENERIERT
```

Meaning
Errors were encountered in the MACRO statement or in the opcode/operand field of the prototype statement of the called macro. The macro is not generated.

Response
Correct the MACRO and/or prototype statement.

```
ASS1504    O04 - SIGNIFICANT ERROR
ASS1504    MISSING OPCODE
ASS1504    OPERATIONS-CODE FEHLT
```

Meaning
The statement contains no operation code (possibly due to a missing blank before the opcode).

Response
Insert the required opcode (or blank).

```
ASS1505    O05 - SIGNIFICANT ERROR
ASS1505    OPCODE (&00) NOT FOUND
ASS1505    OPERATIONS-CODE (&00) NICHT GEFUNDEN

ASS1506    O06 - SIGNIFICANT ERROR
ASS1506    INVALID OPCODE IN OPERAND FIELD OF THE 'OPSYN' INSTRUCTION
ASS1506    OPERATIONS-CODE IN OPERANDENFELD DER 'OPSYN'-ANWEISUNG UNGUELTIG

ASS1522    O22 - SIGNIFICANT ERROR
ASS1522    SYMBOL (&00) CANNOT BE EVALUATED
ASS1522    SYMBOL (&00) NICHT BESTIMMBAR

ASS1601    P01 - WARNING
ASS1601    PRIVILEGED INSTRUCTION
ASS1601    PRIVILEGIERTER BEFEHL

ASS1711    Q11 - SIGNIFICANT ERROR
ASS1711    SYMBOL IN 'ORG' OPERAND DOES NOT BELONG TO THE CURRENT PROGRAM SECTION
ASS1711    SYMBOL IN 'ORG'-OPERAND LIEGT AUSSERHALB DES AKTUELLEN PROGRAMM-ABSCHNITTS

ASS1712    Q12 - SIGNIFICANT ERROR
ASS1712    OPERAND IS ABSOLUTE; MUST BE RELOCATABLE
ASS1712    OPERAND IST ABSOLUT; MUSS RELATIV SEIN

ASS1713    Q13 - SIGNIFICANT ERROR
ASS1713    VALUE IN 'ORG' OPERAND DOES NOT BELONG TO CURRENT PROGRAM SECTION
ASS1713    WERT IN 'ORG'-OPERAND LIEGT AUSSERHALB DES AKTUELLEN PROGRAMM-ABSCHNITTS

ASS1714    Q14 - SIGNIFICANT ERROR
ASS1714    SYMBOL IN ORG-OPERAND NOT PREVIOUSLY DEFINED
ASS1714    SYMBOL IM ORG-OPERAND NICHT VORHER DEFINIERT

ASS1721    Q21 - SIGNIFICANT ERROR
ASS1721    ILLEGAL SYMBOL REFERENCE IN OPERAND (&00)
ASS1721    SYMBOLZUGRIFF IN OPERAND (&00) FEHLERHAFT
```

```
ASS1901    S01 - SIGNIFICANT ERROR
ASS1901    SYMBOL TOO LONG; MAXIMUM LENGTH = 64
ASS1901    SYMBOL ZU LANG; MAXIMALLAENGE = 64
```

Meaning
The symbol in the name or operand field of the assembler instruction is too long.

Response
Shorten the symbol and repeat the run.

```
ASS1902    S02 - SIGNIFICANT ERROR
ASS1902    SEQUENCE SYMBOL NOT ALLOWED IN NAME FIELD
ASS1902    FOLGESYMBOL IM NAMENSFELD UNZULAESSIG
```

Meaning
A sequence symbol appears for the first time in the name field, but the specified sequence symbol is not legal for this opcode.

```
ASS1903    S03 - WARNING
ASS1903    NAME OF V OR Q CONSTANT TRUNCATED TO 8 CHARACTERS
ASS1903    NAME DER V- BZW. Q-KONSTANTE AUF 8 ZEICHEN GEKUERZT
```

Meaning
The name of a V-type or Q-type constant must not exceed 8 characters. Only the first 8 characters are taken into account.

Response
Shorten the name of the V-type or Q-type constant if required.

```
ASS1910    S10 - SIGNIFICANT ERROR
ASS1910    ILLEGAL NAME FIELD ENTRY
ASS1910    NAMENSFELD-EINTRAG UNZULAESSIG

ASS1911    S11 - SIGNIFICANT ERROR
ASS1911    ILLEGAL GENERATION IN NAME FIELD
ASS1911    GENERIERUNG IM NAMENSFELD UNZULAESSIG

ASS1912    S12 - SIGNIFICANT ERROR
ASS1912    'TITLE' INSTRUCTION WITH NAME FIELD ENTRY IS NOT FIRST 'TITLE' INSTRUCTION
ASS1912    'TITLE'-ANWEISUNG MIT NAMENSFELD-EINTRAG IST NICHT ERSTE 'TITLE'-ANWEISUNG

ASS1913    S13 - SIGNIFICANT ERROR
ASS1913    ILLEGAL GENERATION OF A SEQUENCE SYMBOL IN NAME FIELD
ASS1913    GENERIERUNG EINES FOLGESYMBOLS IM NAMENSFELD UNZULAESSIG
```

Meaning
The sequence symbol generated in the name field is ignored.

```
ASS1914    S14 - SIGNIFICANT ERROR
ASS1914    THE GENERATED NAME (&00) IS ILLEGAL; IT WILL BE IGNORED
ASS1914    GENERIERTER NAME (&00) UNGUELTIG; WIRD IGNORIERT

ASS1915    S15 - SIGNIFICANT ERROR
ASS1915    SEQUENCE SYMBOLS MUST NOT BE GENERATED IN THE NAME FIELD
ASS1915    GENERIEREN VON FOLGESYMBOLEN IM NAMENSFELD UNZULAESSIG

ASS1916    S16 - NOTE
ASS1916    'TITLE' NAME TRUNCATED TO 4 CHARACTERS
ASS1916    'TITLE'-NAME AUF 4 ZEICHEN GEKUERZT

ASS1917    S17 - NOTE
ASS1917    THE FIRST 'CSECT' IS UNNAMED; NO GENERATION OF 'AID'-INFORMATION
ASS1917    KEINE 'AID'-INFORMATION ERZEUGT, DA ERSTE 'CSECT' UNBENANNT
```

Meaning
The name of the first CSECT is provided as the source module name for AID.

Response
Name the first CSECT.

```
ASS2010    T10 - SIGNIFICANT ERROR
ASS2010    CHARACTER VALUE TOO LONG; MAXIMUM LENGTH = 1020
ASS2010    ZEICHENWERT ZU LANG; MAXIMALLAENGE = 1020

ASS2013    T13 - SIGNIFICANT ERROR
ASS2013    ILLEGAL AMPERSAND GENERATED AFTER TEXT REPLACEMENT
ASS2013    '&'-ZEICHEN NACH TEXTERSETZUNG UNZULAESSIG

ASS2014    T14 - SIGNIFICANT ERROR
ASS2014    ILLEGAL AMPERSAND GENERATED IN OPERAND (&00) AFTER TEXT REPLACEMENT
ASS2014    '&'-ZEICHEN NACH TEXTERSETZUNG IN OPERAND (&00) UNZULAESSIG
```

Meaning
Following text replacement, an ampersand (&) character, i.e. a new point of substitution, was generated in the named operand. This is illegal.

Response
Check the values of the variable symbols to be replaced.

```
ASS2015    T15 - SIGNIFICANT ERROR
ASS2015    ILLEGAL CONCATENATION
ASS2015    KONKATENIERUNG UNZULAESSIG

ASS2110    U10 - SIGNIFICANT ERROR
ASS2110    SYMBOL (&00) IS UNDEFINED
ASS2110    SYMBOL (&00) UNDEFINIERT
```

```
ASS2111   U11 - SIGNIFICANT ERROR
ASS2111   UNDEFINED 'ENTRY' NAME
ASS2111   'ENTRY'-NAME UNDEFINIERT
```

Response
Check the ENTRY name.

```
ASS2211   V11 - SIGNIFICANT ERROR
ASS2211   LOCAL VARIABLE SYMBOL IN OPERAND (&00) ALREADY DEFINED; FIRST DECLARATION IS
          VALID
ASS2211   LOKALER VARIABLER PARAMETER IN OPERAND (&00) BEREITS DEFINIERT; ERSTE
          DEKLARATION GILT
```

```
ASS2231   V31 - SIGNIFICANT ERROR
ASS2231   GLOBAL VARIABLE SYMBOL IN OPERAND (&00) ALREADY DEFINED. FIRST DECLARATION
          (&01) IS VALID
ASS2231   GLOBALER VARIABLER PARAMETER IN OPERAND (&00) BEREITS DEFINIERT; ERSTE
          DEKLARATION (&01) GILT
```

Meaning
(&01): STMT number of the first definition.

```
ASS2232   V32 - SIGNIFICANT ERROR
ASS2232   TYPE INCONSISTENCY BY USE OF THE GLOBAL VARIABLE SYMBOL IN THE NAME FIELD.
          (&00)
ASS2232   TYP-UNVERTRAEGLICHKEIT BEI VERWENDUNG DES GLOBALEN VARIABLEN PARAMETERS IM
          NAMENSFELD. (&00)
```

Meaning
(&00): STMT number of the first definition.

```
ASS2233   V33 - SIGNIFICANT ERROR
ASS2233   VARIABLE SYMBOL IN OPERAND (&00) OF THE 'GBL' INSTRUCTION IS ALREADY DEFINED AS
          A LOCAL VARIABLE SYMBOL
ASS2233   VARIABLER PARAMETER IN OPERAND (&00) DER 'GBL'-ANWEISUNG BEREITS ALS LOKALER
          VARIABLER PARAMETER DEKLARIERT
```

```
ASS2234   V34 - SIGNIFICANT ERROR
ASS2234   GLOBAL VARIABLE SYMBOL (&00) ALREADY DEFINED. (&01)
ASS2234   GLOBALER VARIABLER PARAMETER (&00) BEREITS DEFINIERT. (&01)
```

Meaning
(&01): STMT number of the first definition.

```
ASS2241    V41 - SIGNIFICANT ERROR
ASS2241    SYSTEM VARIABLES NOT ALLOWED IN PROTOTYPE STATEMENT OR IN THE OPERAND FIELD OF
           'LCL''GBL' INSTRUCTIONS
ASS2241    SYSTEMVARIABLE IN MUSTERANWEISUNG BZW. IM OPERANDENFELD VON 'LCL'-''GBL'-
           ANWEISUNGEN UNZULAESSIG

ASS2242    V42 - NOTE
ASS2242    SYSTEM VARIABLE SYMBOLS ILLEGAL IN NAME FIELD OF A PROTOTYPE STATEMENT
ASS2242    VARIABLE SYSTEMPARAMETER IM NAMENSFELD EINER MUSTERANWEISUNG UNZULAESSIG
```

Meaning
A system variable symbol appeared in the name field of a macro prototype statement. The name field is ignored.

```
ASS2244    V44 - SIGNIFICANT ERROR
ASS2244    ILLEGAL USE OF SYSTEM VARIABLE SYMBOL IN SOURCE
ASS2244    VERWENDUNG DES VARIABLEN SYSTEMPARAMETERS IN DER SOURCE UNZULAESSIG
```

Meaning
Some system variable symbols, e.g. &SYSVERM and &SYSECT, are macro-specific and may not be used in the source.

Response
Refer to the manual for information on the use of system variable symbols.

```
ASS2245    V45 - SIGNIFICANT ERROR
ASS2245    ILLEGAL REFERENCE TO THE SYSTEM VARIABLE SYMBOL '&SYSLIST'
ASS2245    ZUGRIFF AUF VARIABLEN SYSTEMPARAMETER '&SYSLIST' FEHLERHAFT
```

Meaning
No value is assigned to the system variable symbol &SYSLIST. Positional operands (or their sublist elements) can only be referred to via sublist references (&SYSLIST(x,y,...)).

Response
Only sublist references are permitted.

```
ASS2246    V46 - SIGNIFICANT ERROR
ASS2246    ILLEGAL VALUE ASSIGNMENT TO SYSTEM VARIABLE SYMBOL
ASS2246    ZUWEISUNG AN VARIABLEN SYSTEMPARAMETER UNZULAESSIG

ASS2247    V47 - SIGNIFICANT ERROR
ASS2247    OPERAND (&00) IS A SYSTEM VARIABLE SYMBOL
ASS2247    OPERAND (&00) IST SYSTEMPARAMETER
```

Meaning
It is illegal to declare system variable symbols as SET symbols.

```
ASS2248    V48 - SIGNIFICANT ERROR
ASS2248    ONLY THE VALUE ’24’ OR ’31’ IS PERMITTED FOR THE SYSTEM VARIABLE SYMBOL
           ’&SYSMOD’
ASS2248    FUER VARIABLEN SYSTEMPARAMETER ’&SYSMOD’ NUR WERT ’24’ ODER ’31’ ZULAESSIG

ASS2249    V49 - NOTE
ASS2249    OVERFLOW OF THE SYSTEM VARIABLE SYMBOL ’&SYSNDX’
ASS2249    UEBERLAUF DES VARIABLEN SYSTEMPARAMETERS ’&SYSNDX’
```

Meaning
This system variable symbol serves as a counter of variable symbols (maximum permissible value = 10000).

```
ASS2250    V50 - SIGNIFICANT ERROR
ASS2250    INVALID SYMBOLIC PARAMETER NAME
ASS2250    PARAMETERNAME FEHLERHAFT
```

Meaning
Possible error causes:
- The parameter name consists of only & characters
- The parameter name begins with a digit
- The parameter name contains illegal characters

```
ASS2251                V51 - WARNING
ASS2251                INVALID SUBLIST
ASS2251                UNGUELTIGE UNTERLISTE

ASS2252    V52 - SIGNIFICANT ERROR
ASS2252    SUBSCRIPTED GLOBAL VARIABLE SYMBOL (&00) IS REFERENCED WITHOUT SUBSCRIPT. (&01)
ASS2252    NICHT-INDIZIERTER ZUGRIFF AUF GLOBALEN INDIZIERTEN VARIABLEN PARAMETER (&00).
           (&01)
```

Meaning
Variable symbols declared with a subscript can only be referenced with subscripts.
(&01): STMT number of the first definition.

```
ASS2253    V53 - SIGNIFICANT ERROR
ASS2253    ILLEGAL SUBSCRIPT OR SUBLIST REFERENCE TO NONSUBSCRIPTED GLOBAL VARIABLE SYMBOL
           (&00). (&01)
ASS2253    INDEX- ODER UNTERLISTENZUGRIFF AUF GLOBALEN NICHT-INDIZIERTEN VARIABLEN
           PARAMETER (&00) UNZULAESSIG. (&01)
```

Meaning
Only subscripted variable symbols can be referenced by using subscripts.
(&01): STMT number of the first definition.

```
ASS2254   V54 - SIGNIFICANT ERROR
ASS2254   ILLEGAL SUBSCRIPT OR SUBLIST REFERENCE TO NONSUBSCRIPTED LOCAL VARIABLE SYMBOL
          (&00).
ASS2254   INDEX- ODER UNTERLISTENZUGRIFF AUF LOKALEN NICHT-INDIZIERTEN VARIABLEN
          PARAMETER (&00) UNZULAESSIG
```

Meaning
Only subscripted variable symbols can be referenced by using subscripts.

```
ASS2255   V55 - SIGNIFICANT ERROR
ASS2255   ILLEGAL SUBLIST REFERENCE TO THE LOCAL VARIABLE SYMBOL (&00)
ASS2255   UNTERLISTENZUGRIFF AUF LOKALEN VARIABLEN PARAMETER (&00) UNZULAESSIG
```

Meaning
Sublist references are only permitted for symbolic parameters.

```
ASS2256   V56 - SIGNIFICANT ERROR
ASS2256   INVALID '&SYSLIST' REFERENCE TO NAME FIELD. SUBSTITUTION VALUE: NAME FIELD
          ENTRY
ASS2256   '&SYSLIST'-ZUGRIFF AUF NAMENSFELD FEHLERHAFT. ERSATZWERT: NAMENSFELD-EINTRAG
```

Meaning
The name field entry can only be referenced via &SYSLIST(0).

```
ASS2257   V57 - SIGNIFICANT ERROR
ASS2257   ILLEGAL SUBLIST REFERENCE TO THE GLOBAL VARIABLE SYMBOL (&00). (&01)
ASS2257   UNTERLISTENZUGRIFF AUF GLOBALEN VARIABLEN PARAMETER (&00) UNZULAESSIG. (&01)
```

Meaning
Sublist references are only permitted for symbolic parameters.
(&01): STMT number of the first definition.

```
ASS2258   V58 - SIGNIFICANT ERROR
ASS2258   THE SUBSCRIPTED LOCAL VARIABLE SYMBOL (&00) CANNOT BE REFERENCED WITHOUT A
          SUBSCRIPT.
ASS2258   NICHT-INDIZIERTER ZUGRIFF AUF LOKALEN INDIZIERTEN VARIABLEN PARAMETER (&00)
          UNZULAESSIG
```

Meaning
A variable symbol declared with a subscript can only be referenced with a subscript.

```
ASS2259   V59 - NOTE
ASS2259   SYMBOLIC PARAMETERS AND 'SET' SYMBOLS MUST NOT BEGIN WITH 'SYS'
ASS2259   'SET'- UND SYMBOLISCHE PARAMETER DUERFEN NICHT MIT 'SYS' BEGINNEN
```

```
ASS2260   V60 - SIGNIFICANT ERROR
ASS2260   NAME FIELD CONTAINS ILLEGAL SET SYMBOL
ASS2260   'SET'-PARAMETER IM NAMENSFELD FEHLERHAFT
```

Meaning
Possible error causes:
The SET symbol specified in the name field has a syntax error or is a read-only system symbol

```
ASS2262   V62 - SIGNIFICANT ERROR
ASS2262   UNDEFINED VARIABLE SYMBOL IN NAME FIELD OF A 'SET' INSTRUCTION
ASS2262   VARIABLER PARAMETER IM NAMENSFELD DER 'SET'-ANWEISUNG UNDEFINIERT
```

Meaning
A variable symbol that was not defined or was defined twice appeared in the name field of the SET instruction.

Response
Possible responses:
- Provide a unique declaration for the SET symbol beforehand
- Use a SETx instruction with a type specification.

```
ASS2263   V63 - SIGNIFICANT ERROR
ASS2263   ILLEGAL VARIABLE SYMBOL OR 'SETC' EXPRESSION IN THE NAME FIELD
ASS2263   VARIABLER PARAMETER BZW. 'SETC'-AUSDRUCK IM NAMENSFELD UNZULAESSIG
```

```
ASS2264   V64 - SIGNIFICANT ERROR
ASS2264   GENERATED 'SET' SYMBOL IN THE NAME FIELD OF A 'SET' INSTRUCTION IS ILLEGAL
ASS2264   GENERIERTER 'SET'-PARAMETER IM NAMENSFELD EINER 'SET'-ANWEISUNG UNZULAESSIG
```

Meaning
A generated variable symbol appeared in the name field of a SET instruction. This is only permitted for SETx instructions with a type specification.

Response
Use SETx with a type specification instead of SET.

```
ASS2265   V65 - SIGNIFICANT ERROR
ASS2265   TYPE INCONSISTENCY BY USE OF THE LOCAL VARIABLE SYMBOL IN THE NAME FIELD
ASS2265   TYP-UNVERTRAEGLICHKEIT BEI VERWENDUNG DES LOKALEN VARIABLEN PARAMETERS IM
          NAMENSFELD
```

Meaning
The assigned value does not correspond to the declaration type of the variable symbol.

```
ASS2266    V66 - SIGNIFICANT ERROR
ASS2266    ILLEGAL IMPLICIT DEFINITION OF VARIABLE SYMBOL IN NAME FIELD OF A 'SET'
           INSTRUCTION
ASS2266    IMPLIZITE DEFINITION DES IM NAMENSFELD EINER 'SET'-ANWEISUNG STEHENDEN
           VARIABLEN PARAMETERS UNZULAESSIG
```

Meaning
An implicit definition is only permitted with SETA, SETB or SETC; however, not with SET.

```
ASS2267    V67 - SIGNIFICANT ERROR
ASS2267    WRONG NAME FIELD ENTRY IN CORRESPONDING PROTOTYPE STATEMENT
ASS2267    NAMENSFELDEINTRAG IN ZUGEHOERIGER MUSTERANWEISUNG FEHLERHAFT

ASS2268    V68 - SIGNIFICANT ERROR
ASS2268    VARIABLE SYMBOL (&00) TOO LONG; MAXIMUM VALUE = 64
ASS2268    VARIABLER PARAMETER (&00) ZU LANG; MAXIMALLAENGE = 64

ASS2269    V69 - SIGNIFICANT ERROR
ASS2269    SYNTAX ERROR IN VARIABLE SYMBOL (&00)
ASS2269    VARIABLER PARAMETER (&00) SYNTAKTISCH FALSCH

ASS2270    V70 - SIGNIFICANT ERROR
ASS2270    BLANKS ARE NOT ALLOWED AS PART OF VARIABLE OR SEQUENCE SYMBOLS
ASS2270    BLANKS ALS BESTANDTEIL VARIABLER PARAMETER BZW. FOLGESYMBOLE UNZULAESSIG

ASS2271    V71 - SIGNIFICANT ERROR
ASS2271    SYNTAX ERROR IN PARAMETER (&00) OF THE PROTOTYPE STATEMENT
ASS2271    SYNTAX-FEHLER IN OPERAND (&00) DER MUSTERANWEISUNG

ASS2273    V73 - SIGNIFICANT ERROR
ASS2273    ILLEGAL VALUE ASSIGNMENTS TO SUBLIST ELEMENTS
ASS2273    WERTZUWEISUNGEN AN UNTERLISTENELEMENTE UNZULAESSIG

ASS2274    V74 - SIGNIFICANT ERROR
ASS2274    ILLEGAL VALUE ASSIGNMENT TO THE SYSTEM VARIABLE SYMBOL '&SYSLIST'
ASS2274    WERTZUWEISUNG AN VARIABLEN SYSTEMPARAMETER '&SYSLIST' UNZULAESSIG
```

Meaning
No corresponding symbolic parameter was passed in the prototype statement, and no corresponding entry was made in the macro call.

```
ASS2410    X10 - SIGNIFICANT ERROR
ASS2410    VALUE OF THE SETB EXPRESSION IS NEITHER '0' NOR '1'
ASS2410    WERT DES 'SETB'-AUSDRUCKS WEDER '0' NOCH '1'
```

Meaning
A logical expression can only have the value 0 or 1.

```
ASS2412    X12 - WARNING
ASS2412    VALUE OF THE ARITHMETIC EXPRESSION IN THE AGO INSTRUCTION IS
           NEGATIVE, '0', OR GREATER THAN THE NUMBER OF SUPPLIED SEQUENCE SYMBOLS: NO
           BRANCH
ASS2412    WERT DES ARITHMETISCHEN AUSDRUCKS IN 'AGO'-ANWEISUNG NEGATIV,
           '0' ODER GROESSER ALS ANZAHL DER MITGEGEBENEN FOLGESYMBOLE: KEIN SPRUNG
```

```
ASS2413    X13 - SIGNIFICANT ERROR
ASS2413    'ACTR' OPERAND IS NEGATIVE
ASS2413    'ACTR'-OPERAND NEGATIV
```

Meaning
The operand of an ACTR instruction must be a positive arithmetic expression.

```
ASS2414    X14 - SIGNIFICANT ERROR
ASS2414    STRING IN OPERAND TOO LONG; MAXIMUM VALUE = 1020
ASS2414    ZEICHENSTRING IN OPERAND ZU LANG; MAXIMALLAENGE = 1020
```

```
ASS2415    X15 - SIGNIFICANT ERROR
ASS2415    ILLEGAL STRING OR NULL STRING IN DUPLICATION FACTOR OF 'SETC' OPERAND
ASS2415    ZEICHENSTRING ODER NULLSTRING IM WIEDERHOLUNGSFAKTOR IN 'SETC'-OPERAND
           UNZULAESSIG
```

```
ASS2416    X16 - SIGNIFICANT ERROR
ASS2416    ILLEGAL STRING OR NULL STRING AS ARGUMENT OF THE SUBSTRING FUNCTION
ASS2416    ZEICHENSTRING ODER NULLSTRING ALS ARGUMENT DER SUBSTRING-FUNKTION UNZULAESSIG
```

```
ASS2417    X17 - SIGNIFICANT ERROR
ASS2417    ARGUMENT OF THE SUBSTRING FUNCTION IS '0' OR NEGATIVE
ASS2417    ARGUMENT DER SUBSTRING-FUNKTION '0' ODER NEGATIV
```

```
ASS2419    X19 - SIGNIFICANT ERROR
ASS2419    NEGATIVE DUPLICATION FACTOR IN 'SETC' OPERAND
ASS2419    WIEDERHOLUNGSFAKTOR IN 'SETC'-OPERAND NEGATIV
```

```
ASS2420    X20 - SIGNIFICANT ERROR
ASS2420    ILLEGAL ATTRIBUTE REFERENCE
ASS2420    MERKMAL-BEZUG FEHLERHAFT
```

Meaning
Either no symbol or parameter follows the attribute, or the K or N attribute is followed by a symbol (only parameters are permitted).

```
ASS2421    X21 - SIGNIFICANT ERROR
ASS2421    ILLEGAL REFERENCE TO 'T' AND 'D' ATTRIBUTE IN A 'SETA' EXPRESSION
ASS2421    BEZUG AUF 'T'- UND 'D'-MERKMAL IM 'SETA'-AUSDRUCK UNZULAESSIG
```

```
ASS2422    X22 - SIGNIFICANT ERROR
ASS2422    ILLEGAL REFERENCE TO THE 'K' ATTRIBUTE FOR VARIABLE SYMBOL (&00)
ASS2422    BEZUG AUF 'K'-MERKMAL DES VARIABLEN PARAMETERS (&00) UNZULAESSIG
```

Meaning
A non-subscripted reference was made to the K attribute of a subscripted SET symbol or to &SYSLIST.

```
ASS2423    X23 - SIGNIFICANT ERROR
ASS2423    ILLEGAL SUBSCRIPTED REFERENCE TO THE 'N' ATTRIBUTE FOR VARIABLE SYMBOL (&00)
ASS2423    BEZUG AUF 'N'-MERKMAL DES VARIABLEN PARAMETERS (&00) BEI INDIZIERTEM ZUGRIFF
           UNZULAESSIG
```

Meaning
The N attribute for subscripted SET symbols is only defined for a non-subscripted reference.

```
ASS2424    X24 - WARNING
ASS2424    ATTRIBUTE REFERENCE NOT DEFINED; DEFAULT VALUE INSERTED
ASS2424    MERKMAL-BEZUG FEHLERHAFT; ERSATZWERT WIRD VERWENDET
```

Meaning
Either no symbol or parameter follows the attribute, or the symbol/parameter name contains a syntax error.

```
ASS2425    X25 - SIGNIFICANT ERROR
ASS2425    LENGTH OF THE SYMBOL (&00) CANNOT BE EVALUATED
ASS2425    LAENGE DES SYMBOLS (&00) NICHT BESTIMMBAR
```

```
ASS2427    X27 - SIGNIFICANT ERROR
ASS2427    ILLEGAL ATTRIBUTE REFERENCE: VALUE OF SYMBOLIC PARAMETER IS SYMBOL NAME
ASS2427    MERKMAL-BEZUG FEHLERHAFT: WERT DES SYMBOLISCHEN PARAMETERS IST SYMBOLNAME
```

```
ASS2433    X33 - SIGNIFICANT ERROR
ASS2433    SUBSCRIPT OF THE VARIABLE SYMBOL (&00) IS '0' OR NEGATIVE
ASS2433    INDEX DES VARIABLEN PARAMETERS (&00) '0' ODER NEGATIV
```

Meaning
The subscript must be > 0 for subscripted variable symbols.

Response
Replace the subscript by a value > 0.

```
ASS2434    X34 - SIGNIFICANT ERROR
ASS2434    ILLEGAL SUBLIST REFERENCE TO VARIABLE SYMBOL (&00) WITH SUBSCRIPT '0'
ASS2434    UNTERLISTENZUGRIFF AUF VARIABLEN PARAMETER (&00) MIT INDEX '0' UNZULAESSIG
```

Meaning
The subscript 0 is only permitted for reference to the name field &SYSLIST(0) zulaessig.

Response
Replace the subscript by a value > 0.

```
ASS2435    X35 - SIGNIFICANT ERROR
ASS2435    SUBSCRIPT OF THE VARIABLE SYMBOL (&00) IS '0' OR NEGATIVE
ASS2435    INDEX DES VARIABLEN PARAMETERS (&00) '0' ODER NEGATIV

ASS2436    X36 - SIGNIFICANT ERROR
ASS2436    SUBSCRIPTED USE OF PREDEFINED 'SET' SYMBOL IS ILLEGAL
ASS2436    INDIZIERTE VERWENDUNG VORDEFINIERTER 'SET'-PARAMETER UNZULAESSIG

ASS2437    X37 - WARNING
ASS2437    MULTIPLE ASSIGNMENT TO KEYWORD PARAMETER (&00); FIRST ASSIGNMENT IS VALID
ASS2437    MEHRFACH-ZUWEISUNG AN KENNWORTOPERANDEN (&00); ERSTE ZUWEISUNG GILT

ASS2439    X39 - SIGNIFICANT ERROR
ASS2439    SUBSCRIPTING ILLEGAL
ASS2439    INDIZIERUNG UNZULAESSIG

ASS2441    X41 - SIGNIFICANT ERROR
ASS2441    INVALID SEVERITY CODE (> 255) IN 'MNOTE' INSTRUCTION REPLACED BY ERROR CODE '0'
ASS2441    UNGUELTIGER FEHLERCODE (GROESSER 255) IN 'MNOTE'-ANWEISUNG DURCH FEHLERCODE '0'
           ERSETZT

ASS2448    X48 - SIGNIFICANT ERROR
ASS2448    ILLEGAL OPERAND FORMAT IN 'MNOTE' INSTRUCTION
ASS2448    OPERANDENFORMAT BEI 'MNOTE'-ANWEISUNG FEHLERHAFT
```

Meaning
The permissible operands for MNOTE are the severity code and the message string enclosed in single quotes.

```
ASS2449    X49 - SIGNIFICANT ERROR
ASS2449    OPERAND NOT ENCLOSED WITHIN SINGLE QUOTES
ASS2449    OPERAND NICHT IN HOCHKOMMATA EINGESCHLOSSEN
```

Meaning
The message text of an MNOTE or TITLE instruction is not enclosed within single quotes
(one or both quotes missing).

Response
Insert the missing single quote(s).

```
ASS2451    X51 - SIGNIFICANT ERROR
ASS2451    ONLY OPERAND 1 OF THE 'MNOTE' INSTRUCTION IS VALID
ASS2451    NUR OPERAND 1 BEI 'MNOTE'-ANWEISUNG GUELTIG
```

Meaning
The first operand in the MNOTE instruction is enclosed within single quotes; the following operands are ignored.

Response
Remove the excess operands or single quotes in the first operand.

```
ASS2452    X52 - SIGNIFICANT ERROR
ASS2452    INVALID SEVERITY CODE IN 'MNOTE'; THE INSTRUCTION WILL BE IGNORED
ASS2452    FEHLERCODE IN 'MNOTE'-ANWEISUNG UNGUELTIG; ANWEISUNG WIRD IGNORIERT

ASS2453    X53 - SIGNIFICANT ERROR
ASS2453    OPERANDS REQUIRED IN THE 'MNOTE' INSTRUCTION ARE MISSING OR EMPTY
ASS2453    ERFODERLICHE OPERANDEN IN 'MNOTE'-ANWEISUNG FEHLEN BZW. SIND LEER

ASS2454    X54 - WARNING
ASS2454    NO MORE THAN TWO OPERANDS ARE ALLOWED IN THE 'MNOTE' INSTRUCTION
ASS2454    MEHR ALS 2 OPERANDEN IN 'MNOTE'-ANWEISUNG UNZULAESSIG
```

Meaning
Excess operands are treated as comments.

```
ASS2455    X55 - SIGNIFICANT ERROR
ASS2455    CHARACTER EXPRESSION NOT ALLOWED IN ARITHMETIC OR LOGICAL EXPRESSION
ASS2455    ZEICHENAUSDRUCK IM ARITHMETISCHEN ODER LOGISCHEN AUSDRUCK UNZULAESSIG
```

Meaning
Character expressions are not permitted as operands in SETA or SETB expressions or in arithmetic or logical relations.

```
ASS2510    Y10 - SIGNIFICANT ERROR
ASS2510    REFERENCED ADDRESS NOT IN RANGE DEFINED BY 'USING' INSTRUCTION
ASS2510    ANGESPROCHENE ADRESSE AUSSERHALB DES DURCH 'USING'-ANWEISUNG ERFASSTEN BEREICHS
```

Meaning

Possible error causes:

- The specified address is not covered by a base register.
- The base register is dropped.

```
ASS2511    Y11 - WARNING
ASS2511    MISSING USING INSTRUCTION; BASE REGISTER '0' USED
ASS2511    'USING'-ANWEISUNG FEHLT, BASISREGISTER '0' WIRD VERWENDET
```

```
ASS2640    Z40 - SIGNIFICANT ERROR
ASS2640    'ACTR' EXCEEDED WHEN PROCESSING A MACRO
ASS2640    'ACTR'-UEBERLAUF BEI BEARBEITUNG EINES MAKROS
```

Meaning

The maximum number of AGO and AIF instructions was exceeded. This number is defined by the ACTR instruction; the default value is 4096. The macro expansion is terminated.

Response

Possible responses:

- Increment the ACTR counter with the ACTR instruction;
- Check the program for an endless loop.

```
ASS2641    Z41 - FAILURE
ASS2641    'ACTR' EXCEEDED WHEN PROCESSING MACRO INSTRUCTIONS IN THE SOURCE
ASS2641    'ACTR'-UEBERLAUF BEI BEARBEITUNG VON MAKROANWEISUNGEN IN DER SOURCE
```

Meaning

The maximum number of AGO and AIF instructions was exceeded. This number is defined by the ACTR instruction; the default value is 4096. The assembly is terminated.

Response

Possible responses:

- Increment the ACTR counter with the ACTR instruction;
- Check the program for an endless loop.

```
ASS2642    Z42 - SIGNIFICANT ERROR
ASS2642    PROGRAM COUNTER OVERFLOW
ASS2642    BEFEHLSZAEHLER UEBERLAUF
```

```
ASS6000    Z00 - FATAL ERROR
ASS6000    INTERNAL ERROR IN ASSEMBH: UNEXPECTED 'SPL4_RTS_GET_HEAP_RC' IN 'IARH850'
ASS6000    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER 'SPL4_RTS_GET_HAEP_RC' IN 'IARH850'
```

Response
Inform the system administrator.

```
ASS6001    FATAL ERROR
ASS6001    INTERNAL ERROR IN ASSEMBH: UNEXPECTED RETURN CODE: (&00) IN MACRO (&01) IN
           'IARH850'
ASS6001    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER RETURN-CODE: (&00) IN MAKRO (&01) IN
           'IARH850'
```

Response
Inform the system administrator.

```
ASS6002    Z02 - FAILURE
ASS6002    INTERNAL ERROR IN ASSEMBH: UNEXPECTED 'INSTRUCTION-SET' IN MODULE
           'IARH_OCTAB_COPY_700'
ASS6002    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER 'INSTRUCTION-SET' IM MODUL
           'IARH_OCTAB_COPY_700'
```

Meaning
Termination of the assembler due to system error.

Response
Inform the system administrator.

```
ASS6003    NO ERRORS
ASS6003    FILE CANNOT BE OPENED
ASS6003    DATEI KANN NICHT GEOEFFNET WERDEN

ASS6004    Z11 - FAILURE
ASS6004    OVERFLOW OF THE GENERATION BUFFER; MAX. SIZE IS 32K BYTE
ASS6004    MAXIMALE GROESSE DES GENERIERUNGSPUFFERS VON 32K BYTE UEBERSCHRITTEN

ASS6005    NO ERRORS
ASS6005    LISTING GENERATOR TIME FOR 'SAVLST'-CREATION: (&00) MSEC
ASS6005    ZEIT DES LISTEN-GENERATORS FUER 'SAVLST'-ERSTELLUNG: (&00) MSEC

ASS6006    NO ERRORS
ASS6006    LISTING GENERATOR TIME: (&00) MSEC
ASS6006    ZEIT DES LISTEN-GENERATORS: (&00) MSEC

ASS6007    NO ERRORS
ASS6007    TIME OF THE COMPONENT (&00): (&01) MSEC
ASS6007    ZEIT DER KOMPONENTE (&00): (&01) MSEC
```

```
ASS6008    NO ERRORS
ASS6008    ABNORMAL PROGRAM TERMINATION; ASSEMBH RETURN CODE: (&00)
ASS6008    ABNORMALE PROGRAMMBEENDIGUNG, ASSEMBH-RETURN-CODE: (&00)

ASS6009    NOTE
ASS6009    'MNOTE' WITH 'SEVERITY CODE' (&00)
ASS6009    'MNOTE' MIT 'SEVERITY CODE' (&00)
```

Meaning
Every line of the MNOTE-XREF begins with this text.
(&00): severity code 0 ... 255 of the 'MNOTE' instruction.

```
ASS6010    NO ERRORS
ASS6010    (&00) OF BS2000 ASSEMBH(&01) READY
ASS6010    (&00) DES BS2000 ASSEMBH(&01) READY

ASS6011    NO ERRORS
ASS6011    ASSEMBLY TIME: (&00) MSEC
ASS6011    ZEIT DER ASSEMBLIERUNG: (&00) MSEC

ASS6012    NO ERRORS
ASS6012    END OF ASSEMBH(&00)
ASS6012    ENDE ASSEMBH(&00)
```

Meaning
(&00): Functional scope of ASSEMBH (BC or XT).

```
ASS6013    Z13 - FAILURE
ASS6013    INTERNAL ERROR IN ASSEMBH: STXIT IN INSTRUCTION (&00). STXIT ACTIVATED
ASS6013    INTERNER FEHLER IM ASSEMBH: STXIT IN ANWEISUNG (&00). STXIT AKTIVIERT

ASS6014    Z14 - FATAL ERROR
ASS6014    FILE (&00) CANNOT BE CLOSED; RETURN CODE: (&01)
ASS6014    DATEI (&00) KANN NICHT GESCHLOSSEN WERDEN; RETURN-CODE: (&01)

ASS6017    Z17 - FATAL ERROR
ASS6017    INTERNAL ERROR IN ASSEMBH: FILE (&00) CANNOT BE OPENED; RETURN CODE = (&01)
ASS6017    INTERNER FEHLER IM ASSEMBH: OEFFNEN DER DATEI (&00) NICHT
           MOEGLICH; RETURN-CODE: (&01)
```

Meaning
This message is intended for the ASSEMBH development team.

Response
Inform the system administrator.

```
ASS6018    NO ERRORS
ASS6018    (&00) FLAGS, (&01) PRIVILEGED FLAGS, (&02) MNOTES
ASS6018    (&00) FLAGS, (&01) PRIVILEGED FLAGS, (&02) MNOTES
```

Meaning
Statistical information:
(&00): Total number of flags generated;
(&01): Total number of privileged flags generated;
(&02): Total number of macro notes generated.

```
ASS6019    NO ERRORS
ASS6019    HIGHEST ERROR-WEIGHT: (&00)
ASS6019    HIGHEST ERROR-WEIGHT: (&00)
```

```
ASS6020    Z20 - FAILURE
ASS6020    INTERNAL ERROR IN ASSEMBH: ILLEGAL RECORD TYPE IN THE LOCATION COUNTER BASE
           CHAIN (PSTAB)
ASS6020    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER SATZTYP IN DER ADRESSPEGEL-BASISKETTE
           (PSTAB)
```

Meaning
Termination of the assembler.

Response
Inform the system administrator.

```
ASS6021    Z21 - FAILURE
ASS6021    INTERNAL ERROR IN ASSEMBH: ILLEGAL OPCODE IN THE INTERMEDIATE LANGUAGE
ASS6021    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER OPERATIONS-CODE IN DER
           ZWISCHENSPRACHE
```

Meaning
Termination of the assembler.

Response
Inform the system administrator.

```
ASS6022    Z22 - FAILURE
ASS6022    INTERNAL ERROR IN ASSEMBH: UNEXPECTED 'FILE-TYPE' IN THE 'FILE-DESCRIPTOR'
ASS6022    INTERNER FEHLER IM ASSEMBH: UNERWARTETER 'FILE-TYPE' IM 'FILE-DESCRIPTOR'
```

Response
Inform the system administrator.

```
ASS6023   Z23 - FAILURE
ASS6023   INTERNAL ERROR IN ASSEMBH: ILLEGAL RETURN CODE OF THE PARSER
ASS6023   INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER RETURN-CODE DES ZERTEILERS
```

Response
Inform the system administrator.

```
ASS6024   Z24 - FAILURE
ASS6024   INTERNAL ERROR IN ASSEMBH: PARSER OVERFLOW
ASS6024   INTERNER FEHLER IM ASSEMBH: UEBERLAUF DES ZERTEILERS
```

Response
Inform the system administrator.

```
ASS6025   Z25 - FAILURE
ASS6025   INTERNAL ERROR IN ASSEMBH: ERROR DURING ACCESS TO THE PARSER TABLE
ASS6025   INTERNER FEHLER IM ASSEMBH: FEHLER BEIM ZUGRIFF AUF ZERTEILER-TABELLE
```

Response
Inform the system administrator.

```
ASS6026   Z26 - FAILURE
ASS6026   INTERNAL ERROR IN ASSEMBH: UNDERFLOW OF THE SEMANTIC STACK OF THE EXPRESSION
          ANALYSIS
ASS6026   INTERNER FEHLER IM ASSEMBH: UNTERLAUF DES SEMANTISCHEN STACKS DER
          AUSDRUCKSBEARBEITUNG
```

Response
Inform the system administrator.

```
ASS6029   FATAL ERROR
ASS6029   DMS ERROR (&00) WHEN OPENING THE SOURCE. IN SYSTEM MODE: /HELP-MSG DMS(&00)
ASS6029   DVS-FEHLER '(&00)' BEIM OEFFNEN DER SOURCE. IM SYSTEMMODUS: /HELP-MSG DMS(&00)
```

Meaning
When calling ASSEMBH as a subroutine, the source could not be opened. For more detailed information about the DMS error code, enter /HELP-MSG in system mode or see the manual 'BS2000 System Messages, Reference Manual' or one of the BS2000 DMS manuals.

Response
Specify the source correctly.

```
ASS6030   Z30 - FATAL ERROR
ASS6030   INTERNAL ERROR IN ASSEMBH: SYSDTA OPEN ERROR (&00)
ASS6030   INTERNER FEHLER IM ASSEMBH: SYSDTA OPEN ERROR (&00)
```

Response
Inform the system administrator.

```
ASS6031   Z31 - FATAL ERROR
ASS6031   INTERNAL ERROR IN ASSEMBH: OMF CLOSE ERROR (&00)
ASS6031   INTERNER FEHLER IM ASSEMBH: OMF CLOSE ERROR (&00)
```

Response
Inform the system administrator.

```
ASS6032   Z32 - FAILURE
ASS6032   PLAM-LIB OPEN ERROR (&00) WHEN WRITING THE OBJECT MODULE
ASS6032   PLAM-LIB OPEN FEHLER (&00) BEI OBJEKTMODUL-AUSGABE
```

Meaning
An error occurred when opening the PLAM library element.

```
ASS6033   Z33 - FATAL ERROR
ASS6033   EAM-OMF OPEN ERROR (&00)
ASS6033   EAM-OMF OPEN FEHLER (&00)
```

```
ASS6034   Z34 - FATAL ERROR
ASS6034   INTERNAL ASSEMBH ERROR. STREAM-STATUS-LIST IS NOT CORRECT (&00). STREAM COULD
          NOT BE OPENED
ASS6034   INTERNER FEHLER IM ASSEMBH: STROM-STATUS-LISTE NICHT KORREKT (&00). STROM
          KONNTE NICHT GEOEFFNET WERDEN
```

```
ASS6035   Z35 - FATAL ERROR
ASS6035   INTERNAL ERROR IN ASSEMBH: OPEN ERROR (&01) ON INPUT FILE (&00)
ASS6035   INTERNER FEHLER IM ASSEMBH: FEHLER (&01) BEIM EROEFFNEN DER EINGABEDATEI (&00)
```

Meaning
This message is intended for the ASSEMBH development team.

```
ASS6036   Z36 - FATAL ERROR
ASS6036   INTERNAL ERROR IN ASSEMBH: INPUT/OUTPUT NOT INITIALIZED
ASS6036   INTERNER FEHLER IM ASSEMBH: EIN-/AUSGABE NICHT INITIALISIERT
```

Response
Inform the system administrator.

```
ASS6037    Z37 - FATAL ERROR
ASS6037    INTERNAL ERROR IN ASSEMBH: SYSDTA CLOSE ERROR (&00)
ASS6037    INTERNER FEHLER IM ASSEMBH: SYSDTA CLOSE ERROR (&00)
```

Meaning
This message is intended for the ASSEMBH development team.

Response
Inform the system administrator.

```
ASS6038    Z38 - FATAL ERROR
ASS6038    INTERNAL ERROR IN ASSEMBH: WRONG FILE TYPE IN DATATAB
ASS6038    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER DATEITYP IN 'DATATAB'
```

Response
Inform the system administrator.

```
ASS6040    Z01 - FAILURE
ASS6040    INTERNAL ASSEMBH ERROR DURING TEXT REPLACEMENT
ASS6040    INTERNER FEHLER IM ASSEMBH: FEHLER BEI TEXTERSETZUNG
```

Response
Inform the system administrator.

```
ASS6041    Z10 - FAILURE
ASS6041    INTERNAL ERROR IN ASSEMBH: ILLEGAL SYMBOL TYPE FOR ENTRY PROCESSING
ASS6041    INTERNER FEHLER IM ASSEMBH: UNZULAESSIGER SYMBOLTYP IN ENTRY-BEARBEITUNG
```

Response
Inform the system administrator.

```
ASS6042    NO ERRORS
ASS6042    ELAPSED TIME: (&00) SEC
ASS6042    VERBRAUCHTE ZEIT: (&00) SEC

ASS6043    NO ERRORS
ASS6043    OPTION '*INCREMENT' FOR READING LIBRARY ACCESS NOT ALLOWED
ASS6043    OPTION '*INCREMENT' BEI LESENDEM BIBLIOTHEKSZUGRIFF UNZULAESSIG

ASS6044    NO ERRORS
ASS6044    OPTION '*INCREMENT' POSSIBLE ONLY WITH LMS/PLAM V2.0A
ASS6044    OPTION '*INCREMENT' ERST AB LMS/PLAM V2.0A MOEGLICH

ASS6045    NO ERRORS
ASS6045    OPTION '*HIGHEST-EXISTING' POSSIBLE ONLY WITH LMS/PLAM V2.0A
ASS6045    OPTION '*HIGHEST-EXISTING' ERST AB LMS/PLAM V2.0A MOEGLICH
```

```
ASS6050    Z50 - FAILURE
ASS6050    INTERNAL ERROR IN ASSEMBH: ERROR (&00) IN MODULE (&01)
ASS6050    INTERNER FEHLER IM ASSEMBH: FEHLER (&00) IM MODUL (&01)
```

Meaning
This message is intended for the ASSEMBH development team.

Response
Inform the system administrator.

```
ASS6051    FAILURE
ASS6051    INTERNAL ERROR IN ASSEMBH: INVALID 'SET' VALUE
ASS6051    INTERNER FEHLER IM ASSEMBH: 'SET'-WERT UNGUELTIG
```

Response
Inform the system administrator.

```
ASS6052    Z15 - FAILURE
ASS6052    INTERNAL ERROR IN ASSEMBH: ERROR (&00) IN MODULE (&01) IN INCLUDE (&02)
ASS6052    INTERNER FEHLER IM ASSEMBH: FEHLER (&00) IN MODUL (&01) IM INCLUDE (&02)
```

Meaning
This message is intended for the ASSEMBH development team.
(&02): Name of INCLUDE.

Response
Inform the system administrator.

```
ASS6060    FATAL ERROR
ASS6060    SDF SYNTAX FILE NOT CONTAINED IN CATALOG
ASS6060    SDF-SYNTAXDATEI IM KATALOG NICHT ENTHALTEN
```

Meaning
The SDF syntax file is not defined or not activated in BS2000.

Response
Inform the system administrator.

```
ASS6061  FATAL ERROR
ASS6061  'ASSEMBH' NOT DEFINED IN SDF SYNTAX FILE
ASS6061  'ASSEMBH' NICHT IN SDF-SYNTAXDATEI DEFINIERT
```

Meaning
The name ASSEMBH does not exist in the SDF syntax file.

Response
Inform the system administrator.

```
ASS6062  FAILURE
ASS6062  INTERNAL ERROR IN ASSEMBH: SDF INPUT BUFFER TOO SMALL
ASS6062  INTERNER FEHLER IM ASSEMBH: SDF-EINGABEPUFFER ZU KLEIN
```

Response
Inform the system administrator.

```
ASS6063  FATAL ERROR
ASS6063  SDF NOT LOADED IN BS2000
ASS6063  SDF IN BS2000 NICHT GELADEN
```

Response
Inform the system administrator.

```
ASS6064  FATAL ERROR
ASS6064  SDF SYSTEM ERROR; UNEXPECTED RETURN CODE: (&00)
ASS6064  SDF-SYSTEMFEHLER, UNERWARTETER RETURN-CODE: (&00)
```

Response
Inform the system administrator.

```
ASS6065  NOTE
ASS6065  INVALID VALUES FOR 'MARGINS'; DEFAULT VALUES ARE USED
ASS6065  'MARGINS'-WERTE UNGUELTIG; DURCH STANDARDWERTE ERSETZT
```

Response
Check the permissible values in the User Guide.

```
ASS6066  Z66 - NOTE
ASS6066  SOURCE OPEN ERROR: (&00)
ASS6066  FEHLER BEIM OEFFNEN DER SOURCE: (&00)
```

Meaning
(&00): cause of error.

```
ASS6070    Z70 - FAILURE
ASS6070    INTERNAL ERROR IN ASSEMBH: WRONG 'SYMTAB' ENTRY FOR THE SYMBOL (&00)
ASS6070    INTERNER FEHLER IM ASSEMBH: FEHLERHAFTER 'SYMTAB'-EINTRAG FUER SYMBOL (&00)
```

Response
Inform the system administrator.

```
ASS6071    Z71 - FAILURE
ASS6071    NO MEMBER NAME SPECIFIED FOR MODULE OUTPUT AND FIRST 'CSECT' UNNAMED; MODULE
           CANNOT BE OUTPUT
ASS6071    FUER MODULAUSGABE KEIN ELEMENTNAME ANGEGEBEN UND ERSTE 'CSECT' UNBENANNT; MODUL
           WIRD NICHT AUSGEGEBEN
```

Meaning
The element ("member") name cannot be determined, as the first CSECT is unnamed and the element name is omitted in the compiler options. The assembly is terminated.

```
ASS6072    Z72 - FAILURE
ASS6072    ERROR WHILE READING A SOURCE STATEMENT; RETURN CODE: (&00)
ASS6072    FEHLER BEIM LESEN DES SOURCE-STATEMENTS; RETURN-CODE: (&00)
```

```
ASS6073                Z73 - WARNING
ASS6073                ERROR ON OPENING THE MACRO/COPY LIBRARY (&00). LIBRARY
                       IGNORED. ORIGIN ERROR: (&01)
ASS6073                FEHLER BEIM OEFFNEN DER MAKRO-/COPY-BIBLIOTHEK (&00).
                       BIBLIOTHEK WIRD UEBERGANGEN.
           PRIMAERFEHLER: (&01)
```

Meaning
The specified macro or COPY library could not be opened.
(&00): library name, element name
(&01): cause of error.

```
ASS6074    Z74 - FAILURE
ASS6074    INTERNAL ERROR IN ASSEMBH: ERROR (&01) WHILE GENERATING A (&00) RECORD
ASS6074    INTERNER FEHLER IM ASSEMBH: FEHLER (&01) BEIM ERZEUGEN EINES (&00)-SATZES
```

Meaning
This message is intended for the ASSEMBH development team.

Response
Inform the system administrator.

```
ASS6075    Z75 - FAILURE
ASS6075    INSUFFICIENT MEMORY WHEN GENERATING THE INTERNAL TABLE (&00)
ASS6075    SPEICHERMANGEL BEIM ERZEUGEN DER INTERNEN TABELLE (&00)
```

Response
The allocated memory must be increased by system administration.

```
ASS6076    Z76 - NOTE
ASS6076    SOURCE FILE OR SOURCE MEMBER HAS WRONG TYPE: (&00)
ASS6076    SOURCE-FILE BZW. SOURCE-ELEMENT HAT FALSCHEN TYP: (&00)

ASS6080    NOTE
ASS6080    INTERNAL ASSEMBH ERROR IN OPTION TREATMENT. DEFAULT OPTION VALUES USED
ASS6080    INTERNER FEHLER IM ASSEMBH: FEHLER IN DER OPTIONS-VERARBEITUNG; STANDARD-
           OPTIONS-WERTE VERWENDET

ASS6081    NOTE
ASS6081    INVALID OPTION KEYWORD (&00); OPTION WILL BE IGNORED UNTIL NEXT '*' OR ',' IS
           DETECTED
ASS6081    OPTION-SCHLUESSELWORT (&00) UNGUELTIG; OPTION WIRD BIS ZUM NAECHSTEN '*' ODER
           ',' IGNORIERT
```

Meaning

An unidentifiable option keyword was specified after *COMOPT.

```
ASS6082    NOTE
ASS6082    ONLY '*' OR ',' IS ALLOWED AS A DELIMITER BETWEEN OPTIONS. THE OPTIONS WILL BE
           IGNORED UNTIL THEY ARE ENCOUNTERED
ASS6082    ALS TRENNUNGSZEICHEN ZWISCHEN OPTIONEN NUR '*' ODER ',' ZULAESSIG; BIS ZU DEREN
           AUFTRETEN WERDEN OPTIONEN IGNORIERT

ASS6083    NOTE
ASS6083    SYNTAX ERROR IN '*END' OPTION
ASS6083    SYNTAX-FEHLER IN '*END'-OPTION

ASS6084    NOTE
ASS6084    SYNTAX ERROR IN 'INSTR-SET' OPTION. DEFAULT VALUE IS USED
ASS6084    SYNTAX-FEHLER IN 'INSTR-SET'-OPTION. STANDARDWERT WIRD VERWENDET

ASS6085    NOTE
ASS6085    SYNTAX ERROR IN 'ADIAG' OPTION. 'NOADIAG' IS SET
ASS6085    SYNTAX-FEHLER IN 'ADIAG'-OPTION. 'NOADIAG' WIRD GESETZT

ASS6086    NOTE
ASS6086    SYNTAX ERROR IN A NO LONGER SUPPORTED OPTION
ASS6086    SYNTAX-FEHLER IN NICHT MEHR UNTERSTUETZTER OPTION

ASS6087    NOTE
ASS6087    SYNTAX ERROR IN THE 'SYSPARM' OPTION. THE NULL STRING IS ASSIGNED TO '&SYSPARM'
ASS6087    SYNTAX-FEHLER IN DER 'SYSPARM'-OPTION. '&SYSPARM' WIRD NULLSTRING ZUGEWIESEN
```

```
ASS6088    NOTE
ASS6088    SYNTAX ERROR IN THE 'ERR' OPTION. DEFAULT VALUES ARE USED
ASS6088    SYNTAX-FEHLER IN 'ERR'-OPTION. STANDARDWERTE WERDEN VERWENDET

ASS6089    NOTE
ASS6089    SYNTAX ERROR IN 'ERRPR' OPTION. DEFAULT VALUE IS USED
ASS6089    SYNTAX-FEHLER IN 'ERRPR'-OPTION. STANDARDWERT WIRD VERWENDET

ASS6090    NOTE
ASS6090    SYNTAX ERROR IN 'LINECNT' OPTION. DEFAULT VALUE IS USED
ASS6090    SYNTAX-FEHLER IN 'LINECNT'-OPTION. STANDARDWERT WIRD VERWENDET

ASS6091    NOTE
ASS6091    SYNTAX ERROR IN 'PRTOFF' OPTION. DEFAULT VALUE IS USED
ASS6091    SYNTAX-FEHLER IN 'PRTOFF'-OPTION. STANDARDWERT WIRD VERWENDET

ASS6092    NOTE
ASS6092    SYNTAX ERROR IN 'SOURCE' OPTION. 'SOURCE=*' IS SET
ASS6092    SYNTAX-FEHLER IN 'SOURCE'-OPTION. 'SOURCE=*' WIRD GESETZT

ASS6093    NOTE
ASS6093    SYNTAX ERROR IN 'MODULE' OPTION. 'MODULE=*' IS SET
ASS6093    SYNTAX-FEHLER IN 'MODULE'-OPTION. 'MODULE=*' WIRD GESETZT

ASS6094    NOTE
ASS6094    ILLEGAL USE OF THE 'DUET' OPTION TOGETHER WITH 'INSTR=SET2' OR 'INSTR=SET3';
           THE 'DUET' OPTION WILL BE IGNORED
ASS6094    'DUET'-OPTION KOMBINIERT MIT 'INSTR=SET2'- BZW. 'INSTR=SET3' -OPTION
           UNZULAESSIG; 'DUET'-OPTION WIRD IGNORIERT

ASS6095    NOTE
ASS6095    THE 'ISD' AND 'ADIAG' OPTION IS NOT SUPPORTED IN ASSEMBH-BC.
ASS6095    'ISD'- UND 'ADIAG'-OPTION IM ASSEMBH-BC NICHT UNTERSTUETZT

ASS6096    NOTE
ASS6096    INVALID STRING LENGTH IN 'SYSPARM' OPTION; THE NULL STRING IS ASSIGNED TO
           '&SYSPARAM'
ASS6096    STRING-LAENGE BEI 'SYSPARM'-OPTION UNGUELTIG; '&SYSPARM' WIRD NULLSTRING
           ZUGEWIESEN

ASS6097    NOTE
ASS6097    INVALID VALUE IN 'ERRPR' OPTION; DEFAULT VALUE IS USED
ASS6097    WERT IN 'ERRPR'-OPTION UNGUELTIG; STANDARDWERT WIRD VERWENDET
```

```
ASS6098    NOTE
ASS6098    INVALID VALUE IN 'LINECNT' OPTION; DEFAULT VALUE IS USED
ASS6098    WERT IN 'LINECNT'-OPTION UNGUELTIG; STANDARDWERT WIRD VERWENDET

ASS6099    NOTE
ASS6099    INVALID VALUE IN THE OPTION 'ERR=N'; DEFAULT VALUE IS USED
ASS6099    WERT IN 'ERR=N'-OPTION UNGUELTIG; STANDARDWERT WIRD VERWENDET

ASS6100    NOTE
ASS6100    OPTION (&00) IS NO LONGER SUPPORTED IN ASSEMBH
ASS6100    OPTION (&00) IN ASSEMBH NICHT MEHR UNTERSTUETZT

ASS6101    NOTE
ASS6101    INVALID VALUE IN THE OPTION 'PRTOFF=N'; DEFAULT VALUE IS USED
ASS6101    WERT IN 'PRTOFF=N'-OPTION UNGUELTIG; STANDARDWERT WIRD VERWENDET

ASS6102    NOTE
ASS6102    IN THE OPTION 'PRTOFF=X1;X2...'; A INVALID CHARACTER IS GIVEN. IT WILL BE
           IGNORED
ASS6102    ZEICHEN IN 'PRTOFF=X1;X2...'-OPTION UNGUELTIG; ZEICHEN WIRD IGNORIERT

ASS6103    NOTE
ASS6103    ONLY ONE CHARACTER IS ALLOWED PER ENTRY IN THE 'PRTOFF=X1;X2...' OPTION; STRING
           ENTRIES ARE IGNORED
ASS6103    IN 'PRTOFF=X1;X2...'-OPTION JEWEILS NUR EIN ZEICHEN ZULAESSIG; STRING-ANGABE
           WIRD IGNORIERT

ASS6104    NOTE
ASS6104    LENGTH OF FILE NAME IN 'SOURCE' OPTION IS INVALID. 'SOURCE=*' IS SET
ASS6104    LAENGE DES DATEINAMENS IN 'SOURCE'-OPTION UNZULAESSIG; 'SOURCE=*' WIRD GESETZT

ASS6105    NOTE
ASS6105    LENGTH OF MEMBER NAME IN 'SOURCE' OPTION IS INVALID. 'SOURCE=*' IS SET
ASS6105    LAENGE DES ELEMENTNAMENS IN 'SOURCE'-OPTION UNZULAESSIG. 'SOURCE=*' WIRD
           GESETZT

ASS6106    NOTE
ASS6106    LENGTH OF VERSION IN 'SOURCE' OPTION IS INVALID. 'SOURCE=*' IS SET
ASS6106    LAENGE DER VERSIONSANGABE IN 'SOURCE'-OPTION UNZULAESSIG; 'SOURCE=*' WIRD
           GESETZT

ASS6107    NOTE
ASS6107    LENGTH OF LIBRARY NAME IN 'MODULE' OPTION IS INVALID. 'MODULE=*' IS SET
ASS6107    LAENGE DES BIBLIOTHEKNAMENS IN 'MODULE'-OPTION UNZULAESSIG; 'MODULE=*' WIRD
           GESETZT
```

```
ASS6108    NOTE
ASS6108    LENGTH OF MEMBER NAME IN 'MODULE' OPTION IS INVALID. 'MODULE=*' IS SET
ASS6108    LAENGE DES ELEMENTNAMENS IN 'MODULE'-OPTION UNZULAESSIG; 'MODULE=*' WIRD
           GESETZT

ASS6109    NOTE
ASS6109    LENGTH OF VERSION IN 'MODULE' OPTION IS INVALID. 'MODULE=*' IS SET
ASS6109    LAENGE DER VERSIONSANGABE IN 'MODULE'-OPTION UNZULAESSIG; 'MODULE=*' WIRD
           GESETZT

ASS6110    FATAL - ERROR
ASS6110    SOURCE CANNOT BE OPENED; (&00); 'HALT' IS SET
ASS6110    SOURCE KANN NICHT GEOEFFNET WERDEN; (&00); 'HALT' WIRD GESETZT
```

Meaning
(&00): cause of error (e.g. FILE NOT SHAREABLE).

```
ASS6111    NOTE
ASS6111    ONLY 'COMOPT', 'END', OR 'HALT' IS ALLOWED AFTER '*'; ALL OTHER ENTRIES ARE
           IGNORED
ASS6111    NACH '*' NUR 'COMOPT', 'END' ODER 'HALT' ZULAESSIG; ALLES ANDERE WIRD IGNORIERT

ASS6112    NOTE
ASS6112    UNEXPECTED EOF; 'END HALT' IS SET
ASS6112    UNERWARTETES EOF; 'END HALT' WURDE GESETZT

ASS6113    NOTE
ASS6113    UNEXPECTED EOF; 'HALT' IS SET
ASS6113    UNERWARTETES EOF; 'HALT' WURDE GESETZT

ASS6114    NOTE
ASS6114    LAST QUOTE IS MISSING IN THE 'SYSPARM' OPTION. THE NULL STRING IS ASSIGNED TO
           '&SYSPARM'
ASS6114    IN 'SYSPARM'-OPTION FEHLT ABSCHLIESSENDES APOSTROPH; '&SYSPARM' WIRD NULLSTRING
           ZUGEWIESEN

ASS6115    NOTE
ASS6115    THE 'DSDD' OR 'MONSYS RECORDS=YES' OPTION IS ONLY ALLOWED WHEN MODULE IS OUTPUT
           TO A PLAM LIBRARY; OPTION WILL BE IGNORED
ASS6115    'DSDD'- BZW. 'MONSYS-RECORDS=YES'-OPTION NUR BEI MODULAUSGABE IN PLAM-
           BIBLIOTHEK ZULAESSIG; OPTION WIRD IGNORIERT
```

```
ASS6117  NOTE
ASS6117  SYNTAX ERROR IN 'SEQ' OPTION; 'SEQ' OPTION IS IGNORED
ASS6117  SYNTAXFEHLER IN DER 'SEQ' OPTION; 'SEQ' OPTION WIRD IGNORIERT
```

Meaning
'SEQ' option doesn't have the form 'SEQ=(<number>[,<length>[,<id>]])' with 4<= <length> <=8 and <id> <=4 characters and length from <id>+<length> <=8.

```
ASS6121  Z19 - WARNING
ASS6121  INTERNAL ERROR IN ASSEMBH: WARNING BY 'CIF' ACCESS ROUTINE; RETURN CODE: (&00)
ASS6121  INTERNER FEHLER IM ASSEMBH: WARNUNG DURCH 'CIF'-ZUGRIFFSROUTINE, RETURN-CODE:
         (&00)
```

Meaning
This message is intended for the ASSEMBH development team.

```
ASS6122  FAILURE
ASS6122  INTERNAL ERROR IN ASSEMBH: ERROR IN 'CIF' ACCESS ROUTINE
ASS6122  INTERNER FEHLER IM ASSEMBH: FEHLER IN 'CIF'-ZUGRIFFSROUTINE
```

Response
Inform the system administrator.

```
ASS6123  SERIOUS ERROR
ASS6123  INTERNAL ERROR IN ASSEMBH: ERROR BY 'CIF' ACCESS ROUTINE; RETURN CODE: (&00)
ASS6123  INTERNER FEHLER IM ASSEMBH: FEHLER DURCH 'CIF'-ZUGRIFFSROUTINE, RETURN-CODE:
         (&00)
```

Meaning
This message is intended for the ASSEMBH development team.

Response
Inform the system administrator.

```
ASS6124  SERIOUS ERROR
ASS6124  CC-DMS ERROR (&00) IN 'CIF' ACCESS ROUTINE
ASS6124  CC-DMS-FEHLER (&00) IN 'CIF'-ZUGRIFFSROUTINE

ASS6125  FAILURE
ASS6125  INTERNAL ERROR IN ASSEMBH: 'PIOM' TERMINATION CAUSED BY WRONG 'CIF'
ASS6125  INTERNER FEHLER IM ASSEMBH: 'PIOM'-ABBRUCH VERURSACHT DURCH FEHLERHAFTES 'CIF'
```

Response
Inform the system administrator.

```
ASS6126    FAILURE
ASS6126    INTERNAL ERROR IN ASSEMBH: 'PIOM' TERMINATION CAUSED BY INCOMPATIBLE VERSIONS
           OF 'CIF' ACCESS ROUTINES
ASS6126    INTERNER FEHLER IM ASSEMBH: 'PIOM'-ABBRUCH VERURSACHT DURCH INKOMPATIBLE
           VERSIONEN DER 'CIF'-ZUGRIFFSROUTINEN
```

Response
Inform the system administrator.

```
ASS6127    Z18 - FAILURE
ASS6127    INSUFFICIENT MEMORY FOR VIRTUAL CIF
ASS6127    NICHT AUSREICHEND SPEICHER FUER VIRTUELLEN 'CIF' VORHANDEN
```

Response
The allocated memory must be increased by system administration.

```
ASS6128    SERIOUS ERROR
ASS6128    "COMPILER INFORMATION FILE" IS NO PLAM LIBRARY
ASS6128    "COMPILER INFORMATION FILE" IST KEINE PLAM=BIBLIOTHEK
```

Meaning
No listing is generated.

Response
Specify a PLAM library for "Compiler Information File".

```
ASS6129    SERIOUS ERROR
ASS6129    THE PLAM LIBRARY MEMBER FOR "COMPILER INFORMATION FILE" IS LOCKED
ASS6129    PLAM-BIBLIOTHEKSELEMENT FUER "COMPILER INFORMATION FILE" IST GESPERRT
```

Meaning
No listing is generated.

Response
Unlock the PLAM library member for "Compiler Information File".

```
ASS6132    Z29  - FAILURE
ASS6132    COMPILATION CANCELLED DUE TO TERMINATION CONDITION
ASS6132    ABBRUCHKRITERIUM ERREICHT, UEBERSETZUNG ABGEBROCHEN
```

Meaning

Possible termination condition:
- maximum number of errors exceeded
- maximum error weight reached
- maximum nest level of MACRO or COPY exceeded
- ACTR overflow

Response

Correct the source or increase the limits using COMPILER-TERMINATION option or ACTR instruction.

```
ASS6140    FAILURE
ASS6140    INTERNAL ERROR IN ASSEMBH: ERROR IN LISTING GENERATION.
           TERMINATION OF THE ASSEMBH RUN WITH LG RETURN CODE: (&00)
ASS6140    INTERNER FEHLER IM ASSEMBH: FEHLER BEI DER LISTING-ERSTELLUNG.
           ABBRUCH DES ASSEMBH-LAUFS MIT LG-RETURNCODE: (&00)
```

Meaning

This message is intended for the ASSEMBH development team.
(&00): Listing Generator return code.

Response

Inform the system administrator.

```
ASS6141    SIGNIFICANT ERROR
ASS6141    INTERNAL ERROR IN ASSEMBH: ERROR IN LISTING GENERATION. INCOMPLETE OR WRONG
           LISTING WAS GENERATED
ASS6141    INTERNER FEHLER IM ASSEMBH: FEHLER BEI DER LISTING-ERSTELLUNG. UNVOLLSTAENDIGES
           ODER FEHLERHAFTES LISTING WURDE ERZEUGT

ASS6142    NOTE
ASS6142    'AID' IS NOT SUPPORTED IN ASSEMBH-BC
ASS6142    'AID' IM ASSEMBH-BC NICHT UNTERSTUETZT

ASS6143    NO ERRORS
ASS6143    ASSDIAG COMMAND 'RERUN' AFTER ABORT OF ASSEMBH NOT ALLOWED; 'END' COMMAND
           ASSUMED
ASS6143    ASSDIAG-KOMMANDO 'RERUN' NACH ASSEMBH-ABBRUCH UNZULAESSIG; 'END'-KOMMANDO WIRD
           AUSGEFUEHRT

ASS6144    NO ERRORS
ASS6144    ERRORFILE GENERATION TIME: (&00) MSEC
ASS6144    ZEIT FUER ERRORFILE ERSTELLUNG: (&00) MSEC
```

```
ASS6145    FAILURE
ASS6145    ILLEGAL VERSION OF SYNTAXFILE FOR ASSEMBH
ASS6145    VERKEHRTE VERSION DES SYNTAX-FILES FUER DEN ASSEMBH
```

Response

Inform the system administrator so that he/she can install the correct syntax file.

```
ASS6146    NOTE
ASS6146    UNEXPECTED EOF; '//END' IS SET
ASS6146    UNERWARTETES EOF; '//END' WURDE GESETZT
```

11.1.1 Messages of the assembler runtime system for structured programming

```
ASS7001  INITIALIZATION OF THE ASSEMBLER RUNTIME SYSTEM NOT POSSIBLE
ASS7001  INITIALISIERUNG DES ASSEMBLER-LAUFZEITSYSTEMS NICHT MOEGLICH
```

Meaning
Due to memory constraints, the INITIAL STACK cannot be set up for the main procedure during initialization of the runtime system.

Response
Inform the system administrator (increase the user address space).

```
ASS7002  FATAL ERROR
ASS7002  INSUFFICIENT MEMORY FOR THE 'INITIAL-STACK'
ASS7002  SPEICHERMANGEL BEI BESCHAFFUNG DES 'INITIAL-STACKS'
```

Meaning
The administrative data area cannot be set up during initialization of the runtime system.

Response
Inform the system administrator.

```
ASS7003  FATAL ERROR
ASS7003  INSUFFICIENT MEMORY TO INITIALIZE THE 'STACK' AS SPECIFIED BY 'STACK'-PARAMETER
         OF THE @ENTR-MACRO
ASS7003  SPEICHERMANGEL BEI BESCHAFFUNG DES 'STACK' GEMAESS 'STACK'-ANGABE IM '@ENTR'-
         MAKRO
```

Meaning
While initializing the runtime system, the STACK cannot be set up for the main procedure as required by the user or with the default value.

Response
Possible responses:
- Reduce the STACK requirement;
- Ask the system administrator to increase the user address space.

ASS7005 STACK-POINTER DESTROYED; STACK-REGISTER 13 CONTAINS INVALID VALUE.
ASS7005 STACK-ZEIGER ZERSTOERT; STACK-REGISTER 13 ENTHAELT FEHLERHAFTEN WERT.

Meaning
On commencing initialization of the runtime system from external procedures (FORTRAN, COBOL, ASSEMBLER) or in the procedure prologue, the STACK register does not point to a valid SAVE AREA.

Response
Possible responses:
– Load the STACK register correctly prior to initialization;
– Do not change the STACK register within the nested procedure.

ASS7006 NO MORE MEMORY AVAILABLE FOR THE 'STACK'
ASS7006 WEITERER SPEICHERPLATZ FUER 'STACK' NICHT VERFUEGBAR

Meaning
Due to memory constraints, the SAVE-AREA or the area for LOCAL data cannot be initialized in the procedure prologue.

Response
Possible responses:
– Release occupied memory;
– Ask the system administrator to increase the user address space.

ASS7007 NO MORE MEMORY AVAILABLE FOR THE 'AUTOMATIC' AREA
ASS7007 WEITERER SPEICHERPLATZ FUER 'AUTOMATIC'-BEREICH NICHT VERFUEGBAR

Meaning
No STACK memory is available for a service request of class AUTOMATIC.

Response
Possible responses:
– Reduce the service request(s);
– Ask the system administrator to increase the user address space.

```
ASS7008   NO MORE MEMORY AVAILABLE FOR THE 'CONTROLLED' AREA
ASS7008   WEITERER SPEICHERPLATZ FUER 'CONTROLLED'-BEREICH NICHT VERFUEGBAR
```

Meaning
No HEAP memory is available for a service request of class CONTROLLED.

Response
Possible responses:
- Reduce the service request(s);
- Release the HEAP memory not in use;
- Ask the system administrator to increase the user address space.

```
ASS7009   FATAL ERROR
ASS7009   ERROR IN RELEASING MEMORY OF THE 'CONTROLLED' AREA
ASS7009   FEHLER BEI FREIGABE EINES 'CONTROLLED'-BEREICHS
```

Meaning
The specified address does not point to an allocated memory area in the HEAP.

Response
Specify the correct address.

```
ASS7010   WARNING
ASS7010   INITIALIZATION ROUTINE 'IASSIN' WAS ALREADY CALLED
ASS7010   INITIALISIERUNGS-ROUTINE 'IASSIN' WURDE BEREITS AUFGERUFEN
```

Meaning
Multiple calls to initialize the runtime system from external procedures (FORTRAN, COBOL, ASSEMBLER).

Response
Avoid multiple initializations.

```
ASS7011   INCONSISTENT AID-VERSION
ASS7011   INKONSISTENTE AID-VERSION
```

Bedeutung
Fehler während der Initialisierung des Laufzeitsystems, da im System eine inkonsistente AID-Version installiert ist.

11.1.2 Listing generator messages

LGR0001 'CIF' ALREADY OPEN

LGR0002 INFORMATION TABLES CLOSED IMPLICITLY

LGR0003 DUPLICATE KEYS EXIST

LGR0004 END OF PARTITION

LGR0005 INFORMATION TABLE CLOSED ABNORMALLY

LGR0006 PARTITION CLOSED IMPLICITLY

LGR0007 'CIF' ALREADY CLOSED

LGR0101 'CIF' COULD NOT BE OPENED

LGR0102 'CIF' CURRENTLY LOCKED

LGR0103 SPECIFIED 'CIF' DOES NOT EXIST

LGR0104 'CIF' NOT A LIBRARY

LGR0105 'CIF' IDENTIFIER INVALID

LGR0106 MAXIMUM NUMBER OF 'CIF'S EXCEEDED

LGR0107 OPEN MODE ILLEGAL

LGR0108 ENVIRONMENT UNSUITABLE

LGR0109 ACCESS TO INFORMATION TABLE NOT PERMITTED

LGR0110 'IT' NAME INVALID

LGR0111 CLOSE MODE ILLEGAL

LGR0112 SORT ORDER NOT ASCENDING

LGR0113 PARTITION INVALID

LGR0114 SPECIFIED KEY NOT FOUND

LGR0115 FUNCTION NOT SUPPORTED

LGR0116 WRITE ACCESS ILLEGAL

LGR0117 FIELD LENGTH INVALID

LGR0118 SPACE OVERFLOW DURING ALLOCATION

Meaning
During internal space allocation a space overflow occurred

Response
Please erase unnecessary space or increase the space allowance and start the program again

LGR0119 CC-DMS ERROR (&00) WHEN ACCESSING CIF

Meaning
For more detailed information about the DMS error code enter /HELP-MSG in system mode or see the BS2000 manual 'System Messages'

LGR0120 INFORMATION TABLE DOES NOT EXIST

LGR0121 INFORMATION TABLE ALREADY EXISTS

LGR0122 READ ACCESS ILLEGAL

LGR0123 MANDATORY FIELD MISSING

LGR0124 'CIF' TYPE INVALID

LGR0125 ENVIRONMENT ILLEGAL

LGR0126 INFORMATION TABLE NOT OPENED

LGR0201 INCOMPATIBLE VERSION IDENTIFIERS

LGR0202 'CIF' DESTROYED

LGR0203 INTERNAL ERROR: CHECK RETURN CODE

Meaning
message for ASSLG development team: ret_code (stat): INTERNAL_ERROR

Response
contact the system administrator

LGR0299 INTERNAL ERROR: RETURN CODE UNKNOWN

Meaning
message for ASSLG development team: ret_code (stat): DEFAULT

Response
contact the system administrator

LGR0301 FILE IS LIBRARY

Meaning
The medium for the assembler listing is wrong

Response
Correct the LISTING option and start again

LGR0302 FILE IS PLAM LIBRARY

Meaning
The medium for the assembler listing is wrong

Response
Correct the LISTING option and start again

LGR0303 UNEXPECTED 'EOF' DETECTED

LGR0304 FILE NOT A LIBRARY

Meaning
The medium for the assembler listing is wrong

Response
Correct the LISTING option and start again

LGR0305 FILE AN OSM LIBRARY

Meaning
The medium for the assembler listing is wrong

Response
Correct the LISTING Option and start again

LGR0306 FCB TYPE INVALID

LGR0307 NO FCB TYPE SPECIFIED

LGR0308 WRITE NOT ALLOWED IN OSM LIBRARIES

LGR0309 FILE IS AN UNKNOWN LIBRARY

LGR0310 FILE EMPTY

LGR0311 FILE NOT CATALOGED

LGR0312 NO LINK OR FILE NAME FOUND

LGR0313 LIBRARY MEMBER NOT FOUND

LGR0314 FILE LOCKED

LGR0315 FILE NOT SHAREABLE

LGR0316 PASSWORD MISSING

LGR0317 TYPE OF LIBRARY MEMBER INVALID

LGR0318 NAME OF LIBRARY MEMBER INVALID

LGR0319 VERSION OF LIBRARY MEMBER INVALID

LGR0320 MEMORY SPACE SATURATION

LGR0321 LIBRARY MEMBER LOCKED

LGR0322 VARIANT OF LIBRARY MEMBER NOT FOUND

LGR0323 PLAM NOT LOADED IN SYSTEM

LGR0324 FILE NAME INVALID

LGR0325 INSUFFICIENT MEMORY

LGR0326 TOO MANY WILDCARDS

LGR0327 DATE INVALID

LGR0328 FILE IS AN OML LIBRARY

LGR0329 FILE IS A COBLUR LIBRARY

LGR0330 WRONG RETRIEVAL ADDRESS

LGR0331 OPTION *INCREMENT FOR READING LIBRARY ACCESS NOT ALLOWED

LGR0332 OPTION *INCREMENT POSSIBLE ONLY WITH LMS/PLAM V2.0

LGR0333 OPTION *HIGHEST POSSIBLE ONLY WITH LMS/PLAM V2.0

LGR0398 DMS ERROR (&00)

Meaning
For more detailed information about the DMS error code enter /HELP-MSG in system mode or see the BS2000 manual 'System Messages'

LGR0399 INTERNAL ERROR: CC-DMS INTERFACE ERROR

LGR1000 TIME FOR LIST GENERATION: (&00) SECONDS

LGR1001 INTERNAL ERROR IN 'ASSLG' WHEN READING STATEMENT: UNRECOVERABLE SYSTEM ERROR

Meaning
This message is intended for the ASSLG development team.

Response
Contact the system administrator.

LGR1002 INTERNAL ERROR IN 'ASSLG' WHEN READING STATEMENT: OPERAND ERROR IN MACRO 'RDSTMT'

Meaning
This message is intended for the ASSLG development team.

Response
Contact the system administrator.

LGR1003 INTERNAL ERROR IN 'ASSLG' WHEN READING STATEMENT: TRANSFER AREA TOO SMALL

Meaning
This message is intended for the ASSLG development team.

Response
Contact the system administrator.

LGR1004 '//END' ASSUMED DUE TO 'EOF'

LGR1005 'SDF' NOT LOADED

Response
Contact the system administrator.

LGR1006 SYNTAX FILE DOES NOT CONTAIN '//GENERATE STATEMENT'

Response
Contact the system administrator.

LGR1007 INTERNAL ERROR IN 'ASSLG' WHEN READING STATEMENT: 'SDF-RTC=(&00)'

Meaning
This message is intended for the ASSLG development team.

Response
Contact the system administrator.

LGR2000 MANDATORY FIELD '(&00)' NOT IN CURRENT ASPECT

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2001 INVALID OPTION NAME ENCOUNTERED IN LINE (&00)

LGR2002 INVALID DIRECTIVE NAME ENCOUNTERED IN LINE (&00)

LGR2003 DIGIT EXPECTED IN LINE (&00)

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2004 STRING TOO LONG IN LINE (&00)

LGR2005 NAME TOO LONG IN LINE (&00)

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2006 INVALID STRING IN LINE (&00)

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2007 KEYWORD IN LINE (&00) INVALID

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2008 TEMPLATE ID IN LINE (&00) INVALID

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2009 TEMPLATE ID IN LINE (&00) ONLY SIGNIFICANT UP TO 4 CHARACTERS

Meaning
message to ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2010 TEMPLATE IN LINE (&00) NOT DEFINED

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2011 SECTION SPECIFICATION EXPECTED IN LINE (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR2012 'DEF' OR 'ENDDEFS' EXPECTED IN LINE (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR2013 'ASP' OR 'ENDASPS' EXPECTED IN LINE (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR2014 THIS ASP HAS MORE FIELDS THAN ORIGINALLY DEFINED (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR2015 NO MERGE FIELD SPECIFIED. ONLY LAST 'IT' OPENED (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR2016 INTERNAL ERROR. REASON IN LINE (&00)

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR2017 DEFINITION OF TEMPLATE WITH INTERNAL CODE '(&00)' INVALID

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact system administrator

LGR3000 IT NAME '(&00)' INVALID

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR3001 IT NUMBER '(&00)' INVALID

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR3002 FIELD NAME '(&00)' INVALID

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR3003 FIELD NUMBER '(&00)' INVALID

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR3004 OPENING OF FILE '(&00)' NOT POSSIBLE

LGR3005 LG INTERFACE VERSION NUMBER (&00) INVALID

LGR3006 EXCEPTION HANDLER '(&00)' MISSING

LGR3007 REQUIRED FIELD ’(&00)’ MISSING IN ASPECT

Meaning
message for ASSLG development team: error in SCRIPT

Response
contact the system administrator

LGR4000 NO MORE MEMORY SPACE AVAILABLE

LGR4001 LG OPTIONS INVALID

LGR4002 PUT-GET BUFFER NOT YET ALLOCATED

LGR4999 INTERNAL LG ERROR

Meaning
This message is intended for the ASSLG development team.

Response
Contact the system administrator.

11.2 Lookahead mechanism

The lookahead mechanism is a function that is performed in connection with the use of macro language elements in the assembler source program text. Lookahead implies that the source text instructions are read and scanned into an internal file, which can thus be referenced. Lookahead starts with the instruction that satisfies at least one of the following criteria up to the end of the assembly unit:

(1) A still undefined sequence symbol in the operand entry of an AGO or AIF instruction,

(2) A reference to attributes of still undefined symbols in the condition of the AIF instruction,

(3) A reference to attributes of still undefined symbols in the operand entry of the SET instruction,

(4) The first occurrence of a sequence symbol in the name entry of an instruction (1-4, see "ASSEMBH (BS2000) Reference Manual" [1]).

Note

If high performance is required at assembly time, source programs should be written in such a way that no lookahead is needed.

11.3 Format of machine instructions

The instruction list below contains the instructions of the BS2000-NXS (SET1), BS2000-XS (SET3) and BS2000-ESA instruction sets (the Assembler instructions are described in the "Assembler Instructions" Language Reference Manual [11]).
The BS2000-NXS instruction set supports systems with 24-bit addressing (NXS stands for Non-eXtended System).
The BS2000-XS instruction set supports XS systems with 31-bit addressing (XS stands for eXtended System).
The BS2000-ESA instruction set supports ESA systems, which allow for expansion of the virtual address space (ESA stands for Enterprise Systems Architecture).
The BS2000-NXS instruction set is incorporated in the BS2000-XS instruction set, and both are incorporated in the BS2000-ESA instruction set.
The instruction set to which each instruction belongs is indicated by the initial letter N, X or E in the NXS / XS / ESA column.
In the list below, the instructions marked N represent the basic instruction set, while those marked X or E belong to the corresponding extended instruction sets.

Mnemonic code	Instruction name	NXS XS ESA	Mach. code	Length	Operand format
A	Add	N	5A	4	R1,D2(X2,B2)
AD	Add normalized, long	N	6A	4	R1,D2(X2,B2)
ADR	Add normalized, long	N	2A	2	R1,R2
AE	Add normalized, short	N	7A	4	R1,D2(X2,B2)
AER	Add normalized, short	N	3A	2	R1,R2
AH	Add halfword	N	4A	4	R1,D2(X2,B2)
AL	Add logical	N	5E	4	R1,D2(X2,B2)
ALR	Add logical	N	1E	2	R1,R2
AP	Add decimal	N	FA	6	D1(L1,B1),D2(L2,B2)
AR	Add	N	1A	2	R1,R2
AU	Add unnormalized, short	N	7E	4	R1,D2(X2,B2)
AUR	Add unnormalized, short	N	3E	2	R1,R2
AW	Add unnormalized, long	N	6E	4	R1,D2(X2,B2)
AWR	Add unnormalized, long	N	2E	2	R1,R2
AXR	Add normalized with extended length	N	36	2	R1,R2
BAL	Branch and link	N	45	4	R1,D2(X2,B2)
BALR	Branch and link	N	05	2	R1,R2
BAS	Branch and link	N	4D	4	R1,D2(X2,B2)
BASR	Branch and link	N	0D	2	R1,R2
BASSM	Branch and save and set mode	X	0C	2	R1,R2
BC	Branch on condition	N	47	4	I,D2(X2,B2)
BCR	Branch on condition	N	07	2	I,R2
BCT	Branch on count	N	46	4	R1,D2(X2,B2)
BCTR	Branch on count	N	06	2	R1,R2
BSM	Branch and save	X	0B	2	R1,R2
BXH	Branch on index high	N	86	4	R1,R3,D2(B2)
BXLE	Branch on index low or equal	N	87	4	R1,R3,D2(B2)
C	Algebraic comparison	N	59	4	R1,D2(X2,B2)
* CCPU	Check CPU	N	AC	4	D1(B1),I2
CCW	Define channel command word	N		8	I1,I2,I3,I4
CCW0	Define channel command word (format 0)	X		8	I1,I2,I3,I4
CCW1	Define channel command word (format 1)	X		8	I1,I2,I3,I4
CD	Compare long	N	69	4	R1,D2(X2,B2)
CDR	Compare long	N	29	2	R1,R2
CDS	Compare double and swap	N	BB	4	R1,R3,D2(B2)
CE	Compare short	N	79	4	R1,D2(X2,B2)
CER	Compare short	N	39	2	R1,R2
CH	Compare halfword	N	49	4	R1,D2(C2,B2)
* CIOC	Check I/O controller	N	AD	4	D1(B1),I2
* CKC	Check channel	N	9F	4	D1(B1)
CL	Compare logical	N	55	4	R1,D2(X2,B2)
CLC	Compare logical	N	D5	6	D1(L,B1),D2(B2)
CLCL	Compare logical characters long	N	0F	2	R1,R2
CLI	Compare logical	N	95	4	D1(B1),I2
CLM	Compare logical chars. under mask	N	BD	4	R1,M3,D2(B2)

Mnemonic code	Instruction name	NXS XS ESA	Mach. code	Length	Operand format
CLR	Compare logical	N	15	2	R1,R2
CP	Compare decimal	N	F9	6	D1(L1,B1),D2(L2,B2)
CPYA	Copy Access Register	E	B24D	4	R1,R2
CR	Algebraic comparison	N	19	2	R1,R2
CS	Compare and swap	N	BA	4	R1,R3,D2(B2)
* CSCH	Clear subchannel	X	B230	4	No operand
CVB	Convert into binary form	N	4F	4	R1,D2(X2,B2)
CVD	Convert into decimal form	N	4E	4	R1,D2(X2,B2)
D	Divide	N	5D	4	R1,D2(X2,B2)
DD	Divide long	N	6D	4	R1,D2(X2,B2)
DDR	Divide long	N	2D	2	R1,R2
DE	Divide short	N	7D	4	R1,D2(X2,B2)
DER	Divide short	N	3D	2	R1,R2
* DIG	Diagnose	N	83	4	D1(B1)
DP	Divide decimal	N	FD	6	D1(L1,B1),D2(L2,B2)
DR	Divide	N	1D	2	R1,R2
DXR	Divide extended	X	B22D	4	R1,R2
EAR	Extract Access Register	E	B24F	4	R1,R2
ED	Edit	N	DE	6	D1(L,B1),D2(B2)
EDMK	Edit and mark	N	DF	6	D1(L,B1),D2(B2)
** EPAR	Extract primary ASN	X	B226	4	R1
** ESAR	Extract secondary AS	X	B227	4	R1
EX	Execute	N	44	4	R1,D2(X2,B2)
* FC	Execute special functions	N	9A	4	D1(B1),I2
* FCAL	Execute special functions	N	B7	4	D1(B1),I2
HDR	Halve long	N	24	2	R1,R2
* HDV	Halt device	N	9E	4	D1(B1)
HER	Halve short	N	34	2	R1,R2
* HSCH	Halt subchannel	X	B231	4	No operand
** IAC	Insert address space control	E	B224	4	R1
IC	Insert character	N	43	4	R1,D2(X2,B2)
ICM	Insert character with mask	N	BF	4	R1,M3,D2(B2)
* IDL	Idle	N	80	4	I2
** IPK	Insert PSW key	X	B208	4	No operand
IPM	Insert program mask	N	B222	4	R1
* ISK	Interrogate memory protect key	N	09	2	R1,R2
** IVSK	Insert virtual storage key	X	B223	4	R1,R2
L	Load	N	58	4	R1,D2(X2,B2)
LA	Load address	N	41	4	R1,D2(X2,B2)
LAE	Load Address Extended	E	51	4	R1,D2(X2,B2)
LAM	Load Access Multiple	E	9A	4	R1,R3,D2(B2)
LCDR	Load complement, long	N	23	2	R1,R2
LCER	Load complement, short	N	33	2	R1,R2
LCR	Load complement	N	13	2	R1,R2
LD	Load, long	N	68	4	R1,D2(X2,B2)
LDR	Load, long	N	28	2	R1,R2
LE	Load, short	N	78	4	R1,D2(X2,B2)
LER	Load, short	N	38	2	R1,R2
LH	Load halfword	N	48	4	R1,D2(X2,B2)
LM	Load multiple	N	98	4	R1,R3,D2(B2)
LNDR	Load negative, long	N	21	2	R1,R2

Mnemonic code	Instruction name	NXS XS ESA	Mach. code	Length	Operand format
LNER	Load negative, short	N	31	2	R1,R2
LNR	Load negative	N	11	2	R1,R2
LPDR	Load positive, long	N	20	2	R1,R2
LPER	Load positive, short	N	30	2	R1,R2
LPR	Load positive	N	10	2	R1,R2
LR	Load	N	18	2	R1,R2
LRDR	Load rounded extended to long	N	25	2	R1,R2
LRER	Load rounded extended to short	N	35	2	R1,R2
* LSM	Load shadow memory	N	D9	6	D1(L,B1),D2(B2)
* LSP	Load scratch pad	N	D8	6	D1(L,B1),D2(B2)
LTDR	Load and test, long	N	22	2	R1,R2
LTER	Load and test, short	N	32	2	R1,R2
LTR	Load and test	N	12	2	R1,R2
M	Multiply	N	5C	4	R1,D2(X2,B2)
MD	Multiply, long	N	6C	4	R1,D2(X2,B2)
MDR	Multiply, long	N	2C	2	R1,R2
ME	Multiply, short	N	7C	4	R1,D2(X2,B2)
MER	Multiply, short	N	3C	2	R1,R2
MH	Multiply halfword	N	4C	4	R1,D2(X2,B2)
MP	Multiply decimal	N	FC	6	D1(L1,B1),D2(L2,B2)
MR	Multiply	N	1C	2	R1,R2
* MSCH	Modify subchannel	X	B232	4	D2(B2)
MVC	Move characters	N	D2	6	D1(L,B1),D2(B2)
MVCL	Move characters, long	N	0E	2	R1,R2
** MVCP	Move to primary	X	DA	6	D1(R1,B1),D2(B2),R3
** MVCS	Move to secondary	X	DB	6	D1(R1,B1),D2(B2),R3
MVI	Move immediate	N	92	4	D1(B1),I2
MVN	Move numerics	N	D1	6	D1(L,B1),D2(B2)
MVO	Move with offset	N	F1	6	D1(L1,B1),D2(L2,B2)
MVZ	Move zones	N	D3	6	D1(L,B1),D2(B2)
MXD	Multiply long to extended	N	67	4	R1,D2(X2,B2)
MXDR	Multiply long to extended	N	27	2	R1,R2
MXR	Multiply extended	N	26	2	R1,R2
N	AND	N	54	4	R1,D2(X2,B2)
NC	AND	N	D4	6	D1(L,B1),D2(B2)
NI	AND	N	94	4	D1(B1),I2
NR	AND	N	14	2	R1,R2
O	OR	N	56	4	R1,D2(X2,B2)
OC	OR	N	D6	6	D1(L,B1),D2(B2)
OI	OR	N	96	4	D1(B1),I2
OR	OR	N	16	2	R1,R2
PACK	Pack	N	F2	6	D1(L1,B1),D2(L2,B2)
** PC	Change function status	X	B218	4	D2(B2)
** PT	Program transfer	X	B228	4	R1,R2
* RCHP	Reset channel path	X	B23B	4	No operand
* RDD	Read direct	N	85	4	D1(B1),I2
* RSCH	Resume subchannel	X	B238	4	No operand
S	Subtract	N	5B	4	R1,D2(X2,B2)
SAC	Set address space control	E	B219	4	D2(B2)

Mnemonic code	Instruction name	NXS XS ESA	Mach. code	Length	Operand format
* SAL	Set address limit	X	B237	4	No operand
SAR	Set Access Register	E	B24E	4	R1,R2
* SCHM	Set channel monitor	X	B23C	4	No operand
SD	Subtract normalized, long	N	6B	4	R1,D2(X2,B2)
SDR	Subtract normalized, long	N	2B	2	R1,R2
* SDV	Start device	N	9C	4	D1(B1)
SE	Subtract normalized, short	N	7B	4	R1,D2(X2,B2)
SER	Subtract normalized, short	N	3B	2	R1,R2
SH	Subtract halfword	N	4B	4	R1,D2(X2,B2)
SL	Subtract logical	N	5F	4	R1,D2(X2,B2)
SLA	Shift left single	N	8B	4	R1,D2(B2)
SLDA	Shift left double	N	8F	4	R1,D2(B2)
SLDL	Shift left double logical	N	8D	4	R1,D2(B2)
SLL	Shift left single logical	N	89	4	R1,D2(B2)
SLR	Subtract without overflow	N	1F	2	R1,R2
SP	Subtract decimal	N	FB	6	D1(L1,B1),D2(L2,B2)
** SPKA	Set PSW key from address	X	B20A	4	D2(B2)
SPM	Set program mask	N	04	2	R1
SR	Subtract	N	1B	2	R1,R2
SRA	Shift right single	N	8A	4	R1,D2(B2)
SRDA	Shift right double	N	8E	4	R1,D2(B2)
SRDL	Shift right double logical	N	8C	4	R1,D2(B2)
SRL	Shift right single logical	N	88	4	R1,D2(B2)
SRP	Shift and round decimal	N	F0	6	D1(L1,B1),D2(B2),I3
* SSCH	Start subchannel	X	B233	4	D2(B2)
* SSK	Set memory protect key	N	08	2	R1,R2
* SSM	Store shadow memory	N	DA	6	D1(L,B1),D2(B2)
* SSP	Store scratch pad	N	D0	6	D1(L,B1),D2(B2)
ST	Store	N	50	4	R1,D2(X2,B2)
STAM	Store Access Multiple	E	9B	4	R1,R3,D2(B2)
STC	Store character	N	42	4	R1,D2(X2,B2)
STCK	Store clock	N	B2	4	D1(B1)
STCM	Store character with mask	N	BE	4	R1,M3,D2(B2)
* STCPS	Store channel path status	N	B23A	4	D2(B2)
* STCRW	Store channel report word	N	B239	4	D2(B2)
STD	Store long	N	60	4	R1,D2(X2,B2)
STE	Store short	N	70	4	R1,D2(X2,B2)
STH	Store halfword	N	40	4	R1,D2(X2,B2)
STM	Store multiple	N	90	4	R1,R3,D2(B2)
* STSCH	Store subchannel	X	B234	4	D2(B2)
SU	Subtract unnormalized, short	N	7F	4	R1,D2(X2,B2)
SUR	Subtract unnormalized, short	N	3F	2	R1,R2
SVC	Supervisor call	N	0A	2	I
SW	Subtract unnormalized, long	N	6F	4	R1,D2(X2,B2)
SWR	Subtract unnormalized, long	N	2F	2	R1,R2

Mnemonic code	Instruction name	NXS XS ESA	Mach. code	Length	Operand format
SXR	Subtract normalized extended	N	37	2	R1,R2
TAR	Test Access Register	E	B24C	4	R1,R2
* TDV	Test device	N	9D	4	D1(B1)
TM	Test under mask	N	91	4	D1(B1),I2
* TPI	Test pending interruption	X	B236	4	D2(B2)
TR	Translate	N	DC	6	D1(L,B1),D2(B2)
* TRACE	Trace	X	99	4	R1,R3,D2(B2)
TRT	Translate and test	N	DD	6	D1(L,B1),D2(B2)
TS	Test and set	N	93	4	D1(B1)
* TSCH	Test subchannel	X	B235	4	D2(B2)
UNPK	Unpack	N	F3	6	D1(L1,B1),D2(L2,B2)
* WRD	Write direct	N	84	4	D1(B1),I2
X	Exclusive-OR operation	N	57	4	R1,D2(X2,B2)
XC	Exclusive-OR operation	N	D7	6	D1(L,B1),D2(B2)
XI	Exclusive-OR operation	N	97	4	D1(B1),I2
XR	Exclusive-OR operation	N	17	2	R1,R2
ZAP	Zero and add	N	F8	6	D1(L1,B1),D2(L2,B2)

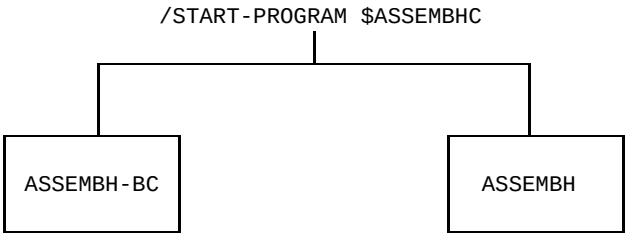
* Privileged instructions

** Semi-privileged instructions

11.4 *COMOPT statements

For reasons of compatibility, the ASSEMBH assembler continues to support the earlier *COMOPT control statements. However, the new features of ASSEMBH are not supported by *COMOPT.

The **ASSEMBH-BC** and **ASSEMBH** assemblers are started for *COMOPT control as follows:



*COMOPT statements are read from SYSDTA:

– as soon as the assembler has been loaded,

– at every restart.

A *COMOPT statement begins with *COMOPT and is followed by one or more options, separated by commas.
It is possible to continue an option beyond the end of a line and into a continuation line; however, the line may only be broken at positions which could also contain a space (blank). This means that words cannot be split,

e.g. *COMOPT SOURCE = AN
TONY is incorrect;

*COMOPT SOURCE = ␣␣
ANTONY is permitted.

There are no format-specific requirements when entering options (e.g. SOURCE␣=␣A␣,␣). The input of *COMOPT statements is terminated by *END (see also *END HALT and *HALT below).

The activated *COMOPT statements are listed in SDF format. Errors are output to SYSOUT and to the listing and may be corrected with the aid of another *COMOPT statement. When ASSEMBH is run with *COMOPT control, the generated module and listing are always compatible with the F-Assembler.

If *COMOPTs are entered via SYSDTA and the source program is read from a file or library, the assembler will, on completing the assembly, request COMOPTs for the next assembly. To prevent this, there are two additional methods of terminating the assembler besides the EOF condition:

– The HALT operand in the *END statement (see section 11.4.1, "Ending the input of options") terminates the assembler after the assembly.

– The *HALT statement instead of a *COMOPT or *END statement terminates the assembler immediately.

If an invalid entry is inadvertently made instead of the first *COMOPT statement, the assembler will interpret this input as the first source program line. By entering ␣END, an assembly of this invalid input can be initiated (a listing will also be created here by default), and the correct options for the next assembly can be entered thereafter.

11.4.1 Table of *COMOPT statements

*COMOPT	Meaning
ADIAG=n	A diagnostic file is generated (see COMOPT SAVLST). Following the assembly, the $ASSDIAG routine is started implicitly when errors with the value n (see chapter 8) or MNOTEs with a corresponding severity code are encountered. 0 ≤ n ≤ 3
ALTLIB[n]	Assigns a macro library or the n-th macro library (2 ≤ n ≤ 5)
NOALTLIB[n]	Default assignment (SYSLIB only)
ATXREF	The references in the cross-reference listing are shown with an attribute that refers to the mode of access: W Write access R Read-only access by instructions A Address access E EQU/ORG instructions Blank Other assembly instructions
NOATXREF	Default assignment. No attribute XREF.
DUET	Allows TRANSDATA 960 instructions.
NODUET	Default assignment

*COMOPT	Meaning
ERR=n	If more than n errors occur, the assembly is terminated with TERM UNIT=STEP,MODE=ABNORMAL. 0 ≤ n ≤ 255
ERR=Sm	If errors with severity (weight) ≥ m or MNOTEs with a severity code class ≥ m are encountered, the assembly is terminated with TERM UNIT=STEP,MODE=ABNORMAL. 0 ≤ m ≤ 3 Assignment Weight | Error class 0 | Warning 1 | Significant 2 | Serious 3 | Fatal / Failure
ERRFIL	Output of the error list to an error file with the link name ERRLINK. If the file is not known at assembly time, it is created under the name ERRFIL.ASSEMBH.{<tsn> | <CSECT-name>} The file is only created if the weight set in the COMOPT statement ERRPR is exceeded. If ERRFIL and SAVLST are specified simultaneously, ERRFIL will be ignored.
ERRPR=n	Only those error flags with a severity lower than n are evaluated for the end messages of the assembler and for the COMOPT ERR. 0 ≤ n ≤ 3 Default value: n=1
FLGLST	For statements with error flags, the error types (max. 3) are printed in the left margin of the line, and the corresponding error texts are printed after the line.
NOFLGLST	For statements with error flags, the error types (max. 3) are printed in the left margin of the line.
HWTST	For special tests the CCW flag byte may be transferred unmodified (unabbreviated). However, this occurrence itself will be flagged.
NOHWTST	No unmodified transfer of the CCW flag byte.

*COMOPT	Meaning
INSTR={SET1 / SET3}	Specifies the instruction set to be generated. SET1 instruction set BS2000-NXS (see section 11.3) SET3 instruction set BS2000-XS (see section 11.3)
ISD	The assembler also outputs AID information to the object module. ISD cards are no longer created.
NOISD	Default assignment
LINECNT=n	Controls the number of lines per print page, including the header line (15 ≤ n ≤ 255). Default assignment: n = 60
LIST	Default assignment. The assembler listing is output to SYSLST.
NOLIST	Only the invalid instructions are displayed.
MLPRNT	The macro identification line consisting of the version number, creation date, and link name of the macro library is output to the assembler listing. The version number consists of blanks if the macro was placed in the macro library by the MLU utility routine.
NOMLPRNT	Default assignment
MODULE= specification	Specifies where the object module is to be placed. If this option is not used, the object module is output to the EAM file. For a full description of this option, see section 11.4.3.

*COMOPT	Meaning
NDLIST	An assembler listing with a layout edited for output on laser printer is generated.
NONDLIST	Default assignment
PRTALL	Generates a complete assembler listing. The options of the PRINT statement, NOGEN, OFF and NOCOPY, are suppressed.
NOPRTALL	Default assignment
PRTIT	The effect of the TITLE statements generated by macros is retained, even if the printing of TITLE statements is suppressed by PRINT NOGEN.
NOPRTIT	The effect of the TITLE statements generated by macros is suppressed by PRINT NOGEN.
PRTOFF= {n {X1[;X2]...[;X5]}	The instructions generated by macros are either printed or not printed, depending on the macro nesting level and the prefix (first character) of the macro name. n Instructions generated by macros from the n-th level onward are not printed. 1 ≤ n ≤ 250 X1[;X2]...[;X5] Instructions generated by macros are never printed if the first character of the macro name is specified in the list X1 to X5 (regardless of any macro level that may have been set with PRTOFF=n). This list may contain up to five first characters of macro names; all macros having names that begin with one of these characters are affected.

*COMOPT	Meaning
SAVLST	The diagnostic listing for ASSDIAG is output to a diagnostic file with the link name SAVLINK. If the file is unknown at assembly time, it is created under the name: SAVLST.ASSEMBH.{<tsn> | <CSECT-name>} The name is created with <tsn> if the CSECT instruction is unnamed; otherwise, with the <CSECT-name>. If the assembler is started with the SAVLST option more than once in a job, a /RELEASE SAVLINK must be issued before each new start in order to avoid overwriting. When a SAVLST is requested, a listing is created via LIST by default. The listing generator of ASSEMBH cannot produce the two outputs (the SAVLST and listing) at the same time; they must be generated sequentially. Since this has an adverse effect on performance, it is advisable to specify the NOLIST option with SAVLST: *COMOPT NOLIST,SAVLST In this way, only the SAVLST is created, and a listing is no longer output.
NOSAVLST	Default assignment
SEQ=(number [,length[,id]])	Entries relating to the identification field (columns 73-80) in the assembler listing: number = Initial numbering with an (implicit) increment of 100. Leading zeros may be omitted. length = Number of positions for numbering, right-justified in identification field. 4 ≤ length ≤ 8 (default value = 8) id = Alphanumeric identifier that is taken over from column 73 into the identification field (maximum 4 characters). In the case of expansion of COPY elements or macros, and with generated literals, no numbering takes place.
SOURCE= specification	Specifies from where the source program is to be read. If this option is omitted, the source is read from SYSDTA. For a full description of this option, see section 11.4.2.

*COMOPT	Meaning
SYSPARM= 'max. 8 characters'	The system parameter &SYSPARM (an 8-byte long character variable; see "ASSEMBH (BS2000) Reference Manual" [1]) is assigned the specified entry and can be interpreted during macro processing.
XREF	The cross-reference listing is output.
NOXREF	Default assignment. No cross-reference listing output.

Notes

– The following *COMOPT statements are no longer supported:
COPYMAC, MCALL, MDIAG, OUTPUT, PROCOM, UPD and SOURCE = +

– If withdrawn statements are used, an appropriate message is issued.

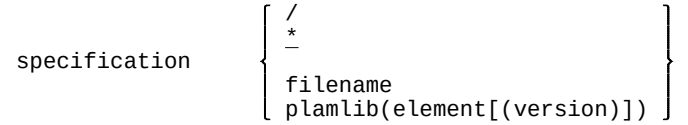
Ending the input of options

	Meaning
*END	End of *COMOPT statement input and start of assembly. Request for new options after assembly.
*END HALT	Same as *END, but with termination of the assembler after assembly.
*HALT	Immediate termination of assembler; assembly not started.

11.4.2 SOURCE option

The SOURCE option can be used to specify the location from which the source program is to be read. If the SOURCE option is omitted, the source program will be read from SYSDTA.

SOURCE = specification



If no entry for "specification" is made, the source program will be read from SYSDTA.

/ — An interrupt occurs after the options are read. SYSDTA can be assigned by the /SYSFILE command via SYSCMD. The source program is then read in via SYSDTA. The new assignment of SYSDTA will, however, not take effect until all options have been processed.

* — The source program is read from SYSDTA. This is the default setting if no specification is made for SOURCE.

filename — Name of a cataloged file containing the source program.
The name may be up to 54 characters in length, including alphanumeric characters, period and hyphen.

plamlib — Name of a program library that was created in accordance with LMS conventions (see "LMS User Guide" [8]) and which contains the source program (as an element of type S).
In line with BS2000 conventions for file names, the name can have a maximum length of 54 characters.

element — Name of the library element (type = S) in which the source program is stored.
The maximum admissible length for element is 54 characters.

version — Version designation of the element.
The version entry may be up to 24 characters in length.
If no version is specified, the element (type = S) with the highest existing version is used.

Notes

– Entries in the SOURCE option (library name, element name, and version) are only checked for admissible length, not for correct syntax according to LMS conventions (see "LMS Reference Manual" [8).

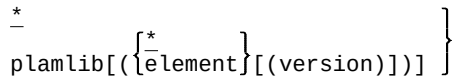
– On libraries
  In addition to PLAM libraries, OSM source program libraries are also allowed: lib(name)

11.4.3 MODULE option

This option can be used to control output of the object module. If the option is omitted, the object module is output to the EAM file.

MODULE = specification

specification



*

The object module is output to the EAM file.

plamlib

Name of a program library created in accordance with LMS conventions (see "LMS User Guide" [8]). The object module is stored as an element of type R (module).
If there is no program library under the given name, a new one is created by the assembler.

element

Element name of the object module.
The element name must comply with the "Rules for element designations in program libraries" (see "LMS User Guidel" [8]). The element (type = R = module) is stored in the program library under this name (maximum 54 characters).

*

If * (asterisk) is specified, the element is assigned the name of the first control section (CSECT name) of the object module. If the first control section is unnamed, the element *cannot* be output to the program library. If the program does not contain a CSECT instruction (or START instruction), the first DSECT or COM name is used as the name of the object element. The same applies if only the library name is specified.

version

Version designation of the element.
The version entry may be up to 24 characters in length.

Character set supported by LMS:

Letters:	A-Z
Digits:	0-9
Special characters:	'.', '-', '@'

If this entry is omitted, the element is assigned the highest version number (represented in the program library by means of a '@' character).
The '@' character may no longer be used as a version as of PLAM V1.4.
If an element with the same version already exists, it is overwritten.

Notes

- The entries in the MODULE option are not checked for syntax (see also the notes on the SOURCE option).
- The linkage editor currently processes element names with a maximum of 8 characters only.

11.4.4 Comparison of *COMOPT and COMPILE statements

*COMOPT	//COMPILE
ADIAG=n	CORRECTION-CYCLE=YES
ALTLIB[n]	MACRO-LIBRARY= COPY-LIBRARY=
ATXREF	[LISTING] CROSS-REFERENCE=(SYMBOL=YES)
DUET	[SOURCE-PROPERTIES] INSTRUCTION-SET=DUET
ERR=n	[COMPILER-TERMINATION] MAX-ERROR-NUMBER=n Default assignment : n = 32767; or n = 0..32767
ERR=Sm	MAX-ERROR-WEIGHT={WARNING / SIGNIFICANT / SERIOUS / FATAL}
ERRFIL	
ERRPR=n	[LISTING] MIN-MESSAGE-WEIGHT={NOTE / WARNING / SIGNIFICANT / SERIOUS / FATAL}
FLGLST	[LISTING] MESSAGE-PLACEMENT=INSERTED
HWTST	[MAINTENANCE-OPTIONS] CHANNEL-INSTRUCTIONS=YES
INSTR={SET1 / SET3}	[SOURCE-PROPERTIES] INSTRUCTION-SET={HOST-STD / BS2000-NXS / BS2000-XS}
ISD	TEST-SUPPORT=YES

*COMOPT	//COMPILE
LINECNT=n	[LISTING] LAYOUT=(LINES-PER-PAGE=n) Default assignment: n = 60; or n = 15..255
NOLIST	[LISTING] SOURCE-PRINT=ERRORS-ONLY
MLPRNT	[LISTING, MACRO-PRINT] MACRO-ORIGIN-INFO=INSERTED
MODULE= specification	MODULE-LIBRARY=
NDLIST	[LISTING] LAYOUT=(LASER-PRINTER=ND2)
PRTALL	[LISTING] SOURCE-PRINT=WITH-OBJECT-CODE(PRINT-STATEMENTS=IGNORED)
NOPRTIT	[LISTING, MACRO-PRINT] TITLE-STATEMENTS=IGNORED
PRTOFF= ⎡n ⎤ ⎪ ⎪ { } ⎪ ⎪ ⎣X1[;X2]...[;X5]⎦	[LISTING, MACRO-PRINT] NOPRINT-NEST-LEVEL=n Default assignment : n = 255; or n = 1..255 PREFIX-EXCEPTION=(A,B,C,...) A list of up to 256 macro name prefixes is permitted.
SAVLST	
SEQ=(number [,length[,id]])	[LISTING] SOURCE-PRINT=(LINE-NUMBERING=YES)
SOURCE= specification SOURCE=/	SOURCE= SOURCE=*SYSDTA-AFTER-BREAK
SYSPARM= 'max. 8 characters'	[SOURCE-PROPERTIES] SYSPARM=C'ABC...' or 'ABC...' A maximum of 255 characters are possible.
XREF	[LISTING] CROSS-REFERENCE=(WITH-ATTRIBUTES=NO)

12 Manual supplements

This chapter is an update for the present manual valid for ASSEMB V1.2D.

12.1 Controlling ASSEMBH, the standalone listing generator ASSLG and *COMOPT statements

Section 2.2 Controlling ASSEMBH (page 8),

Section 2.5 The standalone listing generator ASSLG (page 53) and

Section 11.4 *COMOPT statements (page 322)

ASSEMBH and ASSLG as of V1.2 have a separate start command which is assigned to the SDF "UTILITIES" domain:

COMMAND: START-ASSEMBH

or: START-ASSLG

or: START-ASSEMBHC

OPERANDS : CPU-LIMIT=*JOB-REST,MONJV=*NONE

CPU-LIMIT = *JOB-REST or <integer 1..32767>

Maximum CPU time requirement for the program run in seconds

MONJV = *NONE or <full-filename 1..54 without-gen-vers>

Name of the job variable which is to monitor the program run

12.2 COMPILATION-INFO option

Section 2.4.3 COMPILATION-INFO option (page 36ff)

VERSION = *INCREMENT is not supported

12.3 LISTING option

Section 2.4.4 LISTING option (page 38ff)

The correct text for SOURCE-FORMAT = STD is:

A standard listing is generated.

For TITLE-STATEMENTS = ... the default value is IGNORED and the description should read:

TITLE statements generated by macros are executed, but ignored if the PRINT NOGEN directive is specified.

12.4 OUTPUT option

Section 2.5.1 GENERATE statement (page 53ff)

The operand *SAVLST is no longer supported by the OUTPUT option in the GENERATE statement.

12.5 ESD list

Section 6.1.2 ESD list (page 93ff)

Space for 32 characters is generally provided in the ESD list for the symbol names (as described under 6.6.2 for the LLM format).

12.6 Structured list

Section 6.5 Structured list (starting page 108): 3rd. paragraph

To create a structured list you must use the predefined macros (also called structure macros in the following) for structured programming (see "ASSEMBH, Reference Manual" [1]). These must also be fully logged in the list so that a structure block (see 6.5.2) can be correctly edited. If not all of the structure macros are logged, editing of a structure block can be incomplete or errored.

12.7 Parameter ENV=C and LOADR12

Section 7.2.1 Interfacing structured assembler programs with C programs (page 133), following last paragraph:

The parameters ENV=C and LOADR12 can only be used for C programs that are V1-compatible.

The parameter ILCS=YES must always be specified for programs that are generated in CPLUSPLUS-mode.

12.8 Working with the COLNUMA utility

Section 10.4.1 Extending the structure list (page 225):

Notes Permitted element types are P, S, M, D, X.

Section 10.4.2 Enhancing the assembler listing of a program edited by COLINDA (page 227): Notes

Permitted element types are P, S, M, D, X.

12.9 Utility program messages

Section 10.5 Utility program messages (page 230)

Missing message:

```
015 – IO   Meaning: More than 32767 records in the input.
```

Effect: The excess records are output, but only as text,

12.10 Messages

12.10.1 Not included messages

The following ASSEMBH messages have not been included in the User Guide yet or the texts concerned have been changed:

Supplements and changes of ASSEMBH messages im Section 11.1 ASSEMBH messages (page 241).:

```
ASS0217    B17 - SIGNIFICANT ERROR
ASS0217    CONTINUATION COLUMN IN 'ICTL' OPERAND IS WRONG

ASS0336    C36 - NOTE
ASS0336    MACRO (&00): MULTIPLE DEFINITIONS IN SOURCE
```

Meaning
Note concerning incompatibility: A macro instruction will always generate the macro whose definition was processed last.

```
ASS0597    E97 - WARNING
ASS0597    EXTERNAL NAMES TRUNCATED TO 8 CHARACTERS
```

Meaning
The name for entries in the ESD record of the object is limited to 8 characters. Only the first 8 characters of the name are used.

```
ASS0598    E98 - WARNING
ASS0598    EXTERNAL NAMES TRUNCATED TO 32 CHARACTERS

ASS1310    M10 - SIGNIFICANT ERROR
ASS1310    SYMBOL (&00): MULTIPLE DEFINITIONS

ASS1918    S18 - SIGNIFICANT ERROR
ASS1918    LSDV OF (&00) (&01) TOO BIG; AID INFORMATION INCOMPLETE
```

Meaning
The number of elements in a structure exceeds range
(&00): Type of structure ('DSECT' or 'COM' or 'XDSEC')
(&01): Name of structure

Response
None; testing without AID information is possible

```
ASS6012    NO ERROR
ASS6012    END OF ASSEMBH(&00)

ASS6102    NOTE
ASS6102    IN THE OPTION 'PRTOFF=X1;X2...'; AN INVALID CHARACTER IS GIVEN. IT WILL
           BE IGNORED

ASS6118    NO ERROR
ASS6118    MODULE FORMAT LLM ONLY POSSIBLE WHEN MODULE IS PUT TO A PLAM LIBRARY;
           MODUL PUT TO THE STANDARD LIBRARY 'SYS.PROG.LIB'

ASS6119    WARNING
ASS6119    OPTION 'SOURCE-FORMAT=STRUCTURED' NOT POSSIBLE WITH OPTION 'OUTPUT=*SAVLST'
```

Meaning
All specifications for the structured listing are ignored

```
ASS6128    SERIOUS ERROR
ASS6128    'COMPILER INFORMATION FILE' IS NO PLAM LIBRARY
```

Meaning
No listing is generated.

Response
Specify a PLAM library for 'Compiler Information File'.

```
ASS6129    SERIOUS ERROR
ASS6129    THE PLAM LIBRARY MEMBER FOR 'COMPILER INFORMATION FILE' IS LOCKED
```

Meaning
No listing is generated.

Response
Unlock the PLAM library member for 'Compiler Information File'.

```
ASS6200    SIGNIFICANT ERROR
ASS6200    INTERNAL LLM ACCESS ERROR. FUNCTION START-OUTPUT.
```

Meaning
Error in LLM access with the START-OUTPUT function.

Response
Inform the system administrator.

```
ASS6201    WARNING
ASS6201    INTERNAL LLM ACCESS WARNING. SUB_RC_=(&00).
```

Meaning
Warning with LLM access.

Response
Inform the system administrator.

```
ASS6202    SIGNIFICANT ERROR
ASS6202    INTERNAL LLM ACCESS ERROR: FUNCTION= (&00), MAIN-CODE= (&01), SUB-CODE= (&02)
```

Meaning
Error with LLM access. The error type is shown in the MAIN retcode and the SUB retcode.

Response
Inform the system administrator.

```
ASS6203    SIGNIFICANT ERROR
ASS6203    INTERNAL LLM ACCESS ERROR. NO LLM GENERATED BECAUSE OF PRECEDING

ASS6204    FAILURE
ASS6204    ILLEGAL VERSION OF LLMAM FOR ASSEMBH
```

Response
Inform the system administrator, so that he can install the correct LLMAM version.

```
ASS6205    FAILURE
ASS6205    NO LLM GENERATION POSSIBLE DUE TO PRECEDING ERROR

ASS6206    FAILURE
ASS6206    MEMBER NAME FOR MODULE INVALID: NO LLM GENERATION POSSIBLE

ASS6207    WARNING
ASS6207    'TEST-SUPPORT=AID' AND 'MODULE-FORMAT=LLM' FOR THE PRESENT ONLY
           POSSIBLE WITH 'EXTERNAL-NAMES=TRUNCATED'

LGR0008    STRUCTURE NOT CLOSED

LGR0119    DMS-ERROR (&00) WHEN ACCESSING CIF

LGR2100    INTERNAL ERROR WHEN PROCESSING CIF PARTITIONS, CODE: (&00)
```

Meaning
Error text for ASSLG development.

Response
Inform the system administrator.

12.10.2 New/changed ASSEMBH runtime system messages

Supplements and correction in section 11.1.1 (page 303).

```
ASS7001   FATAL ERROR
ASS7001   INITIALIZATION OF THE ASSEMBLER RUNTIME SYSTEM NOT POSSIBLE; ILCS-RTC=(&00)
```

Meaning
Possible causes: RTC= 2: The BS2000 version is not supported
= 3: Version incompatibility IT0SL# and IT0SL@
= 4: Not enough memory for initial stack management
= 5: Not enough memory for initial heap management
= 6: Standard event handler cannot be initialized
= 7: An IxxSINI routine repoted an error in Reg.15

Response
Inform the system administrator.

```
ASS7012   FATAL ERROR
ASS7012   PROGRAM TERMINATION WITH ERROR; ILCS-RTC=(&00)
```

Meaning
Possible causes: RTC= 2: ILCS is not initialized
= 3: Recursive call to an initialization or termination routine
= 4: IT0TERM called from within a server coroutine

Response
Inform the system administrator.

```
ASS7013   FATAL ERROR
ASS7013   NO MORE HEAP AVAILABLE DURING INITIALIZATION OF THE RUNTIME SYSTEM.
```

Meaning
No HEAP memory can be obtained for management information in the runtime system

Response
- Reduce the data request(s);
- Release HEAP memory that is no longer needed;
- Have the system administrator increase the user address space.

References

[1] **ASSEMBH** (BS2000)
Reference Manual

[2] **AID** (BS2000)
Advanced Interactive Debugger
Debugging of ASSEMBH Programs
User Guide

[3] **AID** (BS2000)
Advanced Interactive Debugger
Core Manual
User Guide

[4] **Introductory Guide to XS Programming (for Assembler Programmers)** (BS2000)
User's Guide

[5] **SDF**
(BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide

[6] **BS2000/OSD-BC**
Commands Volume 1-7
User Guide

[7] BS2000
JV Job Variables

[8] **LMS** (BS2000)
SDF Format
User Guide

[9] **BS2000/OSD-BC**
Dynamic Binder Loader / Starter
User Guide

[10] BS2000
Binder
User Guide

[11] **BS2000**
Assembler Instructions
Language Reference Manual

[12] **BS2000/OSD-BC**
ExecutiveMacros
User Guide

Index

*COMOPT statements (see COMOPT statements) 322

A
address space, extended 90
address space requirement 52
Advanced Interactive Debugger (AID) 175
AID
  Advanced Interactive Debugger 175
  example of a debugging run 179
  prerequisites for symbolic debugging 177
  TEST-SUPPORT option 47
ASSDIAG
  command overview 157
  CORRECTION-CYCLE option 50
  definition of terms 154
  error classes 154
  formatted screen I/O 172
  function overview 153
  interrupting the program run 156
  software requirements 153
  starting 155
ASSEMBH
  basic configuration (BC) 1
  calling 7
  control 8
  diagnostic routine ASSDIAG 153
  functionality 1
  input sources 61
  listings 91
  messages 241
  outputs 67
  restarting 9
  SDF interface 10
  structure 2

ASSEMBH ILCS objects, creating 149
ASSEMBH-BC
  calling 7
  functionality 1
assembly 7
  example 13
  monitoring with job variables 71
  multiple 8
  of structured assembler programs 89
  restarting 9
  simple 8
  terminating an assembly run 48
assembly unit 7, 9
ASSIGN-SYSDTA command 63
ASSLG 53, 91
autolink procedure, TSOSLNK 86
automatic version incrementation 37, 46

B

BINDER
  control statements 81
  linking with 81
  LLMs 80
  OMs 80

C

C programs, language interfacing 133
calling ASSEMBH 7
CCW channel instructions, tests 51
CDT command, ASSDIAG 158
CIF support 36
COBOL program, language interfacing 134, 140
COLBIN call, ILCS 150, 152
COLINDA, utility routine for structured programming 187, 198
COLLIST, utility routine for structured programming 187
COLNAS, utility routine for structured programming 187, 195
COLNUMA, utility routine for structured programming 187, 201
COMOPT statements
  comparison with COMPILE statements 332
  end of input 328
  general 322
  table 323
compilation space 52
COMPILATION-INFO, option 36
COMPILATION-SPACE option 52

COMPILE statement
  comparison with COMOPT statements 332
  example 13
  input in SDF menu mode 11
  overview of options 20
COMPILER-ACTION option 32
COMPILER-TERMINATION option 48
CONTINUE-CDT command, ASSDIAG 163
COPY elements
  COPY-LIBRARY option 26
  input 66
  search order 66
COPY-LIBRARY option 26
correction cycle 50, 153
CORRECTION-CYCLE option 50
cross-reference listings 99

D

data structures, ILCS 144
data types, ILCS 146
DBL 80
  linking and loading 83
  LLMs 80
  OMs 80
debugging, with AID 47, 175
diagnostic file 154
diagnostic routine, ASSDIAG (see ASSDIAG) 153
DISPLAY command, ASSDIAG 164
DLL 80

E

ELDE (loader) 88
END command, ASSDIAG 166
error messages (see messages) 230
ESA support 90
ESD information 177
ESD listing 93
executable program 79
expert mode, SDF 10
EXTERNAL SYMBOLIC DICTIONARY (ESD listing) 93
external symbols, masking out 82

F
FORTRAN program, language interfacing 134, 140
function keys, SDF menu mode 12

G
GENERATE statement 54

H
HELP command, ASSDIAG 166

I
ILCS 77
  program communication interface 142
ILCS (Inter-Language Communication Services) 142
ILCS linkage combinations 150
input
  of COPY elements 66
  of macro elements 64
  of options 8, 10, 11
  of source program 63
instruction set option
  BS2000-ESA 29
  BS2000-NXS 29
  BS2000-XS 29
interfacing structured assembler programs 148

J
job variables (see monitoring job variables) 71

L
language interfaces 129
  assembler programs 129
language interfacing
  assembler program segments 141
  assembler programs 136
  C programs 133
  COBOL and FORTRAN programs 134, 140
  structured assembler programs 131, 136, 141
laser printer listings 105
link-and-load module 31
  output 70
  output location 34
link-and-load module generation 32
linkage editor, BINDER 80

linking
  example (TSOSLNK) 87
  general information 79
  ILCS program systems 147
  of structured assembler programs 89
  temporary 83
  with BINDER 81
  with DBL 83
  with TSOSLNK 85
LIST command, ASSDIAG 167
listing
  controlling the output 38
  cross-references (XREF) 99
  ESD 93
  GENERATE statement 54
  in ASSEMB V30 compatible format 102
  laser printer (ND) 105
  options 92
  SAVLST (with ISAM key) 106
LISTING option 38
listings
  description 91
  standard format 91
lists in LLM format 126
LLM, generating with BINDER 81
LLM format 33
load module 79
LOAD-PROGRAM command
  calling DBL 83
  calling the program 88
loading, general information 79
loading a program 88
long-jump 152
lookahead mechanism 315
LSD information 47, 177

M

machine instructions, format 316
macro elements
  input 64
  MACRO-LIBRARY option 24
  search order 65

macro library
  system 65
  user-own 64
MACRO-LIBRARY option 24
macros, ILCS interface 148
maintenance support 51
MAINTENANCE-OPTIONS option 51
masking out of symbols 82
menu mode, SDF 10
messages
  from utilities for structured programming 230
  of ASSEMBH 241
metalanguage 5
metasyntax, SDF interface 16
mnemonic operation code, machine instructions 316
MODIFY-SDF-OPTIONS command 10
MODIFY-SYMBOL-VISIBILITY, BINDER statement 83
MODULE option, COMOPT statement 330
MODULE-LIBRARY option 34
monitoring job variables
  example 75
  monitoring the assembly 71
  support by the runtime system 78
multiple assembly 8

N

Nassi-Shneiderman diagrams, COLNAS 187
NEXT line, SDF menu mode 12
notational conventions 5

O

object module 31
  output 69
  output location 34
object module generation 32
OM-Format 32
operand form, SDF 11
option, for debugging 47

options
CIF support 36
for input support 21
for listing support 38
for maintenance support 51
for object module generation 31
for reducing the virtual address space requirement 52
input 8, 10, 11
overview 20
to activate the correction cycle 50
to terminate assembly 48
OPTIONS LISTING 92
outputs, of ASSEMBH 67

P

parameter passing, ILCS 146
PCD 145
PRINT command, ASSDIAG 167
program
loading 88
permanent 85
starting 88
temporary 83
program interface, ILCS 142
program linking (see language interfaces) 129
program mask 145
PROGRAM statement (TSOSLNK) 85

R

register conventions, ILCS 143
RERUN command, ASSDIAG 168
restart 9
return code, monitoring job variables 72, 78
return values, transfer 147
runtime system, for structured assembly programs 77

S

save area 144
SAVLST (listing with ISAM key) 106
SDF interface
metasyntax 16
of ASSEMBH 10
simple assembly 8
SOURCE LISTING (source program listing) 96

SOURCE option 22
  COMOPT statement 329
source program
  format 28, 63
  input 63
  input from files 63
  input from libraries 64
  input via SYSDTA 63
  input, SOURCE option 22
source program listing 96
SOURCE-PROPERTIES option 28
standalone listing generator 54
START-PROGRAM command
  calling ASSEMBH 7
  calling DBL 83
  calling the program (ELDE) 88
starting, general information 79
starting a program 88
static linkage (TSOSLNK) 80
static linking (TSOSLNK) 85
status indicator, monitoring job variables 72
structured assembler programs
  assembling and linking 89
  interfacing 148
  language interfacing 131
  runtime system for 77
  utility routines 187
structured list (ASSEMBH) 108
subroutines, linking (see language interfaces) 129
symbolic program linking 129
SYSDTA, input of source program 63
SYSTEM command, ASSDIAG 169

T

TAGS command, ASSDIAG 170
TEST-SUPPORT option 47
TOM editor 201
TSOSLNK 80
  autolink procedure 86
  control statements 85
  example of a linkage run 87
  linking with 85

V
V-type constant 130
visibility, of external symbols 82

X
XREF command, ASSDIAG 171
XS support 90

Contents

1	**Preface**	**1**
1.1	Brief product description	2
1.2	Target group	3
1.3	Summary of contents	3
1.4	Changes since the last version of the manual	4
1.5	Notational conventions	5
2	**Assembly**	**7**
2.1	Calling ASSEMBH	7
2.2	Controlling ASSEMBH	8
2.2.1	Simple assembly	8
2.2.2	Multiple assembly	8
2.2.3	Restarting the assembler	9
2.3	SDF interface of ASSEMBH	10
2.3.1	Processing the operand form	11
2.3.2	Metasyntax for the SDF interface	16
2.3.2.1	Data types and suffixes	18
2.4	COMPILE statement	20
2.4.1	Input support options	21
2.4.1.1	SOURCE option	22
2.4.1.2	MACRO-LIBRARY option	24
2.4.1.3	COPY-LIBRARY option	26
2.4.1.4	SOURCE-PROPERTIES option	28
2.4.2	Options for object module generation	31
2.4.2.1	COMPILER-ACTION option	32
2.4.2.2	MODULE-LIBRARY option	34
2.4.3	Option for CIF support COMPILATION-INFO option	36
2.4.4	Option for listing support LISTING option	38
2.4.5	Option for debugging support TEST-SUPPORT option	47
2.4.6	Option to terminate assembly COMPILER-TERMINATION option	48
2.4.7	Option to activate the correction cycle CORRECTION-CYCLE option	50

2.4.8	Option for maintenance support MAINTENANCE-OPTIONS option	51
2.4.9	Option for reducing the virtual address space requirement COMPILATION-SPACE option	52
2.5	The standalone listing generator ASSLG	53
2.5.1	GENERATE statement	53
3	**Input/output of ASSEMBH**	**61**
3.1	Input sources of ASSEMBH	61
3.1.1	Input of the source program	63
3.1.2	Input of macro elements	64
3.1.2.1	Search order for macro elements	65
3.1.3	Input of COPY elements	66
3.1.3.1	Search order for COPY elements	66
3.2	Outputs of ASSEMBH	67
3.2.1	Output of the object module	69
3.2.2	Output of a link-and-load module	70
3.2.3	Monitoring the assembly with the monitoring job variable MONJV	71
3.2.2.1	Structure of the monitoring job variables	71
4	**Runtime system for structured programming**	**77**
4.1	General information	77
4.2	Support for monitoring job variables	78
5	**Linking, loading and starting**	**79**
5.1	General information	79
5.2	Linking with BINDER	81
5.3	Dynamic linking and loading with DBL	83
5.4	Static linking with TSOSLNK	85
5.5	Loading and starting programs using the loader ELDE	88
5.6	Assembling and linking a structured assembler program	89
5.7	XS support	90
5.8	ESA support	90
6	**Description of listings**	**91**
6.1	Listings in standard format	91
6.1.1	Options listing (OPTIONS LISTING)	92
6.1.2	ESD listing (EXTERNAL SYMBOL DICTIONARY)	93
6.1.3	Source program listing (SOURCE LISTING)	96
6.1.4	Listing of files and libraries used	98
6.1.5	Cross-reference listings	99
6.1.6	End message	101
6.2	Listing compatible with ASSEMB V30	102
6.3	Laser printer listing	105
6.4	SAVLST (listing with ISAM key)	106

6.5	Structured list	108
6.5.1	Features of the structuring function	108
6.5.2	The print-edited assembly log	110
6.5.2.1	Handling of instructions	111
6.5.2.2	Handling of comments	113
6.6	Differences in lists where the module is output in LLM format	125
6.6.1	Lists in OM format	126
6.6.2	Lists in LLM format	127
7	**Language interfaces**	**129**
7.1	Symbolic linking of assembler programs	129
7.1.1	Interfacing with other languages	130
7.2	Linking structured assembler programs	131
7.2.1	Interfacing structured assembler programs with C programs	133
7.2.2	Interfacing structured assembler programs with COBOL and FORTRAN programs	134
7.2.3	Interfacing structured assembler programs with assembler programs	136
7.2.4	Interfacing COBOL and FORTRAN program segments with structured assembler programs	140
7.2.5	Interfacing assembler program segments with structured assembler programs	141
7.3	The program communication interface ILCS	142
7.3.1	ILCS register conventions	143
7.3.2	ILCS data structures	144
7.3.3	Initialization of the program system	145
7.3.4	Program mask handling by ILCS	145
7.3.5	Parameter passing in ILCS program systems	146
7.3.6	Notes on linking of ILCS program systems	147
7.4	Program interfacing of structured assembler programs via the ILCS interface	148
7.4.1	Creating an ASSEMBH ILCS object	149
7.5	ILCS linkage combinations	150
7.5.1	ILCS object calls ASSEMBH ILCS object	150
7.5.2	ASSEMBH ILCS object calls ASSEMBH ILCS object	150
7.5.3	ASSEMBH ILCS object calls non-ILCS ASSEMBH object	151
7.5.4	Non-ILCS ASSEMBH object calls ASSEMBH ILCS object	151
7.5.5	Non-ILCS object calls ASSEMBH ILCS object	152
7.5.6	Long-jump (@EXIT with parameter TO)	152
8	**The ASSEMBH diagnostic routine ASSDIAG**	**153**
8.1	Application	153
8.2	Definition of terms	154
8.3	Starting the diagnostic routine	155
8.4	Interrupting the program run	156

8.5	ASSDIAG commands	156
8.5.1	CDT command	158
8.5.1.1	CDT statements	160
8.5.2	CONTINUE-CDT command	163
8.5.3	DISPLAY command	164
8.5.4	END command	166
8.5.5	HELP command	166
8.5.6	LIST command	167
8.5.7	PRINT command	167
8.5.8	RERUN command	168
8.5.9	SYSTEM command	169
8.5.10	TAGS command	170
8.5.11	XREF command	171
8.6	Formatted screen I/O	172
8.6.1	Basic structure of ASSDIAG formats	172
8.6.2	Example: DISPLAY command	172
8.6.3	Example: TAGS command	173
9	**The Advanced Interactive Debugger (AID)**	**175**
9.1	Introduction	175
9.2	Prerequisites for symbolic debugging	177
9.3	Example of a debugging run	179
9.3.1	Assembler program	179
9.3.2	Debugging run	181
10	**Utility routines for structured programming**	**187**
10.1	Utilities which edit the structured source program	189
10.1.1	COLLIST	189
10.1.1.1	Structure list	189
10.1.1.2	Procedure list	193
10.1.2	COLNAS	195
10.1.2.1	Format of the list	195
10.1.3	COLINDA	198
10.1.3.1	Output from COLINDA	198
10.1.3.2	Structure functions available in the TOM editor	201
10.2	COLNUMA	201
10.2.1	Extending the structure list	201
10.2.2	Extending the assembler listing of a program edited by COLINDA	203

10.3	Working with the COLLIST, COLNAS and COLINDA utilities	205
10.3.1	Input for COLLIST, COLNAS and COLINDA	207
10.3.2	Output from COLLIST and COLNAS	207
10.3.3	Output from COLINDA	208
	Summary	209
10.3.4	Control of COLLIST, COLNAS and COLINDA	211
10.3.5	Parameters	212
	Examples	**221**
10.4	Working with the COLNUMA utility	224
10.4.1	Extending the structure list	224
	Summary	226
10.4.2	Enhancing the assembler listing of a program edited by COLINDA	227
	Summary	228
10.4.3	Parameters	229
10.5	Messages from the utilities	230
10.5.1	Operator error messages and system messages	230
10.5.2	Syntax error messages	235
10.5.3	Meaning of aabb in syntax error messages	237
10.6	Support for monitoring job variables	239
11	**Appendix**	**241**
11.1	ASSEMBH messages	241
11.1.1	Messages of the assembler runtime system for structured programming	303
11.1.2	Listing generator messages	306
11.2	Lookahead mechanism	315
11.3	Format of machine instructions	316
11.4	*COMOPT statements	322
11.4.1	Table of *COMOPT statements	323
11.4.2	SOURCE option	329
11.4.3	MODULE option	330
11.4.4	Comparison of *COMOPT and COMPILE statements	332

Manual supplements

References

Index

FUJITSU

ASSEMBH

User Guide

Valid for
ASSEMBH V1.2
With Supplement chapter for ASSEMBH V1.2D

Edition June 2010

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:
manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2000

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.
This document is a new edition of an earlier manual for a product version which was released a considerable time ago in which changes have been made to the subject matter.
Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.
Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format …@ts.fujitsu.com.
The Internet pages of Fujitsu Technology Solutions are available at http://ts.fujitsu.com/...