High Level Assembler for MVS & VM & VSE

SC26-4941-01

Programmer's Guide
MVS & VM Edition

Release 2

This keyline indicates size
and position of graphic
only, keyline does not print.

High Level Assembler for MVS & VM & VSE

SC26-4941-01

Programmer's Guide
MVS & VM Edition

Release 2

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices" on page x.

Second Edition (March 1995)

This edition applies to Release 2 of IBM High Level Assembler for MVS & VM & VSE, 5696-234, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Programming Interface Information

This manual is intended to help the customer create application programs. This manual documents General-Use Programming Interface and Associated Guidance Information provided by IBM High Level Assembler for MVS & VM & VSE.

General-use programming interfaces allow the customer to write programs that obtain the services of IBM High Level Assembler for MVS & VM & VSE.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

DFSMS/MVS
Enterprise System/9000
Enterprise Systems Architecture/370
Enterprise Systems Architecture/390
ES/9000
ESA
ESA/370
ESA/390
IBM
MVS/DFP
MVS/ESA
MVS/XA
OpenEdition
S/370
System/370
System/390
Virtual Machine/Enterprise Systems Architecture
VM/ESA
VSE/ESA
3090

About this Manual

This manual describes how to use the IBM* High Level Assembler for MVS & VM & VSE licensed program, hereafter referred to as High Level Assembler, or simply the assembler. It is intended to help you assemble, link-edit, and run your High Level Assembler programs. It is meant to be used in conjunction with *IBM High Level Assembler for MVS & VM & VSE Language Reference*.

MVS is used in this manual to refer to Multiple Virtual Storage/Enterprise Systems Architecture (MVS/ESA*)

CMS is used in this manual to refer to Conversational Monitor System under Virtual Machine/Enterprise Systems Architecture* (VM/ESA*)

Who Should Use this Manual

IBM High Level Assembler for MVS & VM & VSE Programmer's Guide is for application programmers coding in the High Level Assembler language. To use this manual, you should be familiar with the basic concepts and facilities of your operating system.

Organization of this Manual

This manual is organized as follows:

Part 1. Understanding and Using the Assembler

- **Chapter 1, Introduction,** describes High Level Assembler, and defines the environmental requirements for using the assembler.
- **Chapter 2, Using the Assembler Listing,** describes the content and format of the assembler listing.
- **Chapter 3, Controlling your Assembly with Options,** describes the assembler options that you can use to control the assembly of your program.
- **Chapter 4, Providing User Exits,** describes how you can provide user exits to compliment the assembler's data-set processing.
- **Chapter 5, Providing External Functions,** describes how to provide user-supplied routines in conditional assembly instructions to set the value of SET symbols.
- **Chapter 6, Diagnosing Assembly Errors,** describes the purpose and format of error messages, MNOTEs, and the MHELP trace facility.

Part 2. Developing Assembler Programs under MVS

- **Chapter 7, Assembling your Program under MVS,** describes the different methods of assembling your program under MVS, including invoking the assembler with job control statements, invoking the assembler under TSO, invoking the assembler dynamically, and batch assembling.
- **Chapter 8, Link-Editing and Running your Program under MVS,** describes link-editing, creating load modules, input and output for the

linkage editor, detecting link-edit errors, and running your program under MVS.

- **Chapter 9, MVS System Services and Programming Considerations,** describes the MVS system services that you can use to maintain macro definitions in a macro library, and the cataloged procedures that are provided to help you assemble, link-edit, and run your program under MVS. This chapter also discusses programming topics such as standard entry and exit procedures.

Part 3. Developing Assembler Programs under CMS

- **Chapter 10, Assembling your Program under CMS,** describes how to invoke the assembler under CMS.
- **Chapter 11, Running your Program under CMS,** describes how to load and run your program under CMS.
- **Chapter 12, CMS System Services and Programming Considerations,** describes the CMS system services that you can use to maintain members in a macro library. It also discusses programming topics such as standard entry and exit procedures.

Appendixes

- **Appendix A, Previous Assembler Compatibility and Migration,** provides a comparison of High Level Assembler and Assembler H Version 2, and High Level Assembler and the DOS/VSE Assembler.
- **Appendix B, Cross-System Portability Considerations,** contains information that helps you prepare your program for running under a different operating system.
- **Appendix C, Object Deck Output,** describes the format of the object module generated by the assembler.
- **Appendix D, Associated Data File Output,** describes the format of the associated data file records generated by the assembler.
- **Appendix E, Sample Program,** provides a sample program that demonstrates many of the assembler language features.
- **Appendix F, MHELP Sample Macro Trace and Dump,** provides a sample program listing which shows the primary functions of MHELP.
- **Appendix G, High Level Assembler Messages,** describes the error diagnostic messages, abnormal termination messages, and CMS command error messages issued by the assembler.
- **Appendix H, User Interface Macros,** lists the macros that are provided as Programming Interfaces with High Level Assembler.
- **Appendix I, Sample ADATA User Exit,** provides a description of the sample ADATA user exit supplied with High Level Assembler.
- **Appendix J, Sample LISTING User Exit,** provides a description of the sample LISTING user exit supplied with High Level Assembler.
- **Appendix K, Sample SOURCE User Exit,** provides a description of the sample SOURCE user exit supplied with High Level Assembler to read variable length input files.

- **Appendix L, How to Generate a Translation Table,** provides instructions for generating a translation table to convert the characters contained in character data constants and literals.

Glossary defines the terms used in this manual.

Bibliography lists the IBM Publications referred to within this manual.

IBM High Level Assembler for MVS & VM & VSE Publications

High Level Assembler runs under MVS, VM and VSE*. The publications for the MVS and VM operating systems are described in this section. Refer to “High Level Assembler Publications for VSE” on page 331 for a list of the High Level Assembler publications for the VSE/ESA* operating system.

Hardcopy Publications

The books in the High Level Assembler library are shown in Figure 1. This figure shows which books can help you with specific tasks, such as application programming.

Figure 1. IBM High Level Assembler for MVS & VM & VSE Publications

Task	Publication	Order Number
Evaluation and Planning	General Information	GC26-4943
Installation and Customization	Installation and Customization Guide Programmer's Guide	SC26-3494 SC26-4941
Application Programming	Programmer's Guide Language Reference	SC26-4941 SC26-4940
Diagnosis	Installation and Customization Guide	SC26-3494
Warranty	Licensed Program Specifications	GC26-4944

General Information
: Introduces you to the High Level Assembler product by describing what it does and which of your data processing needs it can fill. It is designed to help you evaluate High Level Assembler for your data processing operation and to plan for its use.

Installation and Customization Guide
: Contains the information you need to install and customize, and diagnose failures in, the High Level Assembler product.

  The diagnosis section of the book helps users determine if a correction for a similar failure has been documented previously. For problems not documented previously, the book helps users to prepare an APAR. This section is for users who suspect that High Level Assembler is not working correctly because of some defect.

Programmer's Guide
: Describes how to assemble, debug, and run High Level Assembler programs.

Language Reference
: Presents the rules for writing assembler language source programs to be assembled using High Level Assembler.

Licensed Program Specifications
: Contains a product description and product warranty information for High Level Assembler.

Online Publications

The High Level Assembler publications are available in the following softcopy format: *Application Development Collection Kit CD-ROM*, SK2T-1237.

To receive this collection kit you must place an order with IBM, specifying feature code 7526. The collection kit is shipped free of charge.

Related Publications

See "Bibliography" on page 331 for a list of publications that supply information you might need while using High Level Assembler.

Syntax Notation

Throughout this book, syntax is described using the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►── symbol indicates the beginning of a statement.

  The ───► symbol indicates that the statement syntax is continued on the next line.

  The ►─── symbol indicates that a statement is continued from the previous line.

  The ──►◄ indicates the end of a statement.

  Diagrams of syntactical units other than complete statements start with the ►─── symbol and end with the ───► symbol.

- **Keywords** appear in uppercase letters (for example, ASPACE). They must be spelled exactly as shown.

  Variables appear in all lowercase letters in a special typeface (for example, *integer*). They represent user-supplied names or values.

- If punctuation marks, parentheses, or such symbols are shown, they must be entered as part of the syntax.

- Required items appear on the horizontal line (the main path).

```
►►──INSTRUCTION──required item──────────────────────────────────►◄
```

- Optional items appear below the main path. If the item is optional and is the default, the item appears above the main path.

```
                ┌─default item──┐
►►──INSTRUCTION─┼───────────────┼───────────────────────────────►◄
                └─optional item─┘
```

- When you can choose from two or more items, they appear vertically in a stack.

  If you **must** choose one of the items, one item of the stack appears on the main path.

```
►►──INSTRUCTION──┬─required choice1─┬──────────────────────────►◄
                 └─required choice2─┘
```

  If choosing one of the items is optional, the whole stack appears below the main path.

```
►►──INSTRUCTION──┬──────────────────┬──────────────────────────►◄
                 ├─optional choice1─┤
                 └─optional choice2─┘
```

- An arrow returning to the left above the main line indicates an item that can be repeated. When the repeat arrow contains a separator character, such as a comma, you must separate items with the separator character.

```
                 ┌─,─────────────┐
►►──INSTRUCTION──▼─repeatable item─┴───────────────────────────►◄
```

  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

The following example shows how the syntax is used.

Format

```
            A                  B           C

                                        ┌─,───────┐
►►──┬───────────────┬──INSTRUCTION──────▼─┤ 1 ├───┴──►◄
    └─optional item─┘

1 :
├───┬─operand choice1────────┬───┤
    ├─operand choice2─(1)────┤
    └─operand choice3────────┘
```

Note:

1 *operand choice2* and *operand choice3* must not be specified together

A	The item is optional, and can be coded or not.
B	The INSTRUCTION key word must be specified and coded as shown.
C	The item referred to by **1** is a required operand. Allowable choices for this operand are given in the fragment of the syntax diagram shown below **1** at the bottom of the diagram. The operand can also be repeated. That is, more than one choice can be specified, with each choice separated by a comma.

Summary of Changes

Date of Publication: March 1995

Form of Publication: Second Edition (Revision), SC26-4941-01

Installation and Customization

The following enhancements improve installation and customization:

VM Serviceability Enhancements Staged/Extended (VMSES/E): VMSES/E is used to install the High Level Assembler under the CMS component of VM/ESA.

New Installation and Customization Guide: A new *Installation and Customization Guide* combines improved installation and customization procedures with the procedures for diagnosing failures in the High Level Assembler. It also contains chapters about planning to install and customize, and how to maintain, the High Level Assembler in MVS and CMS.

MVS/ESA Procedures: The names of the supplied MVS/ESA procedures have changed to adhere to product element naming standards.

VM/ESA Logical Saved Segment: Most High Level Assembler modules can be placed in a logical saved segment under VM/ESA.

Default Assembler Options: The default assembler options provided with High Level Assembler have changed to reflect the prominent features contained in the language, and concur with user demand.

Performance and Usability

The following enhancements improve system performance and system usability:

Virtual Storage Constraint Relief: The High Level Assembler modules and data areas, with some exceptions, use 31-bit addressing.

Extended Object Format Support: The XOBJECT assembler option instructs the High Level Assembler to produce an extended object format data set. Refer to *DFSMS/MVS Version 1 Release 3 Program Management*, SC26-4916 for more information.

Associated Data Architecture: The structure and content of the ADATA records have been improved. A new ADATA instruction lets you generate user records during assembly of the source program.

User Exits: The EXIT assembler option now supports a user-supplied terminal I/O routine to be used in place of, or in addition to, the terminal processing of the High Level Assembler.

Sample Exits: High Level Assembler provides sample I/O exits to complement the assembler's output processing for the following:

ADATA This exit provides you with an interface to the assembler that lets you write your own filter routines to examine and extract information from ADATA records.

LISTING This exit lets you selectively print parts of the assembler listing.

SOURCE This exit lets you read source statements from variable-length input data sets.

System-Determined Blocksize: The High Level Assembler supports the system-determined blocksize (SDB) feature of MVS/DFP* in the MVS/ESA environment. SDB allows the blocksize for all output datasets, except SYSLIN and SYSPUNCH, to be set to the optimum, system-determined value.

Operation Code Table for VSE: A new selectable operation code table assists in migration from the DOS/VSE Assembler to High Level Assembler.

Programmer Productivity

The following enhancements simplify program development and increase programmer productivity:

Source Program Assembler Options: You can use the new *PROCESS statement to specify selected assembler options in the assembler source program.

Profile Option: The PROFILE assembler option lets you specify the name of a library member containing assembler source statements which the assembler inserts at the start of the assembler source program. The statements in the member are processed after any ICTL instruction or *PROCESS statements.

Translating Character Constants and Literals: The TRANSLATE assembler option lets you specify a translation table that the assembler uses to convert characters contained in character (C-type) data constants (DCs) and literals. You can use the ASCII translation table provided with High Level Assembler or use your own translation table.

Record Numbers: High Level Assembler now provides an absolute record number and a relative record number for each record that is read from an input data set, or written to an output data set. These record numbers are passed to any user exit you specify for the assembly. When you specify the new FLAG(RECORD) assembler option the relative record number is printed in information messages, and is also written to the associated data file in the source analysis record.

Built-In Functions for Conditional Assembly Instructions: High Level Assembler provides new built-in functions for conditional assembly instructions that perform logical, arithmetic, and character string operations in SETA, SETB and SETC expressions.

External Function Calls: The new SETAF and SETCF instructions let you supply your own routines to perform external functions for conditional assembly, and place the results in a variable (SET) symbol. You may write these routines in any programming language that conforms to standard OS Linkage conventions.

System Variable Symbols: There is a new system variable for the data set name, volume serial number, and member name of all High Level Assembler output data sets.

Diagnostic Information

Some of the new diagnostic and listing enhancements are:

- Enhanced statement continuation checking
- Informational messages to help you locate the original source statement that generated diagnostic messages
- Assembler listing headings printed in either uppercase English, or a mixture of uppercase and lowercase English
- Assembler diagnostic messages produced in English, German, Japanese, or Spanish
- An *Unreferenced Symbols Defined in CSECTs* section of the assembler listing
- An *Extended Source and Object* section of the assembler listing to support 31-bit addresses
- Eight-character address and length fields to support 31-bit addresses
- Six-digit statement numbers with leading zeroes suppressed
- A *Macro and Copy Code Cross Reference* section of the assembler listing
- A new compact format of the *Ordinary Symbol and Literal Cross Reference* section of the assembler listing
- Improved page-break handling in conjunction with the EJECT, SPACE and TITLE assembler instructions

Part 1. Understanding and Using the Assembler

Chapter 1. Introduction

IBM High Level Assembler for MVS & VM & VSE is an IBM licensed program that can be used to assemble assembler language programs that use the following machine instructions:

System/370*
System/370 Extended Architecture (370-XA)
Enterprise Systems Architecture/370* (ESA/370*)
Enterprise Systems Architecture/390* (ESA/390*) machine instructions.

Requirements

This section describes the operating systems, the processors, and the amount of storage required to run High Level Assembler.

System Requirements

High Level Assembler runs under the operating systems listed below. Unless otherwise stated, the assembler also operates under subsequent versions, releases, and modification levels of these systems:

MVS/ESA SP Version 4
MVS/ESA SP Version 5
VM/ESA Release 1 (370 feature) running:
  CMS 7
VM/ESA Release 1 (ESA* feature) running:
  CMS 8
VM/ESA Release 2 running:
  CMS 9
  CMS 10
  CMS 11

High Level Assembler supports the operation codes available with the Extended Architecture (370-XA) mode processor and Enterprise System Architecture/370 (ESA/370*) or Enterprise System Architecture/390 (ESA/390*) mode processors and the new operation codes available with Enterprise System/9000* (ES/9000*) mode processors.

Machine Requirements

For assembling High Level Assembler programs: Programs written using High Level Assembler can be assembled, including use of the Extended Architecture mode processor machine instructions and Enterprise System Architecture mode processor machine instructions, on all System/370 family and its follow-on machines supporting the following operating systems:

MVS/ESA
VM/ESA

You might require an operating system-specific macro library to assemble programs that run under that operating system, depending on macro usage.

For running High Level Assembler programs: A generated object program using Extended Architecture (370-XA), Enterprise Systems Architecture/370 (ESA/370), Enterprise Systems Architecture/390 (ESA/390), Enterprise Systems/9000 (ES/9000) or Vector instructions can be run only on an applicable processor under an operating system that provides the necessary architecture support for the instructions used.

Tape device: High Level Assembler is distributed on one of the following:

Standard labeled 9-track magnetic tape written at 1600 or 6250 bpi
3480 tape cartridge
1/4 inch tape cartridge (VM only)

An appropriate tape device is required for installation.

Double-byte data: Double-byte data can be displayed, entered, or both, in their national language representation on the following:

IBM 3800-8 system printer
IBM 3200 system printer
IBM 3820 remote printer
IBM PS/55* family as an IBM 3270 terminal

Storage Requirements

Virtual storage: High Level Assembler requires a minimum of 520K bytes of main storage. 320K bytes of storage are required for High Level Assembler load modules. The rest of the storage allocated to the assembler is used for assembler working storage.

Auxiliary storage space: Depending on the assembler options used, auxiliary storage space might be required for the following data sets:

System input
Macro instruction library—either system or private or both
An intermediate work file, which must be a direct-access device such as 3350, 3380, 3390.
Print output
Object module output
Associated data output

Library space: The space requirements for the High Level Assembler load modules (or phases) and procedures are provided in the *Installation and Customization Guide*.

Installation: Please refer to *Installation and Customization Guide* for installation requirements.

Compatibility

This section describes source program compatibility and migration issues that you need to consider before using High Level Assembler.

Assembler Language Support

The assembler language supported by High Level Assembler has functional extensions to the languages supported by Assembler H Version 2 and the DOS/VSE Assembler. High Level Assembler uses the same language syntax, function, operation, and structure as these earlier assemblers. The functions provided by the Assembler H Version 2 macro facility are all provided by High Level Assembler.

Migration Considerations

Source Programs: Migration from High Level Assembler Release 1, Assembler H Version 2 or DOS/VSE Assembler to High Level Assembler Release 2 requires an analysis of existing assembler language programs to ensure that they do not contain macro instructions with names that conflict with the High Level Assembler Release 2 symbolic operation codes, or SET symbols with names that conflict with the names of High Level Assembler Release 2 system variable symbols.

With the exception of these possible conflicts, and with appropriate High Level Assembler option values, assembler language source programs written for High Level Assembler Release 1, Assembler H Version 2 or the DOS/VSE Assembler, that assemble without warning or error diagnostic messages, should assemble correctly using High Level Assembler Release 2.

Object Programs: Object programs generated by High Level Assembler Release 2 in any one of the supported operating systems can be migrated to any other of the supported operating systems for execution.

The object programs being migrated must be link-edited in the target operating system environment before execution.

You should be aware of the differences in the code generated by system macros in the supported operating systems. Operational facilities available on the source operating system but not available on the target operating system should not be specified for any program which is required to be compatible, either during assembly or link-edit.

Chapter 2. Using the Assembler Listing

This chapter tells you how to interpret the printed listing produced by the assembler. The listing is obtained only if the option LIST is in effect. Parts of the listing can be suppressed by using other options; for information on the listing options, refer to Chapter 3, "Controlling your Assembly with Options" on page 34.

The High Level Assembler listing consists of up to eleven sections, ordered as follows:

- *High Level Assembler Option Summary*
- *External Symbol Dictionary* (ESD)
- *Source and Object*
- *Relocation Dictionary* (RLD)
- *Ordinary Symbol and Literal Cross Reference*
- *Unreferenced Symbols Defined in CSECTs*
- *Macro and Copy Code Source Summary*
- *Macro and Copy Code Cross Reference*
- *DSECT Cross Reference*
- *USING Map*
- *Diagnostic Cross Reference and Assembler Summary*

The following assembler options are used to control the format, and which sections to produce, of the assembler listing:

ASA Allows you to use American National Standard printer control characters, instead of machine printer control characters.

DXREF Produces the *DSECT Cross Reference* section.

ESD Produces the *External Symbol Dictionary* section.

EXIT(PRTEXIT(*mod3***))**
Allows you to supply a listing exit to replace or complement the assembler's listing output processing.

LANGUAGE
Produces error diagnostic messages in the following languages:

- English mixed case (EN)
- English uppercase (UE)
- German (DE)
- Japanese (JP)
- Spanish (ES)

When you select either of the English languages, the assembler listing headings are produced in the same case as the diagnostic messages.

When you select either the German language or the Spanish language, the assembler listing headings are produced in mixed case English.

When you select the Japanese language, the assembler listing headings are produced in uppercase English.

The assembler uses the installation default language for messages produced in CMS by the ASMAHL command.

LINECOUNT
Allows you to specify how many lines should be printed on each page.

LIST Controls the format of the *Source and Object* section of the listing. NOLIST suppresses the entire listing.

MXREF Produces one, or both, of the *Macro and Copy Code Source Summary* and *Macro and Copy Code Cross Reference* sections.

PCONTROL
Controls which statements are printed in the listing, and overrides some PRINT instructions.

RLD Produces the *Relocation Dictionary* section.

USING(MAP)
Produces the *Using Map* section.

XREF Produces one, or both, of the *Ordinary Symbol and Literal Cross Reference* and the *Unreferenced Symbols Defined in CSECTs* sections.

The following additional options can be specified when you run the assembler in CMS:

LINECOUN
An abbreviation of the LINECOUNT option.

PRINT The assembler listing is written to the virtual printer instead of to a disk file.

The sections in the listing are described on the following pages.

High Level Assembler Option Summary

High Level Assembler provides a summary of the options current for the assembly, including:

- A list of the overriding parameters specified when the assembler was called
- The options specified on *PROCESS statements
- In-line error diagnostic messages for any overriding parameters and *PROCESS statements in error

You cannot suppress the option summary unless you suppress the entire listing, or you supply a user exit to control which lines are printed. High Level Assembler provides a sample LISTING exit that allows you to suppress the the option summary or print it at the end of the listing. See Appendix J, “Sample LISTING User Exit” on page 322.

Figure 2 shows an example of the *High Level Assembler Option Summary*. The example includes assembler options that have been specified in the invocation parameters and in *PROCESS statements. It also shows the *PROCESS statements in the *Source and Object* section of the listing.

```
                                       High Level Assembler Option Summary                                    Page    1
                                                                                                      1           2
                                                                                            HLASM R2.0  1995/03/24 08.00
  Overriding Parameters-  NOOBJECT,language(en),size(4meg),xref(short,unrefs)
  Process Statements- ALIGN
                      noDBCS
                      MXREF(FULL),noLIBMAC
                      FLAG(0)
                      noFOLD,LANGUAGE(ue)
                      NORA2
                      NODBCS
                      XREF(FULL)
     3
ASMA400W ** WARNING ** Error in invocation parameter - size(4meg)
ASMA422N ** WARNING ** Option LANGUAGE(ue) is not valid on a *PROCESS statement

  Options for this Assembly
   NOADATA
     ALIGN
   NOASA
   NOBATCH
   NOCOMPAT
   NODBCS
   NODECK
     DXREF
   NOESD
   NOEXIT
     FLAG(0,ALIGN,CONT,RECORD,NOSUBSTR)
   NOFOLD
     LANGUAGE(EN)
   NOLIBMAC
     LINECOUNT(60)
     LIST(121)
     MXREF(FULL)
     OBJECT
     OPTABLE(UNI)
   NOPCONTROL
   NOPESTOP
   NOPROFILE
   NORA2
   NORENT
     RLD
     SIZE(MAX)
     SYSPARM()
     TERM
   NOTEST
   NOTRANSLATE
   NOUSING
   NOXOBJECT
     XREF(SHORT,UNREFS)

   4
  No Overriding DD Names
                                                                                                              Page    2
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R2.0  1995/03/24 08.00
                                           5
                                      1 *PROCESS ALIGN
                                      2 *process noDBCS          any text here is a comment
                                      3 *process MXREF(FULL),noLIBMAC
                                      4 *PROCESS FLAG(0)
                                      5 *process noFOLD,LANGUAGE(ue)
                                      6 *PROCESS NORA2
                                      7 *PROCESS NODBCS
                                      8 *PROCESS XREF(FULL)
000000                                9 A        CSECT
                 R:F  00000          10          USING *,15
```

*Figure 2. Option Summary including options specified on *PROCESS statements*

The highlighted numbers in the example are:

1 Shows the product description at the top of each page of the assembler listing. (You can use the TITLE instruction to generate individual headings for each page of the source and object program listing.)

2 Shows the date and the time of the assembly.

3 Error diagnostic messages for overriding parameters and *PROCESS statements are shown immediately following the list of *PROCESS statement options.

4 If the assembler has been called by a program (see “Invoking the Assembler Dynamically” on page 134) and any standard (default) ddnames have been overridden, both the default ddnames and the overriding ddnames are listed. Otherwise, this statement appears:

```
No Overriding DD Names
```

5 The *PROCESS statements are written as comment statements in the *Source and Object* section.

External Symbol Dictionary (ESD)

This section of the listing contains the external symbol dictionary information passed to the linkage editor or loader, or DFSMS/MVS* binder, in the object module.

This section helps you find references between modules in a multimodule program. The ESD may be particularly helpful in debugging the running of large programs constructed from several modules.

The ESD entries describe the control sections, external references, and entry points in the assembled program. There are eight types of ESD entries (SD, ED, LD, ER, PC, CM, XD, and WX). Figure 3 shows the ESD entries when you specify the OBJECT or DECK option. Figure 4 shows the ESD entries when you specify the XOBJECT option. For each of the different types of ESD entries, the Xs indicate which of the fields have values.

Figure 3. Types of ESD Entries when OBJECT or DECK Option Specified

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID	FLAGS
X	SD	X	X	X	-	X
X	LD	-	X	-	X	-
X	ER	X	-	-	-	-
-	PC	X	X	X	-	X
X	CM	X	X	X	-	X
X	XD	X	X	X	-	-
X	WX	X	-	-	-	-

Figure 4. Types of ESD Entries when XOBJECT Option Specified

SYMBOL	TYPE	ID	ADDR	LENGTH	LD ID	FLAGS
X	SD	X	-	-	-	-
X	ED	X	X	X	X	X
X	LD	X	X	-	X	X
X	ER	X	-	-	X	-
X	CM	X	X	-	X	X
X	XD	X	X	X	-	-
X	WX	X	-	-	X	-

Figure 5 is an example of the *External Symbol Dictionary*, and is followed by a description of its contents.

```
                                              External Symbol Dictionary                                       Page    2
 1         2      3      4         5          6      7      8
Symbol   Type   Id     Address   Length     LD ID  Flags Alias-of                      HLASM R2.0  1995/03/24 08.00
SAMP01    SD 00000001 00000000 0000018C              00
ENTRY1    LD          00000000          00000001
CJMXTRN   WX 00000002
COMMAREA  XD 00000003 00000007 0000002D
RGETDTE   ER 00000004                                   RCNVDTE
RCNVTME   ER 00000005
```

Figure 5. External Symbol Dictionary Listing

1 **SYMBOL:** The name of every external dummy section, control section, entry point, and external symbol. If the external dummy section, control section, entry point or external symbol has a corresponding ALIAS instruction, the symbol shows the operand of the ALIAS instruction.

When you specify the XOBJECT assembler option, the assembler generates an entry type of ED with a symbol name of B_TEXT.

2 **TYPE:** The type designator for the entry, as shown in the table:

SD Control section definition. The symbol appeared in the name field of a START, CSECT or RSECT instruction.

LD Label definition. The symbol appeared as the operand of an ENTRY statement.

When you specify the XOBJECT assembler option, the assembler generates an entry type of LD for each CSECT and RSECT.

ER External reference. The symbol appeared as the operand of an EXTRN statement, or was declared as a V-type address constant.

PC Unnamed control section definition (private code). A CSECT, RSECT, or START statement that commences a control section that does not have a symbol in the name field, or a control section that is commenced (by any instruction which affects the location counter) before a CSECT, RSECT or START.

When you specify the XOBJECT assembler option, the assembler does not generate an entry type of PC. For private code, the assembler creates an SD entry type with a blank name.

CM Common control section definition. The symbol appeared in the name field of a COM statement.

XD External dummy section. The symbol appeared in the name field of a DXD statement or a Q-type address constant. (The external dummy section is also called a pseudo register in the applicable *Linkage Editor and Loader* manual, and *DFSMS/MVS Program Management* manual.)

WX Weak external reference. The symbol appeared as an operand in a WXTRN statement.

3 **ID:** The external symbol dictionary identification number (ESDID). The number is a unique 8-digit hexadecimal number identifying the entry. It is used in combination with the LD entry of the ESD and in the relocation dictionary for referencing the ESD.

4 **ADDR:** The address of the symbol (in hexadecimal notation) for SD- and LD-type entries, and blanks for ER- and WX-type entries. For PC- and CM-type entries, it indicates the beginning address of the control section. For XD-type entries, it indicates the alignment by printing a number one less than the number of bytes in the unit of alignment. For example, 7 indicates doubleword alignment.

5 **LENGTH:** The assembled length, in bytes, of the control section (in hexadecimal notation).

6 **LD ID:** For an LD-type entry, the ESDID of the control section in which the symbol was defined.

7 **FLAGS:** For SD-, PC-, and CM-type entries, this field contains the following flags:

```
Bit 4:    0  = Section is not an RSECT
          1  = Section is an RSECT
Bit 5:    0  = RMODE is 24
          1  = RMODE is ANY
Bits 6-7: 00 = AMODE is 24
          01 = AMODE is 24
          10 = AMODE is 31
          11 = AMODE is ANY
```

8 **ALIAS-OF:** When symbol **1** is defined in an ALIAS instruction, this field shows the external symbol name of which symbol **1** is an alias.

Source and Object

This section of the listing documents the source statements of the module and the resulting object code.

This section is the most useful part of the listing because it gives you a copy of all the statements in your source program (except listing control statements) exactly as they are entered into the machine. You can use it to find simple coding errors, and to locate and correct errors detected by the assembler. By using this section with the *Ordinary Symbol and Literal Cross Reference* section, you can check that your branches and data references are in order. The location-counter values and the object code listed for each statement help you locate any errors in a storage dump. Finally, you can use this part of the listing to check that your macro instructions have been expanded properly.

The assembler can produce two formats of the *Source and Object* section: a 121-character format and a 133-character format. To select one, you must specify either the LIST(121) assembler option or the LIST(133) assembler option. Both sections show the source statements of the module, and the object code of the assembled statements.

The 133-character format shows the location counter, and the first and second operand addresses (ADDR1 and ADDR2) as 8-byte fields in support of 31-bit addresses. This format is required when producing the extended object format data set (see “XOBJECT” on page 64). The 133-character format also contains the first eight characters of the macro name in the identification-sequence field for statements generated by macros.

High Level Assembler lets you write your program, and print the assembler listing headings in mixed-case. Diagnostic messages are printed in the language you specify in the LANGUAGE assembler option described on “LANGUAGE” on page 45.

Figure 6 shows an example of the *Source and Object* section in 121-character format, and in mixed-case.

```
                                                                                                                     Page    3
1        2
SAMP01   Sample Listing Description
  Active Usings: None
  3    4               5         6          7                                                    8          9
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                     HLASM R2.0  1995/03/24 08.00
000000                              2 Samp01   Csect                                                               00002000
                                    3          Sav   (14,12)                  Save caller's registers             00003000
10 ASMA057E *** ERROR *** Undefined operation code - Sav
11 ASMA435I Record 3 in 'ATR010.SAMPLE.SOURCE(SAMP01)' on volume: ATR010
                                               .
                                               .
                                               .
000000                             21 Entry1   DS    0H                                                            00021000
                                   22          Sampmac1 Parm1=YES                                                  00022000
000000 18CF                        23+Label1   LR 12,15                                                            01-SAMPM
                  12
                 R:C  00000        24+         USING Entry1,12                Ordinary Using                      01-SAMPM
000002 0000 0000            00000  25+         LA Savearea,10                                                      01-SAMPM
   ASMA044E *** ERROR *** Undefined Symbol - Savearea
   ASMA029E *** ERROR *** Incorrect Register or Mask Specification
13 ASMA435I Record 10 in 'USER.MACLIB(SAMPMAC1)' on volume: ATR010
000006 50D0 A004            00004  26+         ST 13,4(,10)                                                        01-SAMPM
00000A 50A0 D008            00008  27+         ST 10,8(,13)                                                        01-SAMPM
00000E 18DA                        28+         LR 13,10                                                            01-SAMPM
                                                                                                                 14
               R:A35  0000E        29+         USING *,10,3,5                 Ordinary Using,Multiple Base        01-SAMPM
                                               .
                                               .
                                               .
                                   40+         DROP 10,3,5                    Drop Multiple Registers             01-SAMPM
                                   41          COPY  SAMPLE                                                        00041000
                                        15
                                   42C*        Line from member SAMPLE
                      16
               C 894  00000 00894  43          Using IHADCB,INDCB             Establish DCB addressability        00024000
               C 8F4  00000 008F4  44 ODCB     Using IHADCB,OUTDCB                                                 00025000
                 R:2  00000        45 PlistIn  Using Plist,2                  Establish Plist addressability      00026000
                 R:3  00000        46 PlistOut Using Plist,3                                                       00027000

                                               .
                                               .
                                               .
                                                                                                                     Page    4
SAMP01   Sample Listing Description
17  Active Usings: Entry1,R12  IHADCB,R12+X'894'  PlistIn.Plist,R2  PlistOut.Plist,R3  ODCB.IHADCB,R12+X'8F4'

  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                     HLASM R2.0  1995/03/24 08.00

000010 1851                        47 ?Branch  LR    R5,R1                    Save Plist pointer                  00028000
   ASMA147E *** ERROR *** Symbol too long, or first character not a letter - ?Branch
   ASMA435I Record 26 in 'ATR010.SAMPLE.SOURCE(SAMP01)' on volume: ATR010
000012 5825 0000            00000  48          L     R2,0(R5)                 R2 = address of request list        00029000
000016 47F0 C01A            0001A  49          B     Open                                                          00030000
                                               .
                                               .
                                               .
                                  813          End                                                                 00792000
000958 00000001                   814                =F'1'
00095C 00000000                   815                =V(Rcnvdte)
000960 00000000                   816                =V(Rcnvtme)
000964 00000002                   817                =F'2'
```

Figure 6. Source and Object listing section - 121 format

1 **SAMP01:** The 1- to 8-character deck identification, if any. It is obtained from the name field of the first named TITLE statement. The assembler prints the deck identification and date on every page of the listing, except the Options Summary.

2 **Sample Listing Description:** The information taken from the operand field of a TITLE statement.

3 **Loc:** Is the location counter value that represents the assembled address (in hexadecimal notation) of the object code.

- For ORG statements, the location-counter value before the ORG is placed in the location column and the location counter value after the ORG is placed in the ADDR2 field.
- If the END statement contains an operand, the operand value (transfer address) appears in the location field (LOC).
- In the case of LOCTR, COM, CSECT, RSECT, and DSECT statements, the location field contains the current address of these control sections.
- In the case of EXTRN, WXTRN, ENTRY, and DXD instructions, the location field and object code field are blank.
- For LTORG statements, the location field contains the location assigned to the literal pool.

If, at the time of the page eject, the current control section being assembled is a COM section, the heading line starts with C-LOC. If, at the time of the page eject, the current control section being assembled is a DSECT, the heading line starts with D-LOC. If, at the time of the page eject, the current control section being assembled is an RSECT, the heading line starts with R-LOC.

4 **Object Code:** The object code produced by the source statement. The entries are always left-justified. The notation is hexadecimal. Entries are machine instructions or assembled constants. Machine instructions are printed in full with a blank inserted after every 4 digits (2 bytes). Only the first 8 bytes of a constant appears in the listing if PRINT NODATA is in effect, unless the statement has continuation records. The whole constant appears if PRINT DATA is in effect. (See the PRINT assembler instruction in the *Language Reference*.)

This field also shows the base registers for ordinary USING instructions, and the base register and displacement for dependent USING instructions. See **12** and **16** for more details.

5 **Addr1 Addr2:** Effective addresses (each the result of adding a base register value and a displacement value):

- The field headed ADDR1 contains the effective address for the first operand of an instruction (if applicable). It also contains the value of the first operand of a USING instruction. See **12** and **16** for more details.
- The field headed ADDR2 contains the effective address of the last operand of any instruction referencing storage.
  - For a USING instruction, the ADDR2 field contains the value of the second operand. See **16** for more details.
  - For an EQU instruction, the ADDR2 field contains the value assigned.

Both address fields contain 6 digits; however, if the high-order digit is a 0, it is not printed. For USING and EQU instructions, the ADDR2 field may contain up to 8 digits.

6 **Stmt:** The statement number. A plus sign (+) to the right of the number indicates that the statement was generated as the result of macro call processing. An unnumbered statement with a plus sign (+) is the result of open code substitution.

A minus sign (-) to the right of the statement number indicates that the statement was read by a preceding AREAD instruction.

7 **Source Statement:** The source program statement. The following items apply to this section of the listing:

- Source statements are listed, including those brought into the program by the COPY assembler instruction, and including macro definitions submitted with the main program for assembly. Listing control instructions are not printed, except for PRINT, which is printed unless the NOPRINT operand is specified.

- Macro definitions obtained from a library are not listed, unless the macro definition is included in the source program by means of a COPY statement, or the LIBMAC assembler option was specified.

- The statements generated as the result of a macro instruction follow the macro instruction in the listing, unless PRINT NOGEN is in effect. If PRINT GEN is in effect and PRINT NOMSOURCE is specified, the printing of the source statements generated during macro processing and conditional assembly substitution is suppressed, without suppressing the printing of the generated object code of the statements. If PRINT MCALL is in effect, nested macro instructions including all parameters are printed. When the PRINT NOGEN instruction is in effect, the assembler prints one of the following on the same line as the macro call or model statement:

  – The object code for the first instruction generated
  – The first 8 bytes of generated data from a DC instruction

  When the assembler forces alignment of an instruction or data constant, it generates zeros in the object code and prints the generated object code in the listing. When you use the PRINT NOGEN instruction the generated zeros are not printed.

  Diagnostic Messages and Generated Data: If the next line to print after a macro call or model statement is a diagnostic message, the generated data is not shown.

- Assembler and machine instructions in the source program that contain variable symbols are listed twice: as they appear in the source input, and with values substituted for the variable symbols.

- All error diagnostic messages appear in line except those suppressed by the FLAG option. Chapter 6, “Diagnosing Assembly Errors” on page 123 describes how error messages and MNOTEs are handled.

- Literals that have not been assigned locations by LTORG statements appear in the listing following the END statement. Literals are identified by the equal sign (=) preceding them.

- Whenever possible, a generated statement is printed in the same format as the corresponding macro definition (model) statement. The starting columns of the operation, operand, and comments fields are preserved, unless they are displaced by field substitution, as shown in Figure 7 on page 16.

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                      1 &A SETC 'abcdefghijklmnop'
                                      2 &A       LA   4,1      Comment
000000 4140 0001            00001      +abcdefghijklmnop LA 4,1
                                       +                       Comment
                                      3 &b SETC 'abc'
                                      4 &b       LA   4,1      Comment
000004 4140 0001            00001      +abc      LA   4,1      Comment
```

Figure 7. Source and Object listing section

It is possible for a generated statement to occupy ten or more continuation lines on the listing. In this way, generated statements are unlike source statements, which are restricted to nine continuation lines.

8 **HLASM R2.0:** The release level of High Level Assembler.

9 **1995/03/24 08.00:** The date and time at the start of the assembly.

10 **ASMA057E:** The error diagnostic messages immediately following the source statement in error. Many error diagnostic messages include the segment of the statement that is in error. You can use the FLAG assembler option to control the level of diagnostic messages displayed in your listing.

11 **ASMA435I:** The informational message, ASMA435I, that describes the origin of the source statement in error. This message is only printed when you specify the FLAG(RECORD) assembler option.

12 **R:C:** The Addr1 and Addr2 columns show the first and second operand addresses in the USING instructions. The base registers on an ordinary USING instruction are printed, right justified in the object code columns, preceded by the characters `R:`.

13 **ASMA435I:** The informational message, ASMA435I, that describes the origin of the source statement in error. Conditional assembly statements and comment statements contribute to the record count of macro definitions, as suggested by the record number (10) in this message which is greater than the number of generated statements.

14 **01-SAMPM:** The identification-sequence field from the source statement. For a macro-generated statement, this field contains information identifying the origin of the statement. The first two columns define the level of the macro call, where a level of `01` indicates statements generated by the macro specified within the source code, and higher level numbers indicate statements generated from macros invoked from within a macro.

For a library macro call, the last five columns contain the first five characters of the macro name. For a macro whose definition is in the source program (including one read by a COPY statement or by the LIBMAC assembler option), the last five characters contain the line number of the model statement in the definition from which the generated statement is derived. This information can be an important diagnostic aid in analyzing output resulting from macro calls within macro calls.

15 **C:** Source statements included by COPY instructions have the character `C` suffixed to the statement number.

16 **C 894:** The Addr1 and Addr2 columns show the first and second operand addresses in the USING instructions. The resolved base displacement for a dependent USING instruction is printed in the object code columns, as `register displacement`, where `register` is shown as a hexadecimal value.

17 **Active Usings:** If PRINT UHEAD or PCONTROL(UHEAD) has been specified, a summary of current active USINGs is printed on up to four heading lines, following the TITLE line on each page of the source and object section. The USINGs listed are those current at the end of the assembly of the last statement on the previous page of the listing, with the following exceptions:

- The USINGs summary shows the effect of the USING instruction when:
  - It is the first statement in the source input data set, or
  - It is the first statement on the new page, and the new page is not caused by an EJECT or SPACE instruction.
- The USINGs summary shows the effect of the DROP instruction when:
  - It is the first statement in the source input data set, or
  - It is the first statement on the new page, and the new page is not caused by an EJECT or SPACE instruction.

Current active USINGs include USINGs that are temporarily overridden. In the following example the USING for base register 12 temporarily overrides the USING for base register 10. After the DROP instruction the base register for BASE1 reverts to register 10.

```
USING BASE1,10
USING BASE1,12       Temporarily overrides register 10
LA    1,BASE1        Uses base register 12
DROP  12
LA    1,BASE1        Uses base register 10
```

The summary of active USINGs heading lines have the format:

`Active Usings:` *`label.sectname+offset,registers`*

where:

label The label name specified for a Labeled USING. If the USING is not labeled, this field is omitted.

sectname The section name used to resolve the USING. The section name is listed as `(PC)` if the section is an unnamed CSECT, `(COM)` if the section is unnamed COMMON, and `(DSECT)` if the section is an unnamed DSECT.

offset The offset from the specified section that is used to resolve the USING. This field is omitted if it is zero.

registers The register or registers specified on the USING statement.

For dependent USINGs, the register is printed as `register+offset` where `register` is the register used to resolve the address from the corresponding ordinary USING, and `offset` is the offset from the register to the address specified in the dependent USING.

If there are more active USINGs than can fit into four lines, the summary is truncated, and the characters `'MORE ...'` are appended to the last line.

In this example, the first is an ordinary USING, the second a dependent USING, the third and fourth are labeled USINGs and the last is a labeled dependent USING.

Figure 8 shows an example of the *Source and Object* section when the same assembly is run with assembler option LIST(133):

```
SAMP01   Sample Listing Description                                                                    Page    3
  Active Usings: None
   Loc  Object Code      Addr1    Addr2    Stmt   Source Statement                    HLASM R2.0  1995/03/24 08.00
00000000                                      2 Samp01   Csect                                                     00001000
                                              3          Sav   (14,12)              Save caller's registers        00002000
      ASMA057E *** ERROR *** Undefined operation code - Sav
      ASMA435I Record 3 in 'ATR010 SAMPLE SOURCE(SAMP01)' on volume: ATR010
                                                         .
                                                         .
                                                         .
00000000                                     21 Entry1   DS    0H                                                  00021000
                                             22          Sampmac1 Parm1=YES                                        00022000
00000000 18CF                                23+Label1   LR 12,15                                                  01-SAMPMAC1
                    R:C 00000000             24+         USING Entry1,12            Ordinary Using                 01-SAMPMAC1
00000002 0000 0000             00000000      25+         LA Savearea,10                                            01-SAMPMAC1
   ASMA044E *** ERROR *** Undefined Symbol - Savearea
   ASMA029E *** ERROR *** Incorrect Register or Mask Specification
      ASMA435I Record 10 in 'USER.MACLIB(SAMPMAC1)' on volume: ATR010
00000006 50D0 A004          00000004         26+         ST 13,4(,10)                                              01-SAMPMAC1
0000000A 50A0 D008          00000008         27+         ST 10,8(,13)                                              01-SAMPMAC1
0000000E 18DA                                28+         LR 13,10                                                  01-SAMPMAC1
                  R:A35 0000000E             29+         USING *,10,3,5             Ordinary Using,Multiple Base   01-SAMPMAC1
                                                .
                                                .
                                                .
                                             40+         DROP 10,3,5                Drop Multiple Registers        01-SAMPMAC1
                                             41          COPY  SAMPLE                                              00041000
                                             42C*        Line from member SAMPLE
                  C 894 00000000 00000894    43          Using IHADCB,INDCB         Establish DCB addressability   00024000
                  C 8F4 00000000 000008F4    44 ODCB     Using IHADCB,OUTDCB                                       00025000
                    R:2 00000000             45 PlistIn  Using Plist,2              Establish Plist addressability 00026000
                    R:3 00000000             46 PlistOut Using Plist,3                                             00027000
                                                         .
                                                         .
                                                         .
SAMP01   Sample Listing Description                                                                    Page    4
    Active Usings: Entry1,R12  IHADCB,R12+X'894'  PlistIn.Plist,R2  PlistOut.Plist,R3  ODCB.IHADCB,R12+X'8F4'

   Loc  Object Code      Addr1    Addr2    Stmt   Source Statement                    HLASM R2.0  1995/03/24 08.00

00000010 1851                                47 ?Branch  LR    R5,R1                Save Plist pointer             00028000
   ASMA147E *** ERROR *** Symbol too long, or first character not a letter - ?Branch
   ASMA435I Record 26 in 'ATR010 SAMPLE SOURCE(SAMP01)' on volume: ATR010
00000012 5825 0000             00000000      48          L     R2,0(R5)             R2 = address of request list   00029000
00000016 47F0 C01A             0000001A      49          B     Open                                                00030000
                                                         .
                                                         .
                                                         .
                                            813          End                                                       00792000
00000958 00000001                           814                =F'1'
0000095C 00000000                           815                =V(Rcnvdte)
00000960 00000000                           816                =V(Rcnvtme)
00000964 00000002                           817                =F'2'
```

Figure 8. Source and Object listing section - 133 format

1 **Loc:** The assembled address of the object code occupies 8 characters.

2 **Object Code:** The generated object code remains the same as when LIST(121) is specified.

3 **Stmt:** Shows the statement number, allowing up to 6 digits.

4 **Source Statement:** Shows the source statement.

5 **ASMA057E:** The error diagnostic messages immediately following the source statement in error.

6 **ASMA435I:** The informational message, ASMA435I, that describes the origin of the source statement in error.

7 **Addr1 Addr2:** The Addr1 and Addr2 columns show 8 character operand addresses.

8 **ASMA435I:** The informational message, ASMA435I, that describes the origin of the source statement in error. Conditional assembly statements and comment statements contribute to the record count of macro definitions, as suggested by the record number (10) in this message which is greater than the number of generated statements.

9 **01-SAMPMAC1:** The first 8 characters of the macro name are shown in the identification-sequence field.

10 **C:** Source statements included by COPY instructions have the character `C` suffixed to the statement number.

11 **C 894:** The Addr1 and Addr2 columns show 8 character operand addresses.

12 **Active Usings:** USING heading information showing active USINGs.

Relocation Dictionary (RLD)

This section of the listing describes the relocation dictionary information passed to the linkage editor or loader, or DFSMS/MVS binder, in the object module.

The entries describe the address constants in the assembled program that are affected by relocation. This section helps you find relocatable constants in your program.

```
                                                       Relocation Dictionary                                  Page    5
   1            2          3      4
 Pos.Id       Rel.Id      Flags  Address                                          HLASM R2.0  1995/03/24 08.00
00000001     00000002      1C    00000188
00000001     00000001      08    000008B5
00000001     00000004      1C    0000095C
00000001     00000005      1C    00000960
```

Figure 9. Relocation Dictionary (RLD) Listing

1 **POS.ID:** The external symbol dictionary ID number assigned to the ESD entry for the control section in which the address constant is used as an operand.

2 **REL.ID:** The external symbol dictionary ID number assigned to the ESD entry for the control section in which the referenced symbol is stored.

3 **FLAGS:** The 2-digit hexadecimal number represented by the characters in this field is interpreted as follows:

First Digit:

- 0 indicates that the entry describes an A-type or Y-type address constant
- 1 indicates that the entry describes a V-type address constant
- 2 indicates that the entry describes a Q-type address constant
- 3 indicates that the entry describes a CXD entry

Second Digit: The first three bits of this digit indicate the length of the constant and whether the base should be added or subtracted:

Bits 0 and 1	Bit 2	Bit 3
00 = 1 byte	0 = +	0
01 = 2 bytes	1 = –	0
10 = 3 bytes		0
11 = 4 bytes		0

4 **ADDRESS:** The assembled address (in hexadecimal notation) of the field where the address constant is stored.

Ordinary Symbol and Literal Cross Reference

This section of the listing concerns symbols and literals that are defined and used in the program. This is a useful tool in checking the logic of your program; it helps you see if your data references and branches are in order.

```
                                  Ordinary Symbol and Literal Cross Reference                                 Page    6
1            2         3          4     5  6        7    8
Symbol      Len      Value      Id     R Type     Defn  References                                HLASM R2.0  1995/03/24 08.00
$ASTER    00000001 0000005C FFFFFFFF A    U       1604
Exit      00000004 00000848 00000001      I       0069  59B, 65B
ExMVC     00000006 000000B8 00000000      I       1735  211X
                                                     .
                                                     .
                                                     .
RFCMJD    00000001 00001AB8 00000000      U       1635
RMMWRG    00000001 00000080 FFFFFFFF A    U     120411
R0        00000001 00000000 FFFFFFFF A    U       1581  35, 40M, 44M, 55, 60M, 64M, 74, 79M, 83M
R1        00000001 00000001 FFFFFFFF A    U       1583  39M, 59U, 78D, 90M, 91, 93, 93, 95M, 95
                                                     .
                                                     .
                                                     .
Z1        00000004 00000014 00000000 C    U         19  423
```

Figure 10. Ordinary Symbol and Literal Cross Reference

1 **Symbol:** Shows each symbol or literal. Symbols are shown in the form in which they are defined, either in the name entry of a machine or assembler instruction, or in the operand of an EXTRN or WXTRN instruction. Symbols defined using mixed-case letters are shown in mixed-case letters, unless the FOLD assembler option was specified.

If a symbol name is used as a literal more than once in a program, and the form of the symbol name is coded differently, for example `=V(symbol)` and `=V(SYMBOL)`, and the symbol is not defined in the program, the symbol is listed in the form of the first reference. In the following example the assembler lists the symbol name as `inPUT`, because the third statement is the first occurrence of the symbol, and the symbol is not defined.

```
test     csect
         using      *,15
         la         1,=a(inPUT)        third statement
         la         1,=a(INPUT)
         end
```

In the following example the assembler lists the symbol name as `INput`, because the symbol is defined in the fifth statement.

```
test     csect
         using     *,15
         la        1,=a(inPUT)         third statement
         la        1,=a(INPUT)
INput    dc        cl4' '              fifth statement
         END
```

2 **Len:** Shows, in decimal notation, the byte length of the field represented by the symbol. This field is blank for labeled USINGs.

3 **Value:** Shows the hexadecimal address that the symbol or literal represents, or the hexadecimal value to which the symbol is equated. This field is blank for labeled USING symbols.

4 **Id:** For symbols and literals defined in an executable control section or an external dummy section, this field shows the external symbol dictionary ID (ESDID) assigned to the ESD entry for the control section in which the symbol or literal is defined. For external symbols, this field indicates the ESDID assigned to ESD entry for this symbol. For symbols defined in a dummy control section, this field indicates the control section ID assigned to the control section. For symbols defined using the EQU statement, if the operand contains a relocatable expression, this field shows the external symbol dictionary ID of the relocatable expression. Otherwise, it contains the current control section ID.

5 **A:** Symbols `RMMWRG`, `R0` and `R1` are absolute symbols and are flagged **'A'**, in the R column. Symbol `Z1` is the result of a complex relocatable expression and is flagged **'C'**, in the R column. Symbol `RFCMJD` is simply relocatable and is not flagged. (Column title ‘R’ is an abbreviation for Relocatability Type)

6 **Type:** Indicates the type attribute of the symbol or literal.

7 **Defn:** Is the statement number in which the symbol or literal was defined.

8 **References:** Shows the statement numbers of the statements in which the symbol or literal appears as an operand. Additional indicators are suffixed to statement numbers as follows:

B The statement contains a branch instruction, and the relocatable symbol is used as the branch-target operand address.

D The statement contains a DROP instruction, and the symbol is used in the instruction operand.

M The instruction causes the contents of a register represented by an absolute symbol, or a storage location represented by one or more relocatable symbols to be modified.

U The statement contains a USING instruction, and the symbol is used in one of the instruction operands.

X The statement contains an `EX` machine instruction, and the symbol in the second operand is the symbolic address of the target instruction.

In the case of a duplicate symbol or literal, this column contains the message:

```
****DUPLICATE****
```

The following notes apply to the cross reference section:

Notes:

1. Cross reference entries for symbols used in a literal refer to the assembled literal in the literal pool. Look up the literals in the cross reference to find where the symbols are used.
2. A PRINT OFF listing control instruction does not affect the production of the cross reference section of the listing.
3. In the case of an undefined symbol, the columns Len, and Value contain the message:

   ```
   ****UNDEFINED****
   ```

Unreferenced Symbols Defined in CSECTs

This section of the listing shows symbols that have been defined in CSECTs but not referenced. This helps you remove unnecessary data definitions, and reduce the size of your program. The list of symbols are shown in symbol name order. The XREF(UNREFS) assembler option must be specified to obtain this section of the listing.

```
                              Unreferenced Symbols Defined in CSECTS                                         Page   12
  1      2                                                                     HLASM R2.0  1995/03/24 08.00
 Defn  Symbol

104401  $SYSDATE
  2001  ABC
    14  COMMON_WORK_AREA
   125  HERE_IS_A_LONG_SYMBOL_EQUAL_TO_THE_MAXIMUM_ALLOWED_LENGTH_OF_63
     1  LEN_EIGHT
   123  THIS_LONG_SYMBOL_IS_GREATER_THAN_FORTY_FIVE_BYTES
    94  VAR_UNUSED
  5401  ZSYSDATE
```

Figure 11. Unreferenced Symbols Defined in CSECTS

1 The statement number that defines the symbol.

2 The name of the symbol.

Macro and Copy Code Source Summary

This section of the listing shows the names of the macro libraries from which the assembler read macros or copy code members, and the names of the macros and copy code members that were read from each library. This section of the listing is useful for checking that you have included the correct version of a macro or copy code member.

```
                                      Macro and Copy Code Source Summary                                  Page   13
  1    2                                       3         4
 Con Source                                 Volume    Members                     HLASM R2.0  1995/03/24 08.00
     PRIMARY INPUT                                    CJMMAC
  L1 SYS1.MACLIB                            SYSRES    DCB       DCBD      IHBERMAC IHB01
  L2 USER.MACLIB                            ATR001    REGEQU
```

Figure 12. Macro and Copy Code Source Summary

1 **Con:** Contains a number representing the concatenation order of macro and copy code libraries. This number is not shown for `PRIMARY INPUT`. The number is prefixed with L which indicates Library. The concatenation value is cross referenced in the *Macro and Copy Code Cross Reference* section.

2 **Source:** Shows the name of each library from which the assembler read a macro or a copy code member or, for in-line macros, the words `PRIMARY INPUT`.

3 **Volume:** Shows the volume serial number of the volume on which the library resides.

4 **Members:** Shows the names of the macros or copy members that were retrieved from the library.

You can suppress this section of the listing by specifying the NOMXREF assembler option.

LIBRARY User Exit: If a LIBRARY user exit has been specified for the assembly, and the exit opens the library data set, the exit can return the name of the library to the assembler. In this case the *Macro and Copy Code Source Summary* lists the library names returned by the user exit.

Macro and Copy Code Cross Reference

This section of the listing shows the names of macros and copy code members and the statements where the macro or copy code member was called. Either assembler option, MXREF(XREF) or MXREF(FULL), generates this section of the listing.

```
                                             Macro and Copy Code Cross Reference                              Page   14
 1         2    3                4     5
Macro     Con  Called By        Defn  References                                         HLASM R2.0  1995/03/24 08.00
$EMAC1     L1  PRIMARY INPUT       -   2
$EXMAC     L1  PRIMARY INPUT       -   67
$SHOW      L1  PRIMARY INPUT       -   50
           L1  $EMAC1                  31
           L1  $EXMAC                  70
$SHOWCB    L1  PRIMARY INPUT       -   1781
CVT        L1  PRIMARY INPUT       -   163
COPYBOOK   P1  PRIMARY INPUT       -   154C  6
DCB        L3  PRIMARY INPUT       -   103
DCBD       L3  PRIMARY INPUT       -   1823
IHBERMAC   L3  DCB                 -   104
IHB01      L3  DCBD                    1824
MYMAC      P0  PRIMARY INPUT  122665   122669
TESTMAC    L1  $SHOW               -   41, 61, 80
```

Figure 13. Macro and Copy Code Cross Reference

1 **Macro:** The macro or copy code member name.

2 **Con:** Shows the value representing the input source concatenation, as listed in the *Macro and Copy Code Source Summary*, and under *Datasets Allocated for this Assembly* in the *Diagnostic Cross Reference and Assembler Summary* as shown in Figure 21 on page 30.

3 **Called By:** Shows either the name of the macro that calls this macro or copy code member, or `PRIMARY INPUT`, meaning that the macro or copy code member was called directly from the primary input source. If you use the COPY instruction to copy a macro definition, then references to the macro are shown as called by PRIMARY INPUT.

4 **Defn:** Is one of the following:

- The statement number for macros defined in the primary input file
- (–) meaning that the macro or copy code member was retrieved from a library.

5 **References:** The statement number that contains the macro call or COPY instruction.

Lookahead Processing: If a COPY instruction is encountered during lookahead, the reference number is the number of the statement that causes lookahead processing to commence.

PCONTROL(MCALL) Assembler Option: If you specify the PCONTROL(MCALL) assembler option, and you copy a macro definition from an inner macro, the reference number of the copied member is one less than the statement number containing the inner macro call instruction. See "Effects of LIBMAC and PCONTROL(MCALL) Options" for examples of assemblies using different combinations of the LIBMAC and PCONTROL(MCALL) options.

6 **C:** Statement numbers have a suffix of C when the reference is to a member on a COPY instruction.

The following example shows the *Macro and Copy Code Cross Reference* when assembler option LIBMAC is specified:

```
                                          Macro and Copy Code Cross Reference                                    Page   10
Macro     Con  Called By        Defn  References                                             HLASM R2.0  1995/03/24 08.00
                                   1
$EMAC1     L1  PRIMARY INPUT       2X  18
$EXMAC     L1  PRIMARY INPUT      67X  75
$SHOW      L1  PRIMARY INPUT      66X  76
           L1  $EMAC1              -   47
           L1  $EXMAC              -   86
$SHOWCB    L1  PRIMARY INPUT    3492X  3500
COPYBOOK   P0  PRIMARY INPUT       -   174C
CVT        L1  PRIMARY INPUT     163X  1874
DCB        L3  PRIMARY INPUT     103X  123
DCBD       L3  PRIMARY INPUT    3501X  3521
IHBERMAC   L3  DCB                 -   124
IHB01      L3  DCBD                -   3522
MYMAC      P0  PRIMARY INPUT  124363   124367
TESTMAC    L1  $SHOW               -   57, 87, 106
```

Figure 14. Macro and Copy Code Cross Reference - with LIBMAC option

1 **Defn:** Shows the 'X' flag to indicate the macro was read from a macro library and imbedded in the input source program immediately preceding the invocation of that macro. For example, $EMAC1 was called by the PRIMARY INPUT stream from LIBRARY L1, at statement number 18 after being imbedded in the input stream at statement number 2. See "Effects of LIBMAC and PCONTROL(MCALL) Options" for examples of assemblies using different combinations of the LIBMAC and PCONTROL(MCALL) options.

You can suppress this section of the listing by specifying the NOMXREF assembler option.

Effects of LIBMAC and PCONTROL(MCALL) Options

When you specify different combination of the LIBMAC and PCONTROL(MCALL) assembler options to assemble the same source program, the definition statement and reference statement numbers can be different in each assembly listing.

The example that follows, shows how these options affect the output from an assembly of the same source program. The source program is coded as follows:

```
MACOUTER
END
```

The assembly of this program uses the following library members:

MACOUTER: A macro definition that issues a call to macro MACINNER.

MACINNER: A macro definition that copies member COPYCODE.

COPYCODE: A member containing an MNOTE instruction.

Figure 15 shows the output when you specify the NOLIBMAC and NOPCONTROL options.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R2.0  1995/04/01 08.00

                                      1          MACOUTER
                                      2+*,MNOTE FROM MEMBER COPYCODE                                           02-MACIN
                                      3          END

                                          Macro and Copy Code Source Summary                                Page    3

 Con Source                                        Volume    Members                        HLASM R2.0  1995/04/01 08.00

  L1 MYMAC    MACLIB   A1                          MY-Z-D    COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                               Page    4

Macro     Con  Called By       Defn  References                                             HLASM R2.0  1995/04/01 08.00

COPYCODE   L1  MACINNER           -  1C
MACINNER   L1  MACOUTER           -  1
MACOUTER   L1  PRIMARY INPUT      -  1
```

Figure 15. Assembly with NOLIBMAC and NOPCONTROL options

Figure 16 shows the output when you specify the NOLIBMAC and PCONTROL(MCALL) options.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R2.0  1995/04/01 08.01

                                      1          MACOUTER
   This line produced because  ---->2+   MACINNER                                                      01-MACOU
   PCONTROL(MCALL) specified          3+*,MNOTE FROM MEMBER COPYCODE                                           02-MACIN
                                      4          END

                                          Macro and Copy Code Source Summary                                Page    3

 Con Source                                        Volume    Members                        HLASM R2.0  1995/04/01 18.01

  L1 MYMAC    MACLIB   A1                          MY-Z-D    COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                               Page    4

Macro     Con  Called By       Defn  References                                             HLASM R2.0  1995/04/01 18.01

COPYCODE   L1  MACINNER           -  1C
MACINNER   L1  MACOUTER           -  2
MACOUTER   L1  PRIMARY INPUT      -  1
```

Figure 16. Assembly with NOLIBMAC and PCONTROL(MCALL) options

Figure 17 shows the output when you specify the LIBMAC and NOPCONTROL options.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R2.0  1995/04/01 18.02

                                      1    MACRO
                                      2    MACOUTER
                                      3    MACINNER
                                      4    MEND
                                      5          MACOUTER
                                      6    MACRO
                                      7    MACINNER
                                      8    COPY COPYCODE
                                      9    MNOTE *,'MNOTE FROM MEMBER COPYCODE'
                                     10    MEND
                                     11+*,MNOTE FROM MEMBER COPYCODE                                             02-00008
                                     12          END

                                          Macro and Copy Code Source Summary                                    Page    3

 Con Source                                        Volume    Members                        HLASM R2.0  1995/04/01 18.02

  L1 MYMAC    MACLIB   A1                          MY-Z-D    COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                   Page    4

Macro     Con  Called By        Defn  References                                            HLASM R2.0  1995/04/01 18.02

COPYCODE   L1  MACINNER           -  8C
MACINNER   L1  MACOUTER          7X 10
MACOUTER   L1  PRIMARY INPUT     2X 5
```

Figure 17. Assembly with LIBMAC and NOPCONTROL options

Figure 18 shows the output when you specify the LIBMAC and PCONTROL(MCALL) options.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R2.0  1995/04/01 18.03

                                      1    MACRO
                                      2    MACOUTER
                                      3    MACINNER
                                      4    MEND
                                      5          MACOUTER
                                      6    MACRO
                                      7    MACINNER
                                      8    COPY COPYCODE
                                      9    MNOTE *,'MNOTE FROM MEMBER COPYCODE'
  This line produced because         10    MEND
  PCONTROL(MCALL) specified   ----> 11+    MACINNER                                                      01-00003
                                     12+*,MNOTE FROM MEMBER COPYCODE                                             02-00008
                                     13          END

                                          Macro and Copy Code Source Summary                                    Page    3

 Con Source                                        Volume    Members                        HLASM R2.0  1995/04/01 18.03

  L1 MYMAC    MACLIB   A1                          MY-Z-D    COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                   Page    4

Macro     Con  Called By        Defn  References                                            HLASM R2.0  1995/04/01 18.03

COPYCODE   L1  MACINNER           -  8C
MACINNER   L1  MACOUTER          7X 11
MACOUTER   L1  PRIMARY INPUT     2X 5
```

Figure 18. Assembly with LIBMAC and PCONTROL(MCALL) options

DSECT Cross Reference

This section of the listing shows the names of all internal or external dummy sections defined in the program, and the number of the statement where the definition of the dummy section was begun.

```
                                                     DSECT Cross Reference                                       Page   15
 1          2          3       4
Dsect      Length      Id      Defn                                                        HLASM R2.0  1995/03/24 08.00
Commarea 0000002D  00000003    760
IHADCB   00000060  FFFFFFFD    190
Input_Workarea
         0000002C  FFFFFFFE    182
Plist    0000007C  FFFFFFFF    179
```

Figure 19. DSECT Cross Reference

1 Shows the name of each dummy section defined in your program.

2 Shows, in hexadecimal notation, the assembled byte length of the dummy section.

3 For external dummy sections, this field indicates the external symbol dictionary ID assigned to the ESD entry for the external dummy section. For internal dummy sections, this field shows the control section ID assigned to the dummy control section. You can use this field in conjunction with the ID field in the symbol and literal cross-reference to relate symbols to a specific section.

4 Shows the number of the statement where the definition of the dummy section began.

You can suppress this section of the listing by specifying the NODXREF assembler option.

USING Map

This section of the listing shows a summary of the USING, DROP, PUSH USING, and POP USING instructions used in your program.

```
                                                           USING MAP                                             Page   16
                                                                                           HLASM R2.0  1995/03/24 08.00
    1     2      3              4      5           6       7        8   9       10    11
  STMT ---LOCATION------- ACTION ----------USING-------------- REG MAX     LAST  LABEL AND USING TEXT
        COUNT        ID          TYPE          VALUE     ID        DISP   STMT
    26 00000002  00000001 USING  ORDINARY    00000000 00000001  12 964     69  Sample,R12
    28 00000002  00000001 USING  DEPENDENT  +00000894 FFFFFFFD  12             IDSECT,ISTORE
    29 00000002  00000001 USING  LAB+DEPND  +000008F4 FFFFFFFD  12             OSTR.IDSECT,OSTORE
    30 00000002  00000001 USING  LABELED     00000000 FFFFFFFF   2 078     60  PlistIn.Plist,R2
    31 00000002  00000001 USING  LABELED     00000000 FFFFFFFF   3 078     60  PlistOut.Plist,R3
    61 00000840  00000001 PUSH
    63 00000840  00000001 DROP                                   2             PlistIn
    64 00000840  00000001 DROP                                   3             PlistOut
```

Figure 20. High Level Assembler USING Map

1 Shows the number of the statement that contains the USING, DROP, PUSH USING, or POP USING instruction.

2 Shows the value of the Location Counter when the USING, DROP, PUSH or POP statement was encountered.

3 Shows the value of the ESDID of the current section when the USING, DROP, PUSH or POP statement was encountered.

4 Shows whether the instruction was a USING, DROP, PUSH, or POP instruction.

5 For USING instructions, this field indicates whether the USING is an ordinary USING, a labeled USING, a dependent USING, or a labeled dependent USING.

6 For ordinary and labeled USING instructions, this field indicates the base address specified in the USING. For dependent USING instructions, this field is prefixed with a plus sign (+) and indicates the hexadecimal offset of the address of the second operand from the base address specified in the corresponding ordinary USING.

7 For USING instructions, this field indicates the ESDID of the section specified on the USING statement.

8 For ordinary and labeled USING instructions, and for DROP instructions, this field indicates the register or registers specified in the instruction. There is a separate line in the USING map for each register specified in the instruction. If the DROP instruction has no operands, all registers and labels are dropped and this field contains **.

For dependent USING instructions, the field indicates the register for the corresponding ordinary USING instruction that is used to resolve the address. If the corresponding ordinary USING instruction has multiple registers specified, only the first register used to resolve the address is displayed.

9 For each base register specified in an ordinary USING instruction or a labeled USING instruction, this field shows the maximum displacement calculated by the assembler when resolving symbolic addresses into base-displacement form using that base register.

10 For ordinary and labeled USING instructions, this field indicates the statement number of the last statement that used the specified base register to resolve an address. Where an ordinary USING instruction is used to resolve a dependent USING, the statement number printed reflects the use of the register to resolve the dependent USING.

11 For USING and DROP instructions, this field lists the text specified on the USING or DROP instruction, truncated if necessary. For labeled USING instructions, the text is preceded by the label specified for the USING.

If a DROP instruction drops more than one register or labeled USING, the text for each register or labeled USING is printed on the line corresponding to the register that is dropped.

You can suppress this section of the listing by specifying the USING(NOMAP) assembler option, or the NOUSING assembler option.

Diagnostic Cross Reference and Assembler Summary

This section of the listing summarizes the error diagnostic messages issued during the assembly, and provides statistics about the assembly. A sample listing exit is provide with High Level Assembler to allow the diagnostic cross reference and assembler summary to be suppressed. See Appendix J, “Sample LISTING User Exit” on page 322. The diagnostic messages issued by the assembler are fully documented in Appendix G, “High Level Assembler Messages” on page 271.

```
                                  Diagnostic Cross Reference and Assembler Summary                          Page   17
                                                                                               HLASM R2.0  1995/03/24 08.00
Statements Flagged
         1
   8(P1,8), 10(L1:MAC1,2), 23(P1,10), 24(P1,11), 123(L2:SAVE,3), 254(L2:PART,34)

   2     6 Statements Flagged in this Assembly        8 was Highest Severity Code

HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 2.0  3

SYSTEM: MVS/ESA SP 5.1           JOBNAME: (NOJOB)     STEPNAME: (NOSTEP)   PROCSTEP: (NOPROC)   4

Datasets Allocated for this Assembly    5
 Con DDname   Dataset Name                                Volume              Member
  P1 SYSIN    MVS.PDS.INPUT                               CJM001              INITMEM1
  L1 SYSLIB   USRMAC   MACLIB   A1                        A49191
  L2          XSMACRO  MACLIB   S2                        MNT190
  L3          OSMACRO  MACLIB   S2                        MNT190
6   SYSADATA T$E$S$T  SYSADATA A1                        A49191
     SYSLIN   T$E$S$T  TEXT     A1                        A49191
     SYSPRINT T$E$S$T  LISTING  A1                        A49191
     SYSPUNCH T$E$S$T  SYSPUNCH A1                        A49191
     SYSTERM  T$E$S$T  SYSTERM  A1                        A49191
7
Input/Output Exit Statistics
Exit Type  Name      Calls  ---Records---  Diagnostic
                            Added Deleted  Messages
ADATA      ASMAXADT   2111      0       0         0
LISTING    ASMAXPRT   2093      0       0         0

8  4969K allocated to Buffer Pool,      9   659K would be required for this to be an In-Storage Assembly
10   17 Primary Input Records Read      12  7258 Library Records Read      14  0 Work File Reads
11 2093 Primary Print Records Written   13     0 Object Records Written    15  0 Work File Writes
16 2111 Adata Records Written
Assembly Start Time: 08:00:02 Stop Time: 08:00:10 Processor Time: 00.00.00.0011   17
Return Code 008
```

Figure 21. Diagnostic Cross Reference and Assembler Summary

The sample listing contains a combination of MVS and CMS data sets to show examples of the differences in data set information.

1 The statement number of a statement that causes an error message, or contains an MNOTE instruction, appears in this list. Flagged statements are shown in either of two formats. When assembler option FLAG(NORECORD) is specified only the statement number is shown. When assembler option FLAG(RECORD) is specified the format is: *statement(dsnum:member,record)*, where

statement is the sequential, absolute statement number as shown in the source and object section of the listing.

dsnum is the value applied to the source or library dataset showing the type of input file, and the concatenation number. `P` indicates the statement was read from the primary input source, and `L` indicates the statement was read from a library. This value is cross referenced to the input datasets listed in the *Datasets Allocated for this Assembly* - see **5**.

member is the name of the macro from which the statement was read. In MVS, this may also be the name of a partitioned data set member that is included in the primary input (SYSIN) concatenation.

record is the relative record number from the start of the dataset or member which contains the flagged statement.

2 The number of statements flagged, and the highest non-zero severity code of all messages issued. The highest severity code is equal to the assembler return code.

If no statements are flagged, the following statement is printed:

```
No Statements Flagged in this Assembly
```

If the assembly completes with a non-zero return code, and there are no flagged statements, it indicates there is a diagnostic message in the *Options Summary* section of the listing.

See Chapter 6, “Diagnosing Assembly Errors” on page 123 for a complete discussion of how error messages and MNOTEs are handled.

3 The current release of High Level Assembler and the last PTF applied.

4 Provides information about the system on which the assembly was run. These are:

- The name and level of the operating system upon which the assembly was run.
- The jobname for the assembly job. If the jobname is not available, the value of (NOJOB) is printed.
- The stepname for the assembly job. If the stepname is not available, the value of (NOSTEP) is printed.
- The procedure name for the assembly job. If the procedure name is not available, the value of (NOPROC) is printed.

5 All data sets used in the assembly are listed by their standard ddname. The data set information includes the data set name, and the serial number of the volume containing the data set. In MVS, the data set information may also include the name of a member of a partitioned dat set (PDS).

If a user exit provides the data set information then the data set name is the value extracted from the Exit-Specific Information block described in the *Programmer's Guide*.

The `Con` column shows the concatenation value assigned for each input data set. You use this value to cross reference flagged statements, and macros and copy code members listed in the Macro and Copy Code Cross Reference section.

MVS: Under MVS, the data set name for all data sets is extracted from the MVS job file control block (JFCB). If the data set is a JES2 spool file, for example, the data set name is the name allocated by JES2. If the data set is allocated to DUMMY, or NULLFILE, the data set name is shown as NULLFILE.

CMS: Under CMS, the data set name is assigned one of the values shown in Figure 22.

Figure 22 (Page 1 of 2). Data set names under CMS

File Allocated To:	Data Set Name:
CMS file	The 8-character filename, the 8-character filetype, and the 2-character filemode of the file, each separated by a blank. If the data set is a disk file in the Shared File system, the volume serial number contains `'** SFS'`.
Dummy file (no physical I/O)	`DUMMY`
Printer	`PRINTER`
Punch	`PUNCH`
Reader	`READER`

Figure 22 (Page 2 of 2). Data set names under CMS

File Allocated To:	**Data Set Name:**
Labeled tape file	The data set name of the tape file
Unlabeled tape file	`TAPn`, where *n* is a value from 0 to 9, or A to F.
Terminal	`TERMINAL`

6 Output datasets do not have a concatenation value.

7 The usage statistics of the I/O exits you specified for the assembly. If you do not specify an exit the assembler does not produce any statistics. The following statistics are reported:

EXIT TYPE The type of exit.
NAME The name of the exit module as specified in the EXIT assembler option.
CALLS The number of times the exit was called.
RECORDS The number of records ADDED and DELETED by the exit.
DIAGNOSTIC MESSAGES
The number of diagnostic messages printed, as a result of exit processing.

All counts are shown right justified and leading zeroes are suppressed, unless the count is zero.

8 The amount of storage allocated to the buffer pool.

9 The minimum value for the SIZE option that the assembler estimates would result in the assembly being done in storage. This may be less than the amount of storage allocated to the assembler. If the amount of storage in the buffer pool is not enough for an in-storage assembly, this value contains the assembler's approximation of the amount of storage required.

10 The number of primary input records read by the assembler. This count does not include any records read or discarded by the SOURCE user exit.

11 This is a count of the actual number of records generated by the assembler; it may be less than the total number of printed and blank lines appearing in the listing if the SPACE *n* assembler instruction is used. For a SPACE *n* that does not cause an eject, the assembler inserts *n* blank lines in the listing by generating *n*/3 blank records, rounded to the next lower integer if a fraction results. For example, for a SPACE 2, no blank records are generated. The assembler does not generate a blank record to force a page eject.

This count does not include any listing records generated or discarded by the LISTING user exit.

12 The number of records read from the libraries allocated to SYSLIB. This count does not include any records read or discarded by the LIBRARY user exit.

13 The number of object records written. This count does not include any object records generated or discarded by the OBJECT or PUNCH user exits.

14 The number of reads from the work file (SYSUT1).

15 The number of writes to the work file (SYSUT1).

16 The number of ADATA records written to the associated data file.

17 The assembly start and stop times in hours, minutes and seconds and the approximate amount of processor time used for the assembly, in hours, minutes, and seconds to four decimal places.

The assembly start times does not include the time used during assembly initialization which allocates main storage and data sets, and processes the assembler invocation parameters. The assembly stop times does not include the time used during assembly termination which deallocates main storage and data sets.

Chapter 3. Controlling your Assembly with Options

High Level Assembler offers a number of optional facilities. For example, you can suppress printing of the assembly listing or parts of the listing, and you can specify whether you want an object module or an associated data file. There are two types of options:

- Simple pairs of keywords: A positive form (such as OBJECT) that requests a facility, and an alternate negative form (such as NOOBJECT) that excludes that facility
- Keywords, such as LINECOUNT(50), that permit you to assign a value to a function

This chapter describes each of the assembler options, and when you can use them. Each of the options has a default value that the assembler uses if you do not specify an alternative value. The default values are explained under “Default Options” on page 36.

Specifying Assembler Options

The way you specify the options depends on the environment in which High Level Assembler is running.

MVS Batch: Under MVS Batch, you select the options by specifying them in the PARM field of the JCL EXEC statement that invokes the assembler. For example:

```
//ASSEMBLE EXEC PGM=ASMA90,PARM='LIST(133),DBCS'
```

You can also use catalogued procedures to invoke the assembler. To override options in a cataloged procedure, you must include the PARM field in the EXEC statement that invokes the procedure. If the cataloged procedure contains more than one step, you must also qualify the keyword parameter (PARM) with the name of the step within the procedure that invokes the assembler. For example:

```
//   EXEC    ASMACG,PARM.C='LIST(133),DBCS'
```

“Overriding Statements in Cataloged Procedures” on page 162 contains more examples on how to specify options in a cataloged procedure.

TSO: Under TSO, you select the options by specifying them in the second parameter of the TSO CALL command that invokes the assembler. For example:

```
CALL 'SYS1.LINKLIB(ASMA90)' 'LIST(133),DBCS'
```

CMS: Under CMS, you select the options by specifying them after the left parenthesis on the CMS ASMAHL command that invokes the assembler. For example:

```
ASMAHL filename (LIST(133) DBCS[)]
```

Coding Rules: The rules for coding the assembler options are:

- You can specify the options in any order.
- If you specify contradictory options, for example, LIST and NOLIST, the assembler uses the last (or rightmost) option, and issues a warning message.

- If you specify an incorrect option the assembler issues a diagnostic message, and sets the return code to 2 or higher. You can prevent the setting of the return code by using the FLAG option.
- Under CMS, if you specify two or more options, the options can be separated by spaces or commas.
- Under MVS, if you specify two or more options, the list of options must be enclosed within single quotation marks or parentheses. Each option must be separated by a comma.

  If you specify only one option and it does not include any special characters, the enclosing single quotation marks or parentheses can be omitted.

  All options that have suboptions must be within single quotation marks because they contain special characters.

  The option list must not be longer than 100 characters, including the separating commas.

  If you need to continue the PARM field onto another record, the entire PARM field must be enclosed in parentheses. However, any part of the PARM field enclosed in single quotation marks must not be continued on another record.

Fixed Options: If an option was specified on the DELETE operand of the ASMAOPT macro during installation, you cannot change the option when you invoke the assembler.

PESTOP: If the PESTOP option was specified during installation, and an error is detected in the options you specify at run time, the assembly stops.

****Process Statements:*** Process (*PROCESS) statements let you specify selected assembler options in your assembler source program. You can include them in the primary input data set or provide them from a SOURCE user exit. You cannot specify the following options on process statements:

```
ADATA       LINECOUNT   SYSPARM
ASA         LIST        TERM
DECK        OBJECT      TRANSLATE
EXIT        OPTABLE     XOBJECT
LANGUAGE    SIZE
```

Refer to the *Language Reference* for a description of the *PROCESS statement.

Invoking the Assembler Dynamically: Assembler options can be passed in a parameter list when the assembler is invoked dynamically from an executing program. Refer to “Invoking the Assembler Dynamically” on page 134 for further information.

Default Options

When High Level Assembler is installed, each assembler option is preset to a default. The IBM-supplied default options are shown above the main path of the syntax diagrams in the description of the assembler options that follow. However, these might *not* be the default options in effect at your installation; the defaults could have been changed when High Level Assembler was installed. For example, NOADATA is an IBM-supplied default, and ADATA might be the default at your installation. Default options can be fixed during installation which prevents you from overriding them during the assembly. The assembler issues a message if you try to override a fixed option.

Precedence of Assembler Options

Assembler options are recognized in the order of precedence (highest to lowest) described below.

- Fixed installation defaults (options that may not be specified at assembly time because they were specified in the DELETE operand of the OPTIONS installation macro)
- Options on the JCL PARM parameter of the EXEC statement under MVS or the ASMAHL command under CMS
- Options on *PROCESS statements
- Non-fixed installation defaults

Assembler Options

A description of each of the options you can use to control the assembly follow.

Refer to “Syntax Notation” on page xiv for instructions how to read the option syntax diagrams. The IBM-supplied option defaults are shown above the main path of the syntax diagrams.

ADATA

```
        ┌─NOADATA─┐
►►──────┴─ADATA───┴──────►◄
```

Default
NOADATA

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

ADATA
Specifies that the assembler collect associated data and write it to the associated data file. You define the associated data file with the SYSADATA ddname. Appendix D, “Associated Data File Output” on page 217 describes the format of the associated data file.

NOADATA
Specifies that the assembler is not to collect associated data. If you specify NOADATA, then the assembler ignores the EXIT(ADEXIT)) option.

ALIGN

```
    ┌─ALIGN───┐
►►──┴─NOALIGN─┴──►◄
```

Default
ALIGN

Abbreviations
None

ALIGN
Instructs the assembler to check alignment of addresses in machine instructions for consistency with the requirements of the operation code type. DC, DS, DXD, and CXD are to be aligned on the correct boundaries.

NOALIGN
Instructs the assembler not to check alignment of unprivileged machine instruction data references, but still to check instruction references and privileged machine instruction data references. DC, DS, and DXD are to be aligned on the correct boundaries only if the duplication factor is 0.

Note: Specify the FLAG(NOALIGN) option to suppress the message issued when the assembler detects an alignment inconsistency.

ASA

```
    ┌─NOASA─┐
►►──┴─ASA───┴──►◄
```

Default
NOASA

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

ASA
Instructs the assembler to use American National Standard printer control characters in records written to the listing data set.

NOASA
Instructs the assembler to use machine printer control characters in records written to the listing data set.

BATCH

```
    ┌─BATCH───┐
►►──┴─NOBATCH─┴──►◄
```

Default
BATCH

Abbreviations
None

BATCH
Instructs the assembler that multiple assembler source programs may be in the input data set. The first statement of the second and subsequent source programs must immediately follow the END statement of the previous source program. An end-of-file must immediately follow the last source program.

NOBATCH
Instructs the assembler that only one assembler source program is in the input data set.

COMPAT

```
    ┌─NOCOMPAT─────────────────────┐
    │           ┌─,──────────┐     │
►►──┴─COMPAT(─▼─┬─CASE──────┬┴─)─┴──►◄
                ├─MACROCASE─┤
                └─SYSLIST───┘
```

Default
NOCOMPAT

Abbreviations
CPAT(CASE,MC,SYSL) / NOCPAT

COMPAT(CASE)
Instructs the assembler to maintain uppercase alphabetic character set compatibility with earlier assemblers. It restricts language elements to uppercase alphabetic characters A through Z if they were so restricted in earlier assemblers.

COMPAT(MACROCASE)
Instructs the assembler to convert lowercase alphabetic characters (a through z) in unquoted macro operands to uppercase alphabetic characters (A through Z).

COMPAT(SYSLIST)
Instructs the assembler to treat sublists in SETC symbols as compatible with earlier assemblers. SETC symbols that are assigned parenthesized sublists are treated as character strings, not sublists, when passed to a macro definition in an operand of a macro instruction.

NOCOMPAT
Instructs the assembler to allow lowercase alphabetic characters a through z in all language elements, and to treat sublists in SETC symbols as sublists when passed to a macro definition in the operand of a macro instruction.

DBCS

```
        ┌─NODBCS─┐
▶▶──────┴─DBCS───┴──▶◀
```

Default
NODBCS

Abbreviations
None

DBCS
Instructs the assembler to accept double-byte character set data, and to support graphic (G-type) constants and self-defining terms. The assembler recognizes X'0E' and X'0F' in character strings enclosed by single quotation marks, and treats them as Shift-Out and Shift-In control characters for delimiting DBCS data.

NODBCS
Specifies that the assembler does not recognize X'0E' and X'0F' as double-byte character set data delimiters, and does not support graphic (G-type) constants and self-defining terms.

DECK

```
        ┌─NODECK─┐
▶▶──────┴─DECK───┴──▶◀
```

Default
NODECK

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

DECK
Specifies that the assembler generate object code and write it it to the object data set. You define the object data set with the SYSPUNCH ddname.

NODECK
Instructs the assembler not to write the object code to SYSPUNCH.

If you specify NODECK, NOOBJECT, and NOXOBJECT, the assembler ignores the EXIT(OBJEXIT)) option.

DISK

See “PRINT” on page 55.

DXREF

```
        ┌─DXREF───┐
►►──────┴─NODXREF─┴──────►◄
```

Default
DXREF

Abbreviations
DX / NODX

DXREF
Instructs the assembler to produce the *DSECT Cross Reference* section of the assembler listing. The DSECT cross reference includes:

- The symbolic names of all DSECTs defined in the assembly
- The assembled length of each DSECT
- The ESDID of each DSECT
- The statement number which defines the DSECT

NODXREF
Instructs the assembler not to produce the *DSECT Cross Reference* section of the assembler listing.

ERASE

```
        ┌─ERASE───┐
►►──────┴─NOERASE─┴──────►◄
```

Default
ERASE

Abbreviations
None

Restrictions
This option is not allowed on *PROCESS statements.

This option can only be specified when you use the ASMAHL command under CMS.

ERASE
Specifies that the existing files with a filename the same as the filename on the ASMAHL command, and a filetype of LISTING, TEXT, and SYSADATA, are to be deleted before the assembly is run. Only files on the disk on which the assembler writes the new listing, object, and associated data files are deleted.

NOERASE
Specifies that the existing LISTING, TEXT and SYSADATA files are not to be deleted before the assembly is run.

ESD

```
          ┌─ESD───┐
 ►►───────┴─NOESD─┴────►◄
```

Default
ESD

Abbreviations
None

ESD
Instructs the assembler to produce the *External Symbol Dictionary* section of the assembler listing. The ESD contains the external symbol dictionary information that is passed to the linkage editor or loader, or DFSMS/MVS binder, in the object module.

NOESD
Instructs the assembler not to produce the *External Symbol Dictionary* section of the assembler listing.

EXIT

```
          ┌─NOEXIT────────────────────────────────────────────┐
          │          ┌─,──────────────────────────────────┐   │
 ►►───────┴─EXIT(─▼──┬─INEXIT(mod1─┬────────────┬─)───┬─┴─)─┴───►◄
                     │             └─(─str1─)───┘     │
                     ├─NOINEXIT───────────────────────┤
                     ├─LIBEXIT(mod2─┬────────────┬─)──┤
                     │              └─(─str2─)───┘    │
                     ├─NOLIBEXIT──────────────────────┤
                     ├─PRTEXIT(mod3─┬────────────┬─)──┤
                     │              └─(─str3─)───┘    │
                     ├─NOPRTEXIT──────────────────────┤
                     ├─OBJEXIT(mod4─┬────────────┬─)──┤
                     │              └─(─str4─)───┘    │
                     ├─NOOBJEXIT──────────────────────┤
                     ├─ADEXIT(mod5─┬────────────┬─)───┤
                     │             └─(─str5─)───┘     │
                     ├─NOADEXIT───────────────────────┤
                     ├─TRMEXIT(mod6─┬────────────┬─)──┤
                     │              └─(─str6─)───┘    │
                     └─NOTRMEXIT──────────────────────┘
```

Default
NOEXIT

Abbreviations
EX(INX,LBX,PRX,OBX,ADX,TRX) / NOEX

Restrictions
You cannot specify this option on *PROCESS statements.

INEXIT
Specifies that the assembler use an input (SOURCE) exit for the assembly. *mod1* is the name of the load module for the exit. The assembler passes control to the load module for SOURCE type exit processing.

You can use a SOURCE exit, for example, to read variable-length source input records. See also Appendix K, "Sample SOURCE User Exit" on page 324.

NOINEXIT
Specifies that there is no SOURCE exit.

LIBEXIT
Specifies that the assembler use a LIBRARY exit for the assembly. *mod2* is the name of the load module for the exit. The assembler passes control to the load module for LIBRARY type exit processing.

You can use this exit, for example, to handle non-standard libraries or, under CMS, macros and copy books that are in separate CMS files instead of CMS MACLIBs.

NOLIBEXIT
Specifies that there is no LIBRARY exit.

PRTEXIT
Specifies that the assembler use a LISTING exit for the assembly. *mod3* is the name of the load module for the exit. The assembler passes control to the load module for LISTING type exit processing.

You can use the LISTING exit, for example, to suppress parts of the assembly listing, or provide additional listing lines. See also Appendix J, "Sample LISTING User Exit" on page 322.

NOPRTEXIT
Specifies that there is no LISTING exit.

OBJEXIT
Specifies that the assembler use an OBJECT exit or PUNCH exit, or both for the assembly. *mod4* is the name of the load module for the exit. The assembler passes control to the load module for OBJECT type exit processing when you specify either the OBJECT or XOBJECT option. The assembler passes control to the load module for PUNCH type exit processing when you specify the DECK option. The OBJEXIT suboption is ignored if you specify the assembler options NODECK, NOOBJECT, and NOXOBJECT.

NOOBJEXIT
Specifies that there is no OBJECT exit or PUNCH exit.

ADEXIT
Specifies that the assembler use an ADATA exit for the assembly. *mod5* is the name of the load module for the exit. The assembler passes control to the load module for ADATA type exit processing. See also Appendix I, "Sample ADATA User Exit" on page 315.

NOADEXIT
Specifies that there is no ADATA exit.

TRMEXIT
Specifies that the assembler use a TERM exit for the assembly. *mod6* is the name of the load module for the exit. The assembler passes control to the load module for TERM type exit processing.

NOTRMEXIT
Specifies that there is no TERM exit.

NOEXIT
Specifies that there are no exits for the assembly.

The module names *mod1*, *mod2*, *mod3*, *mod4*, *mod5*, and *mod6* can refer to the same load module.

The suboptions *str1*, *str2*, *str3*, *str4*, *str5*, and *str6* are optional. They are character strings, up to 64 characters in length, that are passed to the exit module during OPEN processing. You may include any character in a string, but you must pair parentheses. JCL restrictions require that you specify two single quotation marks to represent a single quotation mark, and two ampersands to represent a single ampersand.

For more information about the EXIT option, see Chapter 4, “Providing User Exits” on page 66.

You specify these options in the installation default options using the ADEXIT, INEXIT, LIBEXIT, OBJEXIT, PRTEXIT, and TRMEXIT operands.

FLAG

```
                ┌─,──────────┐
                │┌─0─────────┐│
                │├─ALIGN─────┤│
                │├─CONT──────┤│
                │├─RECORD────┤│
                │├─NOSUBSTR──┤│
►►──FLAG(──────▼┴───────────┴┴─)──►◄
                 ├─integer───┤
                 ├─NOALIGN───┤
                 ├─NOCONT────┤
                 ├─NORECORD──┤
                 └─SUBSTR────┘
```

Default
FLAG(0,ALIGN,CONT,RECORD,NOSUBSTR)

Abbreviations
RC,AL,SUB / NORC,NOAL,NOSUB

integer
Specifies that error diagnostic messages with this or a higher severity code are printed in the *source and object* section of the assembly listing. Error diagnostic messages with a severity code lower than *integer* do not appear in the *source and object* section, and the severity code associated with those messages is not used to set the return code issued by the assembler. Any severity code from 0 through 255 may be specified. Error diagnostic messages have a severity code of 0, 2, 4, 8, 12, 16, or 20. MNOTEs can have a severity code of 0 through 255.

When specified with the TERM assembler option, FLAG controls which messages are displayed in the terminal output.

FLAG(ALIGN)
Instructs the assembler to issue diagnostic message `ASMA033W` when an inconsistency is detected between the operation code type and the alignment of

addresses in machine instructions. Assembler option ALIGN describes when the assembler detects an inconsistency.

FLAG(NOALIGN)
Instructs the assembler not to issue diagnostic message `ASMA033W` when an inconsistency is detected between the operation code type and the alignment of addresses in machine instructions.

FLAG(CONT)
Specifies that the assembler is to issue diagnostic messages `ASMA430W` through `ASMA433W` when one of the following situations occurs in a macro call instruction:

- The operand on the continued record ends with a comma, and a continuation statement is present but continuation does not start in the continue column (usually column 16).
- A list of one or more operands ends with a comma, but the continuation column (usually column 72) is blank.
- The continuation record starts in the continue column (usually column 16) but there is no comma present following the operands on the previous record.
- The continued record is full but the continuation record does not start in the continue column (usually column 16).

FLAG(NOCONT)
Specifies that the assembler is not to issue diagnostic messages `ASMA430W` through `ASMA433W` when an inconsistent continuation is encountered.

FLAG(RECORD)
Instructs the assembler to do the following:

- Issue diagnostic message `ASMA435I` immediately after the last diagnostic message for each statement in error. The message text describes the record number and input data set name of the statement in error.
- Include the member name (if applicable), the record number and the input data set concatenation value with the statement number in the list of flagged statements in the *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing.

FLAG(NORECORD)
Instructs the assembler to do the following:

- Not issue diagnostic message `ASMA435I` for statements in error.
- Only show the statement number in the list of flagged statements in the *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing.

FLAG(SUBSTR)
Instructs the assembler to issue warning diagnostic message `ASMA094` when the second subscript value of the substring notation indexes past the end of the character expression.

FLAG(NOSUBSTR)
Instructs the assembler not to issue warning diagnostic message `ASMA094` when the second subscript value of the substring notation indexes past the end of the character expression.

The FLAG suboptions ALIGN, CONT, RECORD, and SUBSTR are specified in the installation default options as ALIGNWARN, CONTWARN, RECORDINFO, and SUBSTRWARN, respectively.

FOLD

```
        ┌─NOFOLD─┐
►►──────┴─FOLD───┴──►◄
```

Default
NOFOLD

Abbreviations
None

FOLD
Instructs the assembler to convert lowercase alphabetic characters (a through z) in the assembly listing to uppercase alphabetic characters (A through Z). All lowercase alphabetic characters are converted, including lowercase characters in source statements, assembler error diagnostic messages, and assembly listing lines provided by a user exit. Lowercase alphabetic characters are converted to uppercase alphabetic characters, regardless of the setting of the COMPAT(CASE) option.

NOFOLD
Specifies that lowercase alphabetic characters are not converted to uppercase alphabetic characters.

The assembler listing headings are not affected by the FOLD option. The LANGUAGE option controls the case for assembler listing headings.

LANGUAGE

```
                   ┌─EN─┐
►►──LANGUAGE(──┼─DE─┼──)──►◄
                   ├─ES─┤
                   ├─JP─┤
                   └─UE─┘
```

Default
LANGUAGE(EN)

Abbreviations
LANG(EN|ES|DE|JP|UE)

Restrictions
This option is not allowed on *PROCESS statements.

LANGUAGE(EN)
Specifies that the assembler issues messages, and prints the assembler listing headings in mixed uppercase and lowercase English.

LANGUAGE(DE)
Specifies that the assembler issues messages in German. The assembler listing headings are printed in mixed case English.

LANGUAGE(ES)
Specifies that the assembler issues messages in Spanish. The assembler listing headings are printed in mixed case English.

LANGUAGE(JP)
Specifies that the assembler issues messages in Japanese. The assembler listing headings are printed in uppercase English.

LANGUAGE(UE)
Specifies that the assembler issues messages, and prints the assembler listing headings in uppercase English.

Note: The assembler uses the language specified in the installation default options for messages produced in CMS by the ASMAHL command.

LIBMAC

```
     ┌─NOLIBMAC─┐
►►───┴─LIBMAC───┴───►◄
```

Default
NOLIBMAC

Abbreviations
LMAC / NOLMAC

LIBMAC
Specifies that, for each macro, macro definition statements read from a macro library are to be imbedded in the input source program immediately preceding the first invocation of that macro. The assembler assigns statement numbers to the macro definition statements as though they were included in the input source program.

NOLIBMAC
Specifies that macro definition statements read from a macro library are not included in the input source program.

LINECOUNT

```
           ┌─60──────┐
►►──LINECOUNT(──┴─integer─┴──)──►◄
```

Default
LINECOUNT(60)

Abbreviations
LC(*integer*)

CMS Only:

The LINECOUNT option can be abbreviated to LINECOUN.

Restrictions
This option is not allowed on *PROCESS statements.

integer
Specifies the number of lines to be printed on each page of the assembly listing. *integer* must have a value of 0, or 10 to 32767. If a value of 0 is specified, no page ejects are generated and EJECT, CEJECT, and TITLE statements in the assembly are ignored.

Up to 7 lines on each page may be used for heading lines.

LIST

```
►►──┬─LIST──┬─────────────────┬─┬──►◄
    │       │    ┌─121─┐      │ │
    │       └─(──┼─133─┼──)───┘ │
    │            └─MAX─┘        │
    └─NOLIST────────────────────┘
```

Default
LIST(121)

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

LIST
Instructs the assembler to produce a listing. Specifying LIST without a sub-option is equivalent to specifying LIST(121).

LIST(133)
Instructs the assembler to produce a listing, and print the *Source and Object* section in the 133-character format. You should use this option when you specify the XOBJECT option.

LIST(121)
Instructs the assembler to produce a listing, and print the *Source and Object* section in the 121-character format.

LIST(MAX)
Instructs the assembler to produce a listing, and print the *Source and Object* section in either the 121-character format or the 133-character format. If the logical record length (LRECL) of the listing data set is less than 133 then the assembler selects the 121-character format. If the LRECL of the listing data set is 133 or more then the assembler selects the 133-character format.

NOLIST
Instructs the assembler to suppress the assembly listing. When you specify NOLIST the assembler ignores the following options:

```
DXREF            MXREF
ESD              PCONTROL
EXIT(PRTEXIT)    RLD
MAP              XREF
```

MXREF

```
►►──┬─MXREF──┬──────────────────────────┬──┬──►◄
    │        │      ┌─SOURCE─┐          │  │
    │        └─(──┼─FULL───┼──)─────────┘  │
    │               └─XREF───┘             │
    └─NOMXREF──────────────────────────────┘
```

Default
MXREF(SOURCE)

Abbreviations
MX / NOMX

MXREF
Specifying MXREF without a suboption is equivalent to specifying MXREF(SOURCE).

MXREF(SOURCE)
Instructs the assembler to produce the *Macro and Copy Code Source Summary* section of the assembler listing. The macro and copy code source summary includes the name of each macro library or copy library accessed, the volume serial number of the first DASD volume on which the library resides, and the names of each member retrieved from the library.

MXREF(FULL)
Instructs the assembler to produce the *Macro and Copy Code Source Summary* section and the *Macro and Copy Code Cross Reference* section of the assembler listing.

MXREF(XREF)
Instructs the assembler to produce the *Macro and Copy Code Cross Reference* section of the assembler listing. The *Macro and Copy Code Cross Reference* includes the name of each macro or copy code member referenced in the assembly, where it was referenced and where it was called or copied from.

SIZE Option: You might need to specify a large value in the SIZE option to obtain the *Macro and Copy Code Cross Reference* section.

NOMXREF
Specifies that macro and copy code information is not generated in the assembler listing.

NOPRINT

See “PRINT” on page 55.

NOSEG

```
►►──┬─NOSEG─┬──►◄
    └─SEG───┘
```

Default
None. The assembler modules are loaded from the Logical Saved Segment (LSEG). If the LSEG is not available, the assembler modules are loaded from disk.

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

You can only specify this option under CMS using the ASMAHL command.

NOSEG
Specifies that the assembler modules are loaded from disk.

SEG
Specifies that the assembler modules are loaded from the Logical Saved Segment (LSEG). If the LSEG is not found the assembler stops.

OBJECT

```
    ┌─OBJECT───┐
►►──┴─NOOBJECT─┴──►◄
```

Default
OBJECT

Abbreviations
OBJ / NOOBJ

Restrictions
You cannot specify this option on *PROCESS statements.

OBJECT
Instructs the assembler to generate object code and write it it to the object data set. You define the object data set with the SYSLIN ddname.

NOOBJECT
Instructs the assembler not to write the object code to SYSLIN.

OPTABLE

```
          ┌─UNI─┐
►►──OPTABLE(─┼─DOS─┼─)──►◄
          ├─ESA─┤
          ├─XA──┤
          └─370─┘
```

Default
OPTABLE(UNI)

Abbreviations
OP(DOS|ESA|UNI|XA|370)

Restrictions
This option is not allowed on *PROCESS statements.

OPTABLE(DOS)
Instructs the assembler to load and use the DOS operation code table. The DOS operation code is designed specifically for assembling programs previously assembled using the DOS/VSE assembler. The following instructions are not included in the DOS operation code table:

- Vector facility machine instructions
- Machine instructions:

BAS	IVSK	SIGP
BASR	LASP	SPX
CLRCH	MVCK	SSAR
CONCS	MVCP	SSKE
DISCS	MVCS	STAP
EPAR	PC	STPX
ESAR	PT	TB
IAC	RIO	TPROT
IPTE	RRBE	
ISKE	SAC	

- Assembler instructions:

ADATA	CCW0	MHELP
AEJECT	CCW1	OPSYN
ALIAS	CEJECT	POP
AMODE	CXD	PUSH
AREAD	DXD	RMODE
ASPACE	EXITCTL	RSECT
CATTR	LOCTR	

OPTABLE(ESA)
Instructs the assembler to load and use the operation code table that contains the ESA/370 and ESA/390 architecture machine instructions, including those with a vector facility.

OPTABLE(UNI)
Instructs the assembler to load and use the operation code table that contains the System/370 and System/390* architecture machine instructions, including those with a vector facility.

OPTABLE(XA)
Instructs the assembler to load and use the operation code table that contains the System/370 extended architecture machine instructions, including those with a vector facility.

OPTABLE(370)
Instructs the assembler to load and use the operation code table that contains the System/370 systems machine instructions, including those with a vector facility.

Notes:

1. These operation code tables do not contain symbolic operation codes for machine instructions that are unique to IBM 4300 Processors operating in ECPS:VSE mode.
2. The operation codes supported by High Level Assembler are described in the manuals listed under “Related Publications (Architecture)” on page 331.

PCONTROL

```
        ┌─NOPCONTROL─────────────────────────┐
        │                ┌─,─────────┐        │
►►──────┴─PCONTROL(─▼─┬─DATA──────┬─┴─)────┴──►◄
                       ├─NODATA────┤
                       ├─GEN───────┤
                       ├─NOGEN─────┤
                       ├─MCALL─────┤
                       ├─NOMCALL───┤
                       ├─MSOURCE───┤
                       ├─NOMSOURCE─┤
                       ├─ON────────┤
                       ├─OFF───────┤
                       ├─UHEAD─────┤
                       └─NOUHEAD───┘
```

Default
NOPCONTROL

Abbreviations
PC(DATA,NODATA,GEN,NOGEN,MC,NOMC,MS,NOMS,ON,OFF,UHD,NOUHD) / NOPC

PCONTROL(DATA)
Specifies that the assembler is to print the object code of all constants in full, as though a PRINT DATA statement were specified at the beginning of the source program. All PRINT NODATA statements in the source program are ignored. However, specifying PCONTROL(DATA) does not override PRINT OFF or PRINT NOGEN statements in the source program.

PCONTROL(NODATA)
Specifies that the assembler is to print only the first 8 bytes of the object code of constants, as though a PRINT NODATA statement were specified at the beginning of the source program. All PRINT DATA statements in the source program are ignored.

PCONTROL(GEN)
Specifies that the assembler is to print all statements generated by the processing of a macro, as though a PRINT GEN statement were specified at the beginning of the source program. All PRINT NOGEN statements in the source program are ignored. However, specifying PCONTROL(GEN) does not override PRINT OFF statements in the source program.

PCONTROL(NOGEN)
Specifies that the assembler is not to print statements generated by the processing of a macro or open code statements with substitution variables, as though a PRINT NOGEN statement were specified at the beginning of the source program. All PRINT GEN and PRINT MSOURCE statements in the source program are ignored.

PCONTROL(MCALL)
Specifies that the assembler is to print nested macro instructions, as though a PRINT MCALL statement were specified at the beginning of the source program. All PRINT NOMCALL statements in the source program are ignored. However, specifying PCONTROL(MCALL) does not override PRINT OFF or PRINT NOGEN statements in the source program.

PCONTROL(NOMCALL)
Instructs the assembler not to print nested macro instructions, as though a PRINT NOMCALL statement were specified at the beginning of the source program. All PRINT MCALL statements in the source program are ignored.

PCONTROL(MSOURCE)
Specifies that the assembler is to print the source statements generated during macro processing, as well as the assembled addresses and generated object code of the statements. All PRINT NOMSOURCE statements in the source program are ignored. However, specifying PCONTROL(MSOURCE) does not override PRINT OFF or PRINT NOGEN statements in the source program.

PCONTROL(NOMSOURCE)
Instructs the assembler to suppress the printing of source statements generated during macro processing, but not suppress the printing of the assembled addresses and generated object code of the statements. All PRINT MSOURCE statements in the source program are ignored.

PCONTROL(OFF)
Specifies that the assembler is not to produce the *source and object* section of the assembly listing. All PRINT ON statements in the source program are ignored.

PCONTROL(ON)
Specifies that the assembler is to produce an assembly listing unless the NOLIST option is specified. All PRINT OFF statements in the source program are ignored.

PCONTROL(UHEAD)
Specifies that the assembler is to print a summary of active USINGs in the heading lines of each page of the source and object code section of the listing, as though a PRINT UHEAD statement were specified at the beginning of the source program. All PRINT NOUHEAD statements in the source program are ignored. However, specifying PCONTROL(UHEAD) does not override PRINT OFF statements in the source program.

PCONTROL(NOUHEAD)
Instructs the assembler not to print a summary of active USINGs, as though a PRINT NOUHEAD statement were specified at the beginning of the source program. All PRINT UHEAD statements in the source program are ignored.

NOPCONTROL
Specifies that the assembler honor all PRINT statements in the source program. The standard PRINT operands active at the beginning of an assembly are ON, GEN, NODATA, NOMCALL, MSOURCE, and UHEAD.

NOLIST Assembler Option: The PCONTROL option cannot be used to override the NOLIST option. If the NOLIST option is specified, the PCONTROL option is ignored.

PESTOP

PESTOP is an installation-default option that instructs the assembler to terminate when an error is detected in the invocation parameters or *PROCESS statements. Refer to the *Installation and Customization Guide*, SC26-3494, for instructions how to specify this option.

PROFILE

```
      ┌─NOPROFILE──────────────────┐
►►────┴─PROFILE──┬─────────────┬───┴──►◄
                 └─(──name──)─┘
```

Default
NOPROFILE

Abbreviations
PROF

PROFILE
Instructs the assembler to copy the installation-default profile member into the source program, as if the source program contained a COPY instruction.

PROFILE(*name*)
Instructs the assembler to copy the member *name* into the source program, as if the source program contained a COPY instruction.

NOPROFILE
Specifies that the assembler is not to copy a library member into the source program.

Notes:

1. The profile member is copied into the source program immediately following an ICTL statement or *PROCESS statements, or both.
2. You specify the default profile member name in the PROFMEM parameter of the installation options macro ASMAOPT. If the PROFMEM parameter is not specified, ASMAOPT generates a default member name of ASMAPROF. Refer to the *Installation and Customization Guide* for instructions how to use the ASMAOPT macro.

3. The assembler searches for the member in the macro and copy code libraries defined in the SYSLIB DD statement.
4. The assembler processes the source statements in the profile member the same way it does for source statements obtained using the COPY instruction. Refer to the *Language Reference* for further information about the COPY instruction.

PRINT

```
        ┌─DISK────┐
►►──────┼─PRINT───┼──►◄
        └─NOPRINT─┘
```

Default
DISK

Abbreviations
PR/NOPR/DI

Restrictions
This option is not allowed on *PROCESS statements.

This option can only be specified when you use the ASMAHL command under CMS.

PRINT
Specifies that the LISTING file is to be written on the virtual printer.

NOPRINT
Specifies that the writing of the LISTING file is suppressed. Any diagnostic messages to be written to SYSTERM are not affected.

DISK
Specifies that the LISTING file is to be written to disk.

RA2

```
        ┌─NORA2─┐
►►──────┴─RA2───┴──►◄
```

Default
NORA2

Abbreviations
None

RA2
Instructs the assembler to suppress error diagnostic message ASMA066 when 2-byte relocatable address constants, such as AL2(*) and Y(*), are defined in the source program.

NORA2
Instructs the assembler to issue error diagnostic message ASMA066 when 2-byte relocatable address constants, such as AL2(*) and Y(*), are defined in the source program.

RENT

```
       ┌─NORENT─┐
►►─────┴─RENT───┴───►◄
```

Default
NORENT

Abbreviations
None

RENT
Specifies that the assembler checks for possible coding violations of program reenterability. Non-reenterable code is identified by an error message, but is not exhaustively checked because the assembler cannot check the logic of the code. Therefore, the assembler might not detect all violations of program reenterability.

NORENT
Specifies that the assembler not check for possible coding violations of program reenterability.

RLD

```
       ┌─RLD────┐
►►─────┴─NORLD──┴───►◄
```

Default
RLD

Abbreviations
None

RLD
Instructs the assembler to produce the *Relocation Dictionary* (RLD) section of the assembler listing. The RLD shows the relocation dictionary information that is passed to the linkage editor or loader, or DFSMS/MVS binder, in the object module.

NORLD
Instructs the assembler not to produce the RLD section of the assembler listing.

SEG

See “NOSEG” on page 49.

SIZE

```
          ┌─MAX─┐
►►──SIZE(─┼─────┴──┬────────┬─────┼─)──►◄
          │        └─,──ABOVE─┘   │
          ├─integerK──────────────┤
          ├─integerM──────────────┤
          ├─MAX-integerK─┬──────────┬─┤
          │              └─,──ABOVE─┘ │
          └─MAX-integerM─┬──────────┬─┘
                         └─,──ABOVE─┘
```

You use the SIZE option to specify the amount of virtual storage available to the assembler to perform an in-storage assembly. If you specify an insufficient amount, the assembler uses the work data set (SYSUT1) for intermediate storage.

Default
SIZE(MAX)

Abbreviations
SZ

Restrictions
You cannot specify this option on *PROCESS statements.

SIZE(*integer*K)
Specifies the amount of virtual storage in 1024-byte (1K) increments.

The minimum acceptable value is 200K[2].

SIZE(*integer*M)
Specifies the amount of virtual storage in 1048576-byte (1M) increments.

The minimum acceptable value is 1M.

SIZE(MAX)
Specifies that the assembler requests all the available space[3] *below* the 16MB line in the user region (MVS), or virtual machine (CMS).

SIZE(MAX,ABOVE)
Specifies that the assembler requests all the available space[3] in the user region (MVS), or virtual machine (CMS), that is *above* the 16MB line.

SIZE(MAX-*integer*K)
Specifies that the assembler requests all the available space[3] *below* the 16MB line in the user region (MVS), or virtual machine (CMS), less the amount of *integer*K of storage (1K equals 1024 bytes).

The minimum acceptable value is 1K.

SIZE(MAX-*integer*K,ABOVE)
Specifies that the assembler requests all the available space[3] *above* the 16MB line in the user region (MVS), or virtual machine (CMS), less the amount of *integer*K of storage (1K equals 1024 bytes).

The minimum acceptable value is 1K.

SIZE(MAX-*integer*M)
Specifies that the assembler requests all the available space[3] *below* the 16MB line in the user region (MVS), or virtual machine (CMS), less the amount of *integer*M of storage (1M equals 1048756 bytes).

The minimum acceptable value is 1M.

SIZE(MAX-*integer*M,ABOVE)
Specifies that the assembler requests all the available space[3] *above* the 16MB line in the user region (MVS), or virtual machine (CMS), less the amount of *integer*M of storage (1M equals 1048756 bytes).

The minimum acceptable value is 1M.

Notes:

1. The maximum storage value you can specify might not be available in the user region (MVS), or virtual machine (CMS), after storage has been allocated for the operating system.
2. The minimum amount of working storage required by the assembler is 200K or 10 times the work data set block size, whichever is the greater.
3. When you specify the MAX suboption, the assembler releases 128K back to the user region (MVS), or virtual machine (CMS), for system usage. When you specify the MAX suboption, there might not be enough storage remaining in the user region (MVS), or virtual machine (CMS), to load any exits you specify, or any external functions you use in your assembly.
4. The assembler loads user I/O exits before it obtains the working storage. If the user exit obtains storage, then it reduces the amount available for the assembler.
5. The assembler loads external function routines after it obtains working storage. If you use external functions in your program, you should reduce the value you specify in the SIZE option, to allow storage space for the external function modules, and any storage they might acquire.

High Level Assembler acquires the amount of storage you specify in the SIZE option from the user region (MVS), or virtual machine (CMS). The assembler only requires a work data set when it has insufficient virtual storage to perform an in-storage assembly. An in-storage assembly usually reduces the elapsed time needed to complete the assembly.

The statistics in the *Diagnostic Cross Reference and Assembler Summary* section of the assembly listing shows the amount of storage the assembler used and an estimate of the amount of storage it requires to perform an in-storage assembly. If you do not provide a work data set, you must specify a large enough value on the SIZE option to allow the assembler to perform an in-storage assembly.

Use the STORAGE operand of the installation default options macro, ASMAOPT, to specify the equivalent of the ABOVE suboption.

SYSPARM

```
►►──SYSPARM(──┬────────┬──)──►◄
              └─string─┘
```

Default
The &SYSPARM system variable is set to NULL.

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

string
Specifies the character string the assembler assigns to the &SYSPARM system variable symbol. The character string is up to 255 characters in length. Any character may be included in the string, subject to the rules for building character strings defined in *Language Reference*. If the string includes blanks, commas, or parentheses, it must be enclosed in single quotation marks. Any parentheses inside the string must be paired.

Under MVS, you must use two single quotation marks to represent a single quotation mark, and two ampersands to represent a single ampersand. For example:

```
PARM='OBJECT,SYSPARM((&&AM,''EO).FY)'
```

assigns the following value to &SYSPARM:

```
(&AM,'EO).FY
```

Under MVS, JCL restrictions limit the length of the SYSPARM value as explained in the notes below. When you call the assembler from a problem program (dynamic invocation), you can specify a SYSPARM value up to 255 characters long.

Under CMS, you can specify SYSPARM(?). This causes the assembler to issue the following message at your terminal:

```
ENTER SYSPARM:
```

In response to this message you can enter up to 255 characters. To specify a SYSPARM value of ?, you must specify SYSPARM(?) and enter ? at the terminal prompt.

MVS Batch: Under MVS Batch, the restrictions imposed upon the PARM field limit the maximum length of the SYSPARM value to 56 characters, unless you use symbolic procedure parameters to substitute for the value, or the value contains commas that can be used as breaking points between records. Consider the following example:

```
// EXEC ASMAC,PARM=(ADATA,MXREF(XREF),
// 'SYSPARM(Parametervalue........................................)')
↑  ↑          ↑                                                     ↑
1  4          13                                                    68
```

Because SYSPARM uses parentheses, you must surround it with single quotation marks. The leftmost column that you can use is column 4 on a continuation record. A quotation mark and the keyword, as well as the closing quotation mark, must appear on that line. In addition, either a right parenthesis, indicating the end of the PARM field, or a comma, indicating that the PARM field is continued on the next record, must be coded before or in the last column of the statement field (column 71). Because of these JCL rules, you cannot continue the SYSPARM value on the next record.

TSO: Under TSO, the restriction of the length of the PARM parameter of the CALL command limits the maximum length of the SYSPARM value to 91 characters.

TERM

```
      ┌─NOTERM────────────────────────┐
►►────┴──┬──────────────────────────┬─┴──►◄
         └─TERM(──┬─WIDE───┬──)─────┘
                  └─NARROW─┘
```

Default
NOTERM

Abbreviations
None

Restrictions
This option is not allowed on *PROCESS statements.

TERM
Is equivalent to WIDE. See the description of TERM(WIDE) below.

TERM(WIDE)
Instructs the assembler to write error messages to the terminal data set. You define the terminal data set with the SYSTERM ddname.

TERM(NARROW)
Instructs the assembler to write error messages to the terminal data set. You define the terminal data set with the SYSTERM ddname. The NARROW suboption instructs the assembler to compress multiple consecutive blanks into a single blank.

NOTERM
Instructs the assembler not to write error messages to SYSTERM.

TEST

```
    ┌─NOTEST─┐
►►──┴─TEST───┴──►◄
```

Default
NOTEST

Abbreviations
None

TEST
Specifies that the object module contains the special source symbol table (SYM records) required by the TSO TEST command.

NOTEST
Specifies that the object module does not contain the special source symbol table (SYM records) required by the TSO TEST command.

If you specify the TEST option with the XOBJECT option, the assembler ignores the TEST option.

TRANSLATE

```
    ┌─NOTRANSLATE────────────┐
►►──┴─TRANSLATE(──┬─AS─┬─)──┴──►◄
                  └─xx─┘
```

Default
NOTRANSLATE

Abbreviations
TR

Restrictions
This option is not allowed on *PROCESS statements.

TRANSLATE(AS)
Specifies that characters contained in character (C-type) data constants (DCs) and literals are converted into ASCII characters using the ASCII translation table provided with High Level Assembler.

TRANSLATE(*xx*)
Specifies that characters contained in character (C-type) data constants (DCs) and literals are converted using a user-supplied translation table. The translation table must be named ASMALT*xx*.

Notes:

1. The assembler does not convert DBCS strings.
2. The assembler searches for the user-supplied translation table load module in the standard load module search order. See also Appendix L, "How to Generate a Translation Table" on page 325.

USING

```
         ┌─USING(MAP,WARN(15))──────────────────────┐
         │            ┌─,──────────────┐             │
►►───────┼─USING(──▼─┬─MAP────────────┴──)─────────┼──►◄
         │           ├─WARN(n)───────┤             │
         │           ├─LIMIT(xxxx)───┤             │
         │           ├─NOLIMIT───────┤             │
         │           ├─NOMAP─────────┤             │
         │           └─NOWARN────────┘             │
         └─NOUSING─────────────────────────────────┘
```

Default
USING(MAP,WARN(15))

Abbreviations
US / NOUS

LIMIT(*xxxx*)
This suboption, when used in combination with the WARN(8) suboption, specifies the maximum displacement that base-displacement address resolution checks.

xxxx is the decimal value of the displacement, and must be less than or equal to 4095. X'xxx' may also be used to specify the value in hexadecimal. If specified, this value must be less than or equal to X'FFF'.

If more than one base register is specified in a USING statement, the value specified in the LIMIT suboption is used only to check the maximum displacement from the last specified base register. For example, if USING(LIMIT(X'F00'),WARN(8)) were specified at invocation, the messages would be issued as in Figure 23.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                           HLASM R2.0  1995/04/01 08:00

000000                                1 EXAMPLE  CSECT
                 R:AB 00000           2          USING EXAMPLE,10,11
                                                 .
                                                 .
000190 47F0 AF80           00F80   174           B     LABEL111           1
                                                 .
                                                 .
                           00F80   496 LABEL111 EQU   *
                                                 .
                                                 .
001504 47F0 BF40           01F40   908           B     LABEL999           2
   ASMA304W  ** WARNING **  Displacement exceeds LIMIT value specified
                                                 .
                                                 .
                           01F40  1510 LABEL999 EQU   *
001F40 07FE                       1511           BR    14
                                  1512           END
```

Figure 23. Effect of the LIMIT suboption

Although the resolved displacement of the instruction at **1** is greater than the specified limit, error diagnostic message ASMA304 is not issued because register 10 was not the last specified base register. However, the instruction at **2** causes the message to be issued because register 11 was the last specified base register.

NOLIMIT
This suboption specifies that displacements are not checked. Specifying this suboption is equivalent to specifying the LIMIT suboption with a value of 4095 or X'FFF'.

MAP
This suboption instructs the assembler to produce the *USING Map* section of the assembler listing. For more information, see "USING Map" on page 28.

NOMAP
This suboption specifies that no USING map is produced.

WARN(*n*)
This suboption specifies the conditions under which warning error diagnostic messages are issued. Each condition has an associated condition number, *n*. The allowable values for *n* are:

1 **Nullified USINGs**: the assembler issues message:

- ASMA300 when a previous active ordinary (unlabeled) USING's range coincides with and supersedes that of the USING being processed
- ASMA301 when the range of the USING being processed coincides with and supersedes that of a previous active ordinary (unlabeled) USING.

2 **R0 based USINGs**: the assembler issues message ASMA302 when a USING specifies R0 as a base register, with a non-zero absolute or relocatable expression for the base address.

4 **Multiple resolutions**: the assembler issues message ASMA303 when multiple resolutions are possible for an implicit address.

8 **LIMIT**: the assembler issues message ASMA304 when the calculated displacement in any valid resolution exceeds the threshold specified in the LIMIT suboption. This has no effect if the LIMIT suboption is not specified.

Several conditions may be combined by adding together the associated condition numbers. For example, specifying WARN(12) would request the assembler to issue warning diagnostic messages for the conditions with condition numbers 4 and 8.

NOWARN
This suboption specifies that no USING warning messages are issued.

The USING suboptions LIMIT, MAP, and WARN are specified in the installation default options as LIMIT, MAP, and WARN.

XOBJECT

```
          ┌─NOXOBJECT──────────────────┐
          │             ┌─NOADATA─┐    │
►►────────┴─XOBJECT(────┴─ADATA───┴─)──┴────►◄
```

Default
NOXOBJECT

Abbreviations
XOBJ / NOXOBJ

Restrictions
You cannot specify this option on *PROCESS statements.

XOBJECT
Instructs the assembler to produce an extended format object data set. You define the object data set with the SYSLIN ddname. Refer to *DFSMS/MVS Version 1 Release 3 Program Management*, SC26-4916 for details about the extended format object data set.

XOBJECT(NOADATA)
The same as XOBJECT without a suboption.

XOBJECT(ADATA)
Instructs the assembler to produce an extended object format data set, and include ADATA text record in the object data set.

Use the XOBJADATA operand on the ASMAOPT macro to specify the ADATA suboption in the installation default options.

NOXOBJECT
Instructs the assembler not to produce the extended object format data set.

Notes:

1. The XOBJECT option is mutually exclusive with the DECK option and the OBJECT option.
2. You should specify the LIST(133) option when you specify the XOBJECT option. If the logical record length of the listing data set is less than 133, the assembler truncates the listing lines.
3. The extended object format does not support TESTRAN (SYM) records. If you specify the TEST option with the XOBJECT option, the assembler issues a diagnostic error message.

XREF

```
       ┌─XREF(SHORT,UNREFS)────────────────┐
►►─────┼─XREF(──┬─FULL─────────────┬─)─────┼───►◄
       │        │  ┌─,──────────┐  │       │
       │        └──▼─┬─SHORT────┴──┘       │
       │             └───UNREFS───┘        │
       └─NOXREF────────────────────────────┘
```

Default
XREF(SHORT,UNREFS)

Abbreviations
None

XREF(FULL)
Instructs the assembler to produce the *Ordinary Symbol and Literal Cross Reference* section of the assembler listing. This includes symbols that are defined, but never referred to.

XREF(SHORT)
Instructs the assembler to produce the *Ordinary Symbol and Literal Cross Reference* section of the assembler listing. Symbols that are defined, but not referred to, are not included in the cross reference listing. SHORT may be specified with the UNREFS suboption to produce a list of unreferenced symbols. The SHORT suboption can not be specified with the FULL suboption.

XREF(UNREFS)
Instructs the assembler to produce the *Unreferenced Symbols Defined in CSECTs* section of the assembler listing. The symbols are listed in symbol name order. UNREFS may be specified with the SHORT suboption to produce a cross reference list of referenced symbols. The UNREFS suboption can not be specified with the FULL suboption.

NOXREF
Specifies that symbol cross reference information is not generated as part of the assembly listing.

Any suboption you specify overrides the suboptions specified in the installation default options, unless the XREF option is fixed.

If you specify the XREF option more than once, the assembler uses the last one you specify. For example, if you specify `XREF(SHORT),XREF(UNREFS)`, the assembler uses `XREF(UNREFS)`. To use both suboptions specify `XREF(SHORT,UNREFS)`.

Chapter 4. Providing User Exits

This chapter describes how you can provide user exits to complement the assembler's data set processing. It describes the type of exits, how to specify them during assembly, and the details you need to write an exit.

Exit Types

You can instruct the assembler to call the following types of exits:

SOURCE Exit: To process Primary Input records.

You use a SOURCE exit to replace or complement the assembler's primary input data set processing. You can use it to supply primary input records to the assembler, or monitor and modify records the assembler has read before the assembler processes them. The exit can supply all the primary input records, or extend the primary input by supplying additional records during the assembly. The exit can also discard records.

LIBRARY Exit: To process Library Input records.

You use a LIBRARY exit to replace or complement the assembler's macro call (MACRO) and copy code (COPY) library processing. You can use it to supply MACRO and COPY library records to the assembler, or monitor and modify records the assembler has read before the assembler processes them. The exit can supply all the MACRO and COPY library records, or extend the library input processing by supplying additional MACRO and COPY records during the assembly. The exit can also discard records.

LISTING Exit: To process Listing Output records.

You use a LISTING exit to replace or complement the assembler's listing output processing. You can use it to write the listing records the assembler supplies, or monitor and modify the records before the assembler writes them to the listing data set. The exit can write all the listing records, or supply additional listing records for the assembler to write during the assembly. The exit can also discard records.

OBJECT and PUNCH Exit: To process Object and Punch Output records.

You use an OBJECT and PUNCH exit to replace or complement the assembler's object module output processing. You can use it to write the object module records the assembler supplies, or monitor and modify the records before the assembler writes them to the object data set. The exit can write all the object module records, or supply additional records for the assembler to write during the assembly. The exit can also discard records.

ADATA Exit: To process Associated Data Output records.

You use an ADATA exit to monitor the associated data records that are written by the assembler. The ADATA exit cannot modify the records, discard records, or provide additional records.

TERM Exit: To process Terminal Output records.

You use a TERM exit to replace or complement the assembler's terminal output processing. You can use it to write the terminal records the assembler supplies, or monitor and modify the records before the assembler writes them to the terminal data set. The exit can write all the terminal records, or supply additional terminal records for the assembler to write during the assembly. The exit can also discard records.

Specifying User Exits

You use the EXIT option to specify the name of one or more user exits to load, and optionally pass to the exit a character string up to 64 characters long that is processed during assembly initialization. You can use the EXITCTL assembler instruction to pass data from the assembler source program to the user exit during the assembly.

The *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing shows the statistics for records processed by the user exits during the assembly. See “EXIT” on page 41 for the syntax of the EXIT assembler option.

The following table lists the exit type, the EXIT suboption, the default data set *ddname* that the exit corresponds to, and a Page number reference to the section that describes how the assembler processes the exit:

Exit Type	**Exit Suboption**	*ddname*	**Page Number**
SOURCE	INEXIT	SYSIN	81
LIBRARY	LIBEXIT	SYSLIB	83
LISTING	PRTEXIT	SYSPRINT	87
PUNCH	OBJEXIT	SYSPUNCH	90
OBJECT	OBJEXIT	SYSLIN	90
ADATA	ADEXIT	SYSADATA	93
TERM	TRMEXIT	SYSTERM	94

Loading User Exits

The assembler loads the user exits during initialization. The assembler must be able to locate the user exits as follows:

Under MVS: The user exit must be a link-edited load module that is located in a partitioned data set in the standard search sequence. The user exit can also be located in the Link Pack Area (LPA).

If you use the same exit load module for more than one user exit type, for example as a SOURCE and LISTING exit, the load module can be loaded more than once, depending on the link edit options specified.

Under CMS: The user exit must be a MODULE that is located on one of the accessed disks. You generate the module using the CMS LOAD and GENMOD commands. When the LOAD command is issued, the RLDSAVE option must be specified to make the module relocatable. If RLDSAVE is not specified, it might result in the assembler program or data storage being overlaid.

If you use the same exit load module for more than one user exit type, for example as a SOURCE and LISTING exit, only one copy of the module is loaded.

The user exits may be link-edited in any addressing mode (AMODE) and residency mode (RMODE).

Calling User Exits

The assembler calls user exits using the standard OS Linkage conventions. The user exit can be written in any language that conforms to the following:

- Uses standard OS linkage conventions
- Can be called many times via the exit module entry point
- Retains storage for variables across invocations

Refer to the language's *Programming Guide* to find out if you can use it to write a user exit for the assembler.

The contents of the registers upon entry to the user exit are as follows:

Register 0	Undefined
Register 1	Address of Exit Parameter list, see Figure 24 on page 69.
Register 2 through 12	Undefined
Register 13	Address of 72 byte save area
Register 14	Return address
Register 15	Address of Entry point of user exit

Exit Parameter List

The assembler passes an Exit Parameter list to the user exit. On entry to the exit, Register 1 contains the address of this parameter list. Each exit is passed a separate copy of the parameter list. The parameter list includes a pointer to an Exit-Specific Information block that contains specific information for each exit type. High Level Assembler provides macro ASMAXITP to map the Exit Parameter list and the Exit-Specific Information block. Figure 24 on page 69 describes the format of the Exit Parameter list, Figure 28 on page 78 describes the format of the Exit-Specific Information block for the LISTING exit, and Figure 29 on page 78 describes the format of the Exit-Specific Information block for the other exit types.

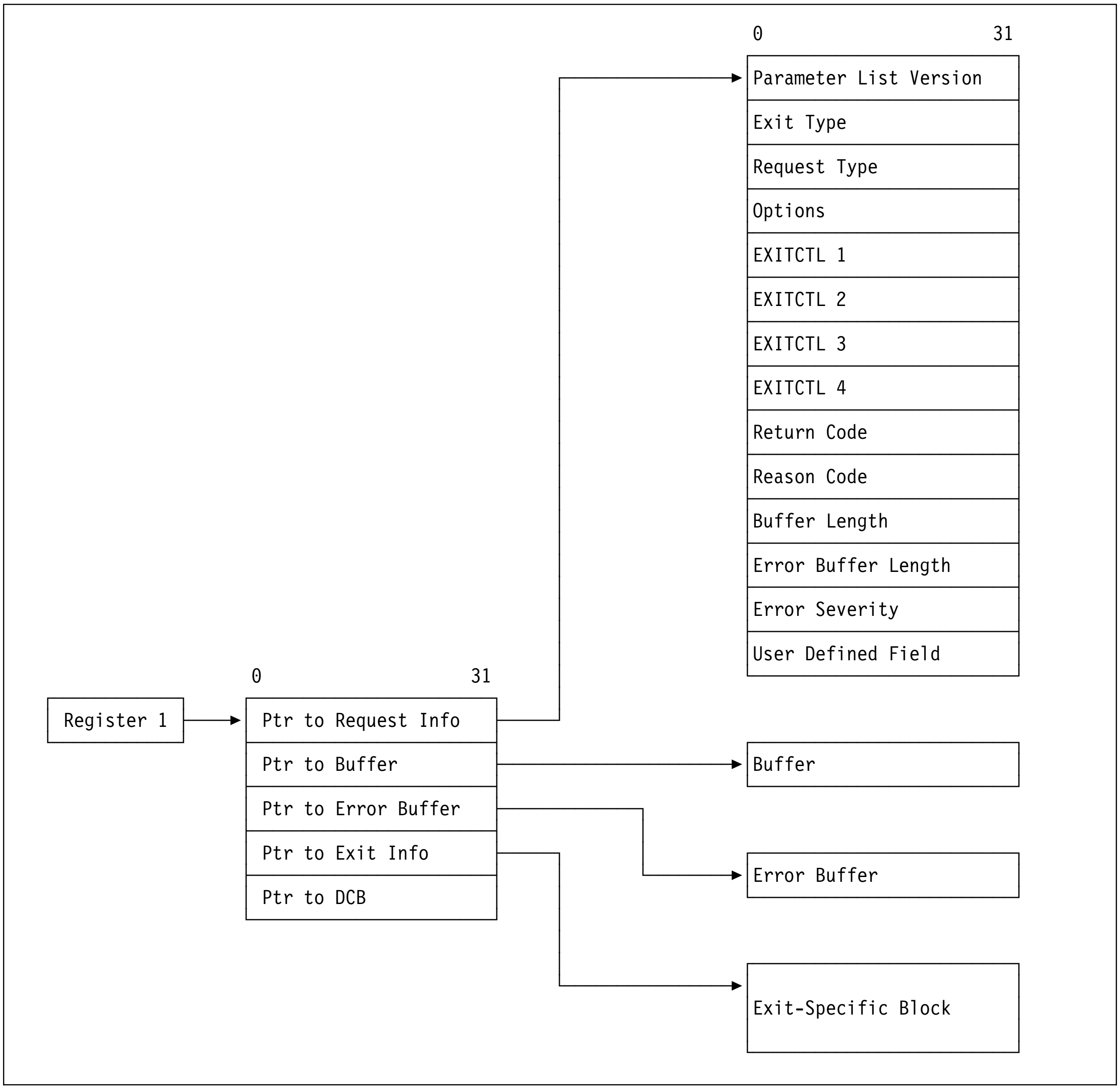

Figure 24. Exit Parameter List Format

The following sections describe the Exit Parameter list.

Request Info Pointer

The request info pointer points to a list of fullword fields that describe the exit request. The assembler sets this pointer, which is always a valid address.

Parameter List Version

A fullword identifying the version of the parameter list. For High Level Assembler Release 2 this field contains a value of 2.

Exit Type

A fullword identifying the type of exit being called. You use this field when the exit handles more than one exit type. The exit type is identified by the following values:

1 SOURCE Input

2 LIBRARY Input

3 LISTING Output

4 PUNCH Output

5 OBJECT Output

6 ADATA Output

7 TERM Output

The assembler always sets this field.

Request Type

A fullword identifying the type of processing request. The request type is identified by the following values:

1 OPEN - exit receives control before any input or output processing

2 CLOSE - exit receives control before the assembler does any close processing

3 READ - exit receives control to provide a record to the assembler

4 WRITE - exit receives control to write a record provided by the assembler

5 PROCESS (for exit types other than LIBRARY) - exit receives control to inspect or manipulate the record provided by the assembler

5 PROCESS MACRO (for LIBRARY exit type only) - exit receives control to inspect or manipulate the macro definition record provided by the assembler

6 PROCESS COPY (for LIBRARY exit type only) - exit receives control to inspect or manipulate the copy member record provided by the assembler

7 FIND MACRO (for LIBRARY exit type only) - exit receives control to locate the specified library macro

8 FIND COPY MEMBER (for LIBRARY exit type only) - exit receives control to locate the specified copy member

The assembler always sets this field.

Options

A fullword that provides additional information to the exit.

For the SOURCE and LIBRARY Exits: The following values are provided:

0 No additional information available.

1 New information is available in the Exit-Specific Information block. The assembler updates this block whenever the primary input data set changes.

For example, the SOURCE input might be a concatenation of data sets. When the first data set is opened, and when each subsequent concatenated data set is opened, this value is set to 1 to inform the exit that a data set switch has occurred. It is also set for LIBRARY processing to inform the exit which data set in the concatenation is being used to provide the specific member.

2 For the LIBRARY exit, when the request type is FIND MACRO or FIND COPY, this indicates that the copy code or macro should resume after the saved record position.

3 For the LIBRARY exit, when the request type is FIND MACRO or FIND COPY, this indicates that copy code or macro definition is currently being processed. The user exit should save the position within the current member to allow it to be resumed when the new member has been processed.

See "Nesting COPY Instructions and Macro Definitions" on page 85.

For the LISTING exit: The following decimal values are provided:

00 No additional information available
10 *High Level Assembler Options Summary* heading line
11 *High Level Assembler Options Summary* detail line
15 *High Level Assembler Options Summary* diagnostic message
20 *External Symbol Dictionary* heading line
21 *External Symbol Dictionary* detail line
30 *Source and Object* heading line
31 *Source and Object* machine instruction
32 *Source and Object* DC/DS instruction
33 *Source and Object* comment
34 *Source and Object* statement in error
35 *Source and Object* diagnostic message
36 *Source and Object* other
40 *Relocation Dictionary* heading line
41 *Relocation Dictionary* detail line
50 *Ordinary Symbol and Literal Cross Reference* heading line
51 *Ordinary Symbol and Literal Cross Reference* detail line
52 *Unreferenced Symbols Defined in CSECTs* heading line
53 *Unreferenced Symbols Defined in CSECTs* detail line
60 *Macro and Copy Code Source Summary* heading line
61 *Macro and Copy Code Source Summary* detail line
62 *Macro and Copy Code Cross Reference* heading line
63 *Macro and Copy Code Cross Reference* detail line
70 *DSECT Cross Reference* heading line
71 *DSECT Cross Reference* detail line
80 *USING Map* heading line
81 *USING Map* detail line
90 *Diagnostic Cross Reference and Assembler Summary* heading line

91 *Diagnostic Cross Reference and Assembler Summary* detail line

For the PUNCH, OBJECT, and ADATA Exits: This field contains **0**.

The assembler sets this field.

EXITCTLn

Four fullword fields containing the exit-control values for this exit type. Exit-control values are set by the EXITCTL assembler instruction during the assembly.

For the SOURCE and LIBRARY Exits: The new EXITCTL values are available to the exit when the input record following the EXITCTL instruction is passed to the exit.

For the LISTING, ADATA and TERM Exits: The new EXITCTL values are available to the exit with the output record containing the EXITCTL instruction.

For the OBJECT and PUNCH Exits: The new EXITCTL values are available to the exit when the next object module record is passed to the exit. This may happen several source statements after the EXITCTL instruction statement. A possible consequence is that one or more EXITCTL statements can be processed without the exit receiving the EXITCTL parameter values, if they occur between object records.

Return Code

A fullword, set by the exit, that indicates success or failure of the exit call, and the action taken by the assembler on return from the exit. Figure 25 summarizes the return codes.

Figure 25 (Page 1 of 2). User-Exit Return Codes

Exit	**Request**	**RC=0**	**4**	**8**	**16**	**20**
SOURCE	OPEN	Assembler to open the primary input data set[1]	Exit provides records[2]			Operation failed
	CLOSE	Operation successful				Operation failed
	READ	Exit has provided record			End-of-file indicator	Operation failed
	PROCESS	Accept record	Discard record			Operation failed
LIBRARY	OPEN	Assembler to open its library [1]	Exit has opened its library [3]	Exit has opened its library, assembler to open its library		Operation failed
	CLOSE	Operation successful				Operation failed
	READ	Exit has provided record			EOD on input source	Operation failed
	PROCESS (macro or copy member)	Accept record	Discard record			Operation failed

Figure 25 (Page 2 of 2). User-Exit Return Codes

Exit	Request	RC=0	4	8	16	20
	FIND (macro or copy member)	Operation successful	Member not found - search assembler library if available			Operation failed
LISTING PUNCH OBJECT TERM	OPEN	Assembler opens the output data set[1]	Exit has opened its output data set[4]			Operation failed
	CLOSE	Operation successful				Operation failed
	WRITE	Exit has written record				Operation failed
	PROCESS	Accept record	Discard record			Operation failed
ADATA[5]	OPEN	Operation successful. Exit has initialized successfully.				Operation failed
	CLOSE	Operation successful				Operation failed
	PROCESS	Operation successful				Operation failed

Notes:

1. The assembler only uses the PROCESS and CLOSE operation codes on subsequent calls.
2. The assembler only uses the READ and CLOSE operation codes on subsequent calls.
3. The assembler only uses the READ, FIND, and CLOSE operation codes on subsequent calls.
4. The assembler only uses the WRITE and CLOSE operation codes on subsequent calls.
5. The ADATA exit can only be used to monitor records written to the associated data file. Unlike other exit types, the ADATA exit cannot be used to replace assembler I/O processing, and can not manipulate the data in the records passed to it by the assembler.

Reason Code

A fullword, set by the exit, to qualify the return code. Figure 26 shows reason codes for each exit type, and which request they are checked after.

Figure 26. User Exit Reason Codes

Exit	Request	Reason Code=0	4
SOURCE	OPEN	No additional information	Input source information available
	READ	No additional information	Input source information available
	PROCESS	No additional information	Return to exit with empty buffer
LIBRARY	FIND (macro or copy member)	No additional information	Input source information available
	PROCESS (macro or copy member)	No additional information	Return to exit with empty buffer
LISTING	OPEN	No additional information	When return code is 0, reason code 4 indicates the exit has provided a listing print line length in the buffer length field. When return code is 4, reason code 4 indicates the exit has provided the listing data set information.
LISTING PUNCH OBJECT TERM	PROCESS	No additional information	Return to exit with empty buffer
PUNCH OBJECT	OPEN	No additional information	Exit has provided the output data set information
TERM	OPEN	No additional information	When return code is 0, reason code 4 indicates the exit has provided a terminal line length in the buffer length field. When return code is 4, reason code 4 indicates the exit has provided the terminal data set information.

Buffer Length

A fullword containing the length of the area pointed to by the buffer pointer.

For OPEN Requests: This field contains the length of the character string you specified in the EXIT assembler option.

For WRITE and PROCESS Requests: This field contains the length of the record pointed to by the buffer pointer.

For READ Requests: This field contains the length of the area pointed to by the buffer pointer where the exit may return a record to the assembler.

All Other Requests: This field contains zero.

Setting the Length: When either the SOURCE, LIBRARY, PUNCH, or OBJECT exit is invoked for a READ, WRITE, or PROCESS request, the assembler sets the buffer length to 80.

If you specify the XOBJECT assembler option, and the OBJECT exit is invoked, the buffer length might be fixed-length 80, or variable-length depending on the JCL (MVS) you supply. The maximum value for variable-length records is 8212.

For an OPEN request the LISTING exit can use this field to pass the listing line length to the assembler. The exit indicates that it has provided a print line length by setting the return code to 0 and the reason code to 4. The line length must be in the range 121 to 255. If it is any other value the assembler issues message ASMA402 and does not call the exit to process listing records.

For all other calls to the LISTING exit, the assembler sets this field to the length determined during the OPEN call.

The TERM exit can use this field to indicate to the assembler the length of the terminal record. This may be done when the exit is invoked with an OPEN request. The exit indicates that it has provided a terminal line length by setting the Return Code to 0 and the Reason Code to 4. The value must not be zero, and must not be greater than 255. If the value is not correct, the assembler issues message ASMA404 and does not call the exit to process terminal records.

For all other calls to the TERM exit, the assembler sets this field to the length determined during the OPEN call.

Error Buffer Length

An unsigned fullword, set by the exit, that contains the length of the text pointed to by the error buffer pointer. The maximum length is 255 bytes. If the exit specifies a larger value the assembler uses 255.

The assembler uses this length to determine whether to issue an error message. If the length is greater than zero, the text in the error buffer is inserted into one of the messages ASMA700 to ASMA704. The assembler selects which message to issue by checking the value of the error severity field.

Error Severity

A fullword, set by the exit, that contains the severity code the assembler uses to determine which diagnostic message to issue.

The severity code should be a value of 0, 4, 8, 12 or 16. If the severity code is not one of these values it is rounded up to the nearest value or, if the severity code is greater than 16, it is reset to 16.

The values 0, 4, 8, 12 and 16 correspond to diagnostic messages ASMA700 to ASMA704, respectively. For example, severity code of 4 causes the assembler to issue message ASMA701. Figure 27 summarizes the return code values and the associated diagnostic message.

Figure 27. Error Severity and Associated Diagnostic Message

Return Code Specified	Return Code Used	Associated Message
0	0	ASMA700
3-4	4	ASMA701
5-8	8	ASMA702
9-12	12	ASMA703
> 13	16	ASMA704

User Defined Field

A fullword, set to zero by the assembler before it calls the exit with an OPEN request. The exit can use this field to store information (such as the address of acquired storage) between calls. This field is separately maintained for each exit type and is preserved across all calls until the exit is closed. The assembler does not modify or interpret this field.

Buffer Pointer

The buffer pointer points to a fullword that contains the address of the area containing a record to be processed by the exit.

For OPEN Requests: This field contains the character string from the EXIT assembler option. If you did not specify a character string in the EXIT assembler option this area contains zeros, and the buffer length field is set to zero.

For READ Requests: This field points to an empty buffer area.

For PROCESS and WRITE Requests: This field points to the record supplied by the assembler.

All Other Requests: This field is set to zero.

Error Buffer Pointer

The error buffer pointer points to a fullword that contains the address of the error text buffer.

The assembler sets this pointer. If you want the assembler to issue a message on behalf of the exit, you must supply the text of the error messages in the area pointed to by the error buffer pointer. The text can be up to 255 characters. The exit must place the length of the text in the error buffer length field. The assembler selects a message number based on the value you place in the error severity field.

Exit-Specific Information Pointer

The exit-specific information pointer is a fullword that contains the address of the Exit-Specific Information block. The assembler sets this pointer. See “Exit-Specific Information Block” on page 77 for more details.

DCB Pointer

The DCB pointer is a fullword that contains the address of the Data Control Block.

The assembler sets this address which points to the applicable DCB for the exit being called as follows:

Exit	**DCB**
SOURCE	SYSIN
LIBRARY	SYSLIB
LISTING	SYSPRINT
PUNCH	SYSPUNCH
OBJECT	SYSLIN
ADATA	SYSADATA
TERM	SYSTERM

When an exit is invoked with an OPEN request, the data set referred to by the DCB is not open, and the contents of the DCB might not be complete.

When an exit is invoked with a PROCESS request, the exit may use the DCB to obtain additional information about the data set or member being used. For example, under MVS, the exit can obtain user information from a PDS directory by using the BLDL system macro.

User Exit Work Area

The user exit parameter list contains a user defined field that you can use to retain information between calls to the exit. For example, you can use this field to hold the address of storage the exit might acquire for a reentrant work area. This field is initialized to zero before the assembler calls the exit with the OPEN request. The assembler does not inspect or modify this field after the OPEN request.

Error Handling

Exit Failure Handling: You can signal an exit failure for any call to the exit by setting the return code field in the Exit Parameter list to 20. When the assembler receives this return code it issues message ASMA940, and stops the assembly. You can provide the assembler with additional information to insert in the message text by placing the information in the error buffer pointed to by error buffer pointer, and the length of the information in the error buffer length.

If the exit sets the return code field in the Exit parameter list to any value other than those described in Figure 25 on page 72, the assembler issues message ASMA940 and stops the assembly.

User Error Handling: You can instruct the assembler to produce an error message after any call to the exit by placing information in the error buffer pointed to by error buffer pointer, and the length of the information in the error buffer length. You can indicate the severity of the message by placing the severity code in the error severity field. The message is issued as a normal assembler message, and, as such, can be suppressed using the FLAG assembler option.

Exit-Specific Information Block

All user exits are passed an Exit-Specific Information block pointed to by the Exit Parameter list. It contains a list of character data items which describe the data for the exit, and the absolute and relative record numbers for the record passed to the exit. The Exit-Specific Information block passed to all exits, except the LISTING exit, is shown in Figure 29 on page 78. The Exit-Specific Information block passed to the LISTING exit has additional parameters as shown in Figure 28.

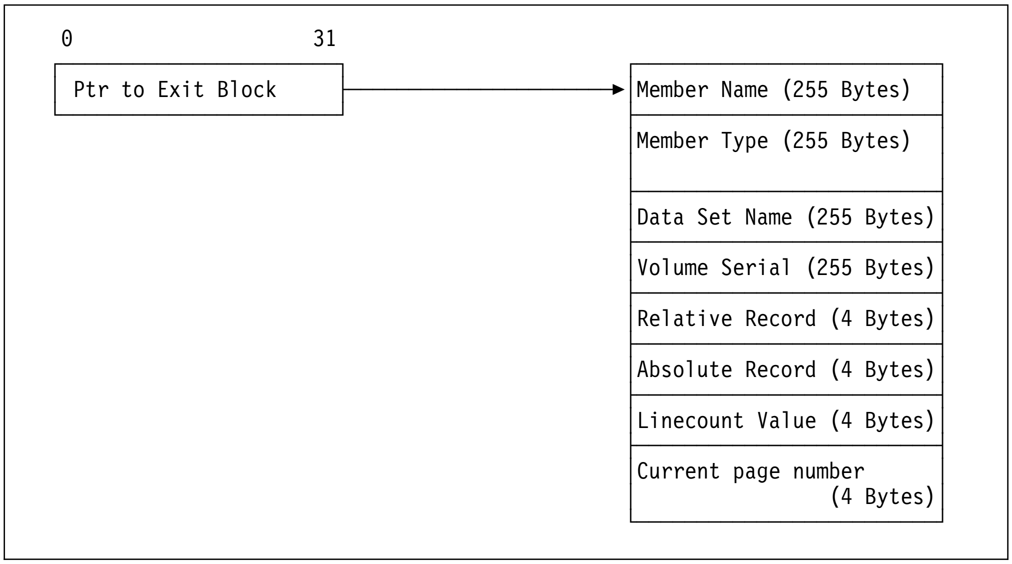

Figure 28. Exit-Specific Information Block - LISTING exit

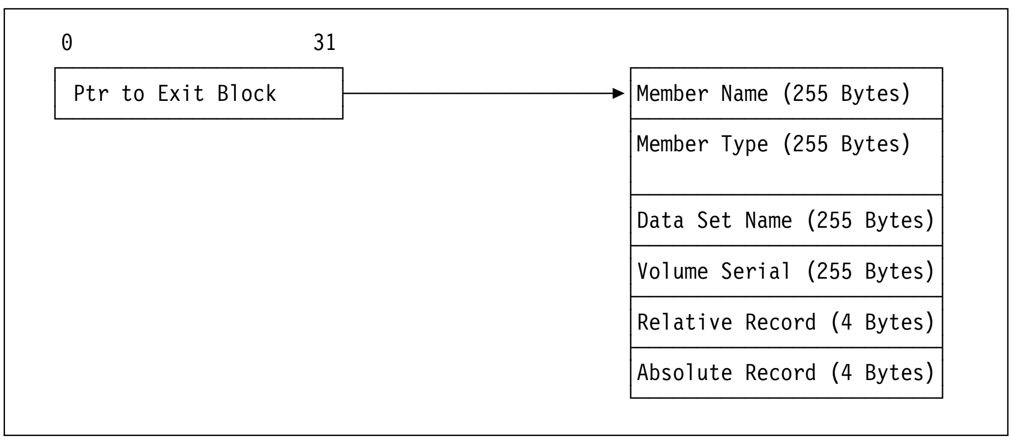

Figure 29. Exit-Specific Information Block - Other exit types

The Exit-Specific Information block consists of the following fields:

Member Name

Member name within the data set. It is always provided for library members and is also provided for data set members under MVS where the data set is a partitioned data set.

The assembler also sets this field as a parameter for the FIND operation. It is left justified and padded with blanks.

For output files, the information should not be updated after it has been set by the OPEN call.

The assembler uses this field to update the system variable symbols, as described in Figure 30.

Member type

Always blank. This field is present to maintain compatibility with High Level Assembler running under VSE.

Data Set Name

The name of the data set from which the last input record was retrieved, or to which the next output record is written. It is left justified and padded with blanks.

For output files, the information should not be updated after it has been set by the OPEN call.

The assembler uses this field to update the system variable symbols, as described in Figure 30.

Volume Serial

Volume serial where the data set is located. It is left justified and padded with blanks.

For output files, the information should not be updated after it has been set by the OPEN call.

The assembler uses this field to update the system variable symbols, as described in Figure 30.

Figure 30. System Variable Symbols

Data Set	Member Name	Data Set Name	Volume Serial
SYSIN	&SYSIN_MEMBER	&SYSIN_DSN	&SYSIN_VOLUME
SYSLIB	&SYSLIB_MEMBER	&SYSLIB_DSN	&SYSLIB_VOLUME
SYSPRINT	&SYSPRINT_MEMBER	&SYSPRINT_DSN	&SYSPRINT_VOLUME
SYSTERM	&SYSTERM_MEMBER	&SYSTERM_DSN	&SYSTERM_VOLUME
SYSPUNCH	&SYSPUNCH_MEMBER	&SYSPUNCH_DSN	&SYSPUNCH_VOLUME
SYSLIN	&SYSLIN_MEMBER	&SYSLIN_DSN	&SYSLIN_VOLUME
SYSADATA	&SYSADATA_MEMBER	&SYSADATA_DSN	&SYSADATA_VOLUME

Relative Record Number

The relative record number is the number assigned to the current record being processed.

PROCESS Calls: For PROCESS calls, it represents the total number of records the assembler has passed to the exit for the current data set. Each time a new data set or library member is opened for input, the relative record number is reset to 1 for the first record. If the new data set is a library member, caused by a macro call or a COPY instruction, the relative record number is returned to the correct sequential number when the macro or COPY processing is complete.

LISTING Exit: The relative record number is reset to 1 for the LISTING exit whenever the assembler forces a page eject.

BATCH Assembler Option: The relative record number is reset to 1 for all output data sets prior to each assembly when the BATCH assembler option is specified.

READ and WRITE Calls: For READ calls and WRITE calls the exit should maintain the relative record number. The assembler uses the relative record number in information messages when you specify the FLAG(RECORD) option. If you specify the ADATA option the assembler includes the record number in the associated data file (ADATA) Source Analysis record.

Absolute Record Number

The absolute record number is the number assigned to the current record being processed. The number is incremented by 1 for each record since the assembly started. For PROCESS calls, it represents the total number of records provided to the exit for the current exit type. It starts at 1, but is not reset when the BATCH assembler option is specified to assemble multiple source programs.

For READ calls and WRITE calls the exit should maintain the absolute record number. The number provided after READ calls is written to the associated data file (ADATA) in the Source Analysis record.

Linecount

This field is only provided for the LISTING exit.

The linecount value is set to the value of the LINECOUNT assembler option before the OPEN call to the LISTING exit. This option contains the number of lines per page in the assembler listing. The exit may change the linecount value only during the OPEN call.

For PROCESS calls, the linecount field contains the number of logical records written to the current listing page. A page eject occurs when the number exceeds the linecount value specified in the LINECOUNT assembler option or during the OPEN call.

Current Page Number

The assembler sets this field to the value of the current page number. Any change the exit makes to this number is ignored.

This field is only provided for the LISTING exit and only for the PROCESS, WRITE and CLOSE call types.

SOURCE Exit Processing

The assembler calls the SOURCE exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly. This is the first call to the exit.

The exit may set the return code in the Exit parameter list to one of the following:

0 Instructs the assembler to open the primary input data set, and supply the primary input records to the exit in later PROCESS calls.

4 Indicates that the exit supplies the primary input records to the assembler in later READ calls. If you wish to provide the assembler with the values for the system variables &SYSIN_DSN, &SYSIN_MEMBER and &SYSIN_VOLUME, the user exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler also shows this information in the *Diagnostic Cross Reference and Assembler Summary* section of the listing, and includes it in the associated data file Job Identification record.

If you provide a character string in the *str1* suboption of the EXIT assembler option, the buffer pointer points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

READ

The assembler calls the exit with a request type of 3 (READ) when the exit is supplying the primary input records.

The exit may set the return code in the Exit Parameter list to one of the following:

0 A record is supplied. The record must be placed in the area pointed to by the buffer pointer field. The record area is 80 characters in length.

The user exit should maintain the absolute record number and the relative record number. These fields are set to zero before the OPEN request. The assembler uses the relative record number in diagnostic messages when you specify the FLAG(RECORD) assembler option. If you specify the ADATA assembler option, the assembler includes both fields in the associated data file Source Analysis record.

If you wish to provide the assembler with the values for the system variables &SYSIN_DSN, &SYSIN_MEMBER and &SYSIN_VOLUME, the user exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. You can provide this information during the OPEN call, or whenever the exit supplies a record to the assembler. If the exit is reading records from concatenated data sets, it should supply the data set information with the first record from each data set in the concatenation.

If the exit does not supply the data set information, the system variables are set to null, and the primary input data set details are not shown in the *Diagnostic Cross Reference and Assembler Summary*, nor are they included in the ADATA Job Identification record.

16 Indicates to the assembler that there are no more records. This is equivalent to end-of-file processing for input data sets.

PROCESS

The assembler calls the exit with a request type of 5 (PROCESS) when the assembler is reading the primary input data set, and it has a record for the exit to process. The address of the record read is in the buffer pointer field, and the length is in the buffer length field. The record length is always 80.

The exit can set the return code in the Exit Parameter list to one of the following:

0 Indicates that the record has been accepted, and the assembler is to process it. The exit may modify the record before it returns control to the assembler. The user exit may also insert extra records in the primary input by setting the reason code to 4. The assembler processes the current record and then calls the user exit with an empty buffer. The exit must place the record in the 80-byte area pointed to by the buffer pointer field. The exit can continue to supply additional records, by setting the reason code to 4. The exit must keep track of when the assembler calls it with an empty buffer, and ensure that it resets the reason code to zero to resume normal processing.

4 Instructs the assembler to discard the current record.

Although the user exit might insert or discard records, the assembler maintains the absolute record number and relative record number.

If the options field is set to 1, the assembler has provided the exit with the current primary input data set information in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler updates this information when it reads the first record of each data set in a data set concatenation.

Figure 31 summarizes the SOURCE exit processing.

Figure 31. SOURCE Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens primary input.
	4	Exit supplies primary input records. If reason code=4, exit supplies data set information.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
3=READ	0	Exit supplies record in buffer. If reason code=4, exit supplies data set information.
	16	Exit indicates end-of-file.
5=PROCESS	0	Record accepted. Exit may modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.

LIBRARY Exit Processing

The assembler calls the LIBRARY exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly. This is the first call to the exit.

The exit can set the return code in the Exit parameter list to one of the following:

0 Instructs the assembler to open the library data set, and supply the macro and copy code library input records to the exit in later PROCESS calls.

4 Indicates that the exit supplies the macro and copy code library records to the assembler in later READ calls. If you wish to provide the assembler with the values for the system variables &SYSLIB_DSN, &SYSLIB_MEMBER and &SYSLIB_VOLUME, the user exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler also shows this information in the *Diagnostic Cross Reference and Assembler Summary* section of the listing, and includes it in the associated data file Library record.

8 Indicates that both the assembler and user exit supply the macro and copy code library records. On return from the exit the assembler opens the library data set. When a macro or copy member is required, the assembler calls the exit with a FIND request. If the member is found by the exit, the exit supplies the records in later READ calls. If the exit cannot find the member, the assembler attempts to find the member in the library data set. If the assembler finds the member, the records are passed to the exit in later PROCESS calls.

If you provide a character string in the *str2* suboption of the EXIT assembler option, the buffer pointer field points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

READ

The assembler calls the exit with a request type of 3 (READ) when the exit is supplying the library records, and after a successful FIND request. For copy members the assembler calls the exit until the exit indicates the end-of-file. For macro definitions the assembler calls the exit until it receives a MEND statement, or the exit indicates the end-of-file.

The exit can set the return code in the Exit parameter list to one of the following:

0 The exit is supplying a record. The record must be placed in the area pointed to by the buffer pointer field. The record area is 80 characters in length.

The user exit should maintain the absolute record number and the relative record number. These fields are set to zero before the OPEN request. The assembler uses the relative record number in diagnostic messages when you specify the FLAG(RECORD) assembler option. If you specify the ADATA assembler option, the assembler includes both fields in the associated data file Source Analysis record.

16 Indicates to the assembler that there are no more records. This is equivalent to end-of-file processing for input members.

PROCESS MACRO or PROCESS COPY

The assembler calls the exit with a request type of 5 (PROCESS MACRO) or 6 (PROCESS COPY) when the assembler is reading members from the library data set, and it has a record for the exit to process. The exit is also called with these request types when both the assembler and the exit are supplying library records (return code 8 from the OPEN call), and the assembler is supplying the record. The address of the record read is in the buffer pointer field, and the length is in the buffer length field. The record length is always 80.

The exit can set the return code in the Exit parameter list to one of the following:

0 Indicates that the record has been accepted, and the assembler is to process it. The exit can modify the record before it returns control to the assembler. The user exit can also insert extra records in the library member by setting the reason code to 4. The assembler processes the current record and then calls the user exit with an empty buffer. The exit must place the record in the 80-byte area pointed to by the buffer pointer field. The exit can continue to supply additional records, by setting the reason code to 4. The exit must keep track of when the assembler calls it with an empty buffer, and ensure that it resets the reason code to zero to resume normal processing.

4 Instructs the assembler to discard the current record.

Although the user exit can insert or discard records, the assembler maintains the absolute record number and relative record number.

If the options field is set to 1, the assembler has provided the exit with the current primary input data set information in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler updates this information when it reads the first record of each data set in a data set concatenation.

FIND MACRO or FIND COPY

The assembler calls the exit with a request type of 7 (FIND MACRO) whenever the assembler cannot find an operation code. The member name field contains the operation code, and is the name of the macro definition that the assembler is searching for.

The assembler calls the exit with a request type of 8 (FIND COPY) whenever the assembler processes a COPY instruction. The member name field contains the name of the copy code member.

If the user exit is supplying the library records, the exit can set the return code in the Exit Parameter list to one of the following:

0 Indicates that the exit supplies the library records. The assembler calls the user exit with later READ calls to retrieve each record.

4 Indicates that the exit is not supplying the macro or copy member, and is equivalent to not finding the member in the library.

If both the assembler and the user exit are supplying the library records, the exit can set the return code in the Exit Parameter list to one of the following:

0 Indicates that the exit supplies the library records. The assembler calls the user exit with later READ calls to retrieve each record.

4 Indicates that the exit is not supplying the macro or copy member, and is equivalent to not finding the member in the library. On return from the exit, the assembler attempts to find the member in the library. If the assembler finds the member, it calls the user exit with later PROCESS MACRO or PROCESS COPY calls passing each record read from the library.

System Variables: If you wish to provide the assembler with the values for the system variables &SYSLIB_DSN, &SYSLIB_MEMBER and &SYSLIB_VOLUME, the user exit must set the return code to 0, the reason code to 4, and place the values in the data set name, member name, and volume serial fields of the exit-specific information block.

If the exit does not supply the data set information, the system variables are set to null, and the library data set details are not shown in the *Diagnostic Cross Reference and Assembler Summary*, nor are they included in the ADATA Library record.

Nesting COPY Instructions and Macro Definitions: The assembler lets you code COPY instructions and macro call instructions in copy code members. It also lets you code COPY instructions in macro definitions. This type of coding is described as *nesting*.

If the exit is processing a member, and supplies a record to the assembler containing a COPY instruction, or a macro call instruction, the assembler calls the exit with a request type of FIND COPY, or FIND MACRO, respectively. In this case, the exit needs to save the position in the currently active member before reading the new copy code or macro member. This enables the exit to resume processing the currently active member after it finishes with the new member.

The assembler indicates that it is processing a new (or nested) member by setting the options field to 3. When the assembler finishes processing the new member and resumes the previous (or outer) member, it issues a FIND call to the exit with the options field set to 2 indicating that the previous member is resumed. After the FIND call is complete, the assembler continues with PROCESS or READ calls to the exit for the previous member.

When the assembler calls the exit with a FIND COPY or FIND MACRO request, and the options field is set to 3, the exit should save the current member control information in a stack.

When the assembler calls the exit with a FIND COPY or FIND MACRO request, and the options field is set to 2, the exit should restore the previous member control information from the stack. The next READ request expects the next record from the previous member.

The assembler does not limit the number of levels of nesting.

There is a corresponding FIND (resume) request for every successful nested FIND request, except under the following situations:

- An END instruction is found while reading a copy code member. The END instruction causes the assembly to stop.
- When the assembler issues a PROCESS call, and provides the last record in a copy code member, and the record is a macro call. In this case there are no more copy records to resume reading.
- When a macro call (outer macro) inside a copy code member in turn issues a macro call (inner macro). In this case, the assembler processes the outer macro to completion, and then begins to generate the outer macro. During generation, the assembler begins to process the inner macro, without issuing a FIND (resume) request for the outer macro or copy code member. The assembler issues a FIND request for each nested macro call, with options set to 3. It does not issue a FIND request for the outer macro, with options set to 2, because the outer macro processing is complete.
- An error occurs during the assembly that prevents the member from being read completely.

If the FIND COPY or FIND MACRO is unsuccessful, the position in the currently active member should not be affected.

Figure 32 summarizes the LIBRARY exit processing.

Figure 32. LIBRARY Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens its library for input.
	4	Exit supplies library records.
	8	Both the assembler and the exit supply library records. The assembler opens its library.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
3=READ	0	Exit supplies record in buffer. Record with MEND statement indicates end of macro member.
	16	Exit indicates end-of-file for member.
5=PROCESS MACRO	0	Record accepted. Exit can modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.
6=PROCESS COPY	0	Record accepted. Exit can modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.
7=FIND MACRO	0	Macro member found by exit; the exit supplies the records. If options=3, the exit should save the current member position. If options=1, the exit should restore the previous member position. If reason code=4, exit supplies data set information.
	4	Macro member not found by exit; the exit does not supply the records.
8=FIND COPY	0	Copy code member found by exit; the exit supplies the records. If options=3, the exit should save the current member position. If options=1, the exit should restore the previous member position. If reason code=4, exit supplies data set information.
	4	Copy code member not found by exit; the exit does not supply the records.

LISTING Exit Processing

You can use the LISTING exit to override the effect of the LIST assembler option. The exit does this by indicating to the assembler that it opens the listing data set and does all listing output processing. Then, as each listing record is passed to the exit, the exit can decide whether to print the record, and where it writes the record. For instance, the exit can write the listing records to a different data set than the assembler would normally write them.

The LISTING exit is not called if you specify the NOLIST assembler option. If you wish to process the listing records in the exit but you do not want the assembler to write the records to the normal output data set, you can do one of the following:

- Instruct the assembler to discard the listing records by setting the exit return code
- Suppress the listing output as follows:

  MVS Provide a `//SYSPRINT DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSPRINT DUMMY` command.

The sections of the listing that are passed to the exit depend on the assembler options you specify. For instance, if you specify the NORLD option, then no *Relocation Dictionary* listing records are passed to the exit.

The assembler calls the LISTING exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly.

The exit may set the return code in the Exit parameter list to one of the following:

0 Instructs the assembler to open the listing data set, and supply the listing output records to the exit in later PROCESS calls.

The exit can set the record length for the listing data set by setting the reason code to 4 and the buffer length field. The buffer length field can be set to any value from 121 to 255. If the value is less than 121 or greater than 255, the assembler issues message `ASMA402` and does not call the exit for any further processing.

The assembler sets the linecount field to the value of the LINECOUNT assembler option. This value is the number of lines per page in the listing. The exit can change the line count to a value of 0, or any value from 10 to 32767. “LINECOUNT” on page 47 describes the LINECOUNT assembler option.

4 Indicates that the exit writes the listing records in later WRITE calls. If you wish to provide the assembler with the values for the system variables &SYSPRINT_DSN, &SYSPRINT_MEMBER and &SYSPRINT_VOLUME, the exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler also shows this information in the *Diagnostic Cross Reference and Assembler Summary* section of the listing, and includes it in the associated data file Output File Information record.

If you provide a character string in the *str3* suboption of the EXIT assembler option, the buffer pointer field points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

WRITE

The assembler calls the exit with a request type of 4 (WRITE) when the exit is writing the listing records. The buffer pointer field points to the listing record, and the buffer length contains the length of the record. Depending on the setting of the ASA assembler option, the record has either an American National Standard or a machine printer control character at the start of the record.

The options field contains a value that represents the type of listing record that is passed. The listing record types, and their corresponding options values, are shown on page 71.

The user exit should maintain the absolute record number and the relative record number. These fields are set to zero before the OPEN request. The assembler uses the relative record number and the linecount value to determine when to start a new page in the assembler listing. A new page is started when the relative record number exceeds the line count.

PROCESS

The assembler calls the exit with a request type of 5 (PROCESS) when the assembler is writing the listing records, and it has a record for the exit to process. The address of the record is in the buffer pointer field, and the length is in the buffer length field. The record has either an American National Standard or a machine printer control character at the start of the record depending on the setting of the ASA option.

The options field contains a value that represents the type of listing record that is passed. The listing record types, and their corresponding options values, are shown on page 71.

The exit can set the return code in the Exit Parameter list to one of the following:

0 Indicates that the record has been accepted, and the assembler is to write it to the listing data set. The exit may modify the record before it returns control to the assembler. The user exit may also insert extra records in the listing by setting the reason code to 4. The assembler writes the current record and then calls the user exit with an empty buffer. The exit must place the additional listing record in the area pointed to by the buffer pointer field. The exit can continue to supply additional records, by setting the reason code to 4. The exit must keep track of when the assembler calls it with an empty buffer, and ensure that it resets the reason code to zero to resume normal processing. The exit must also ensure that a valid printer control character is placed in the first character of the record. The printer control character may be either American National Standard or machine. The exit can check the DCB, pointed to by the DCB pointer field in the Exit parameter list, to find out which printer control character to use.

4 Instructs the assembler to discard the listing record.

Although the user exit can insert or discard records, the assembler maintains the absolute record number and relative record number.

Figure 33 summarizes the LISTING exit processing.

Figure 33. LISTING Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens listing data set. If reason code=4, exit supplies listing line length.
	4	Exit writes listing records. If reason code=4, exit supplies data set information.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record.
5=PROCESS	0	Record accepted. Exit may modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.

OBJECT and PUNCH Exit Processing

When you specify the OBJEXIT suboption of the EXIT assembler option, the assembler calls either the OBJECT user exit or the PUNCH user exit, or both as follows:

- If you specify the OBJECT or XOBJECT assembler option, the assembler calls the OBJECT user exit.
- If you specify the DECK assembler option, the assembler calls the PUNCH user exit.
- If you specify the OBJECT and the DECK assembler options, the assembler calls the user exit as an OBJECT exit, and then as a PUNCH exit. If you specify the XOBJECT assembler option, you cannot specify the DECK assembler option, and therefore the assembler cannot call the PUNCH exit.

You can use the exit to override the effect of the DECK, OBJECT, or XOBJECT assembler options. The exit does this by indicating to the assembler that it opens the output data set and does all the output processing. Then, as each object record is passed to the exit, the exit can decide whether to write the record, and where to write the record. For instance, the exit can write the records to a different data set than the assembler would normally write them.

The exit is not called if you specify the NODECK, NOOBJECT, and NOXOBJECT assembler options. If you wish to process the object records in the exit but you do not want the assembler to write the records to the normal output data set, you can do one of the following:

- Instruct the assembler to discard the records by setting the exit return code
- Suppress the object output as follows:

MVS Provide a `//SYSLIN DD DUMMY` JCL statement, and a `//SYSPUNCH DD DUMMY` JCL statement.

CMS Issue a `FILEDEF SYSLIN DUMMY` command, and a `FILEDEF SYSPUNCH DUMMY` command.

The assembler calls the OBJECT and PUNCH exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly. The exit type field indicates which exit is being called. The OBJECT exit is type 5, and the PUNCH exit is type 4.

The exit can set the return code in the Exit Parameter list to one of the following:

0 Instructs the assembler to open the object data set, and supply the object records to the exit in later PROCESS calls.

4 Indicates that the exit writes the object records in later WRITE calls. If you wish to provide the assembler with the values for the system variables &SYSLIN_DSN, &SYSLIN_MEMBER and &SYSLIN_VOLUME, then during the OPEN call for the OBJECT exit the exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. If you wish to provide the assembler with the values for the system variables &SYSPUNCH_DSN, &SYSPUNCH_MEMBER and &SYSPUNCH_VOLUME, then during the OPEN call for the PUNCH exit the exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler also shows the information for both OBJECT and PUNCH data sets in the *Diagnostic Cross Reference and Assembler Summary* section of the listing, and includes it in the associated data file Output File Information record.

If you provide a character string in the *str4* suboption of the EXIT assembler option, the buffer pointer field points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

WRITE

The assembler calls the exit with a request type of 4 (WRITE) when the exit is writing the object records. The buffer pointer field points to the object record, and the buffer length contains the length of the record. The record length is always 80 bytes when you specify the DECK or OBJECT assembler option. If you specify the XOBJECT assembler option the record length is 80 bytes for fixed-length output or up to 8212 bytes for variable-length output. The record length for variable-length records does not include the 4-byte length of the record descriptor word (RDW), and the buffer pointer field points at the object data, not the RDW.

The user exit should maintain the absolute record number and the relative record number. These fields are set to zero before the OPEN request.

PROCESS

The assembler calls the exit with a request type of 5 (PROCESS) when the assembler is writing the object records, and it has a record for the exit to process. The address of the record is in the buffer pointer field, and the length is in the buffer length field. The record length is always 80 bytes when you specify the DECK or OBJECT assembler option. If you specify the XOBJECT assembler option the record length is 80 bytes for fixed-length output or up to 8212 bytes for variable-length output. The record length for variable-length records does not include the 4-byte length of the record descriptor word (RDW), and the buffer pointer field points at the object data, not the RDW.

The exit can set the return code in the Exit parameter list to one of the following:

0 Indicates that the record has been accepted, and the assembler is to write it to the object data set. The exit can modify the record before it returns control to the assembler. The user exit can also insert extra records in the object data set by setting the reason code to 4. The assembler writes the current record and then calls the user exit with an empty buffer. The exit must place the additional object record in the area pointed to by the buffer pointer field. The exit can continue to supply additional records, by setting the reason code to 4. The exit must keep track of when the assembler calls it with an empty buffer, and ensure that it resets the reason code to zero to resume normal processing.

4 Instructs the assembler to discard the record.

Although the user exit can insert or discard records, the assembler maintains the absolute record number and relative record number.

Figure 34 summarizes the OBJECT and PUNCH exit processing.

Figure 34. OBJECT and PUNCH Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens object data set.
	4	Exit writes object records. If reason code=4, exit supplies data set information.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record.
5=PROCESS	0	Record accepted. Exit can modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.

ADATA Exit Processing

When you specify the ADEXIT suboption of the EXIT assembler option, the assembler calls the ADATA user exit if you also specify the ADATA assembler option.

The ADATA exit is not called if you specify the NOADATA assembler option. If you wish to process the associated data records in the exit but you do not want the assembler to write the records to the normal output data set, you can do one of the following:

- Instruct the assembler to discard the records.
- Suppress the associated data output as follows:

  MVS Provide a `//SYSADATA DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSADATA DUMMY` command.

The assembler calls the ADATA exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly.

If you provide a character string in the *str5* suboption of the EXIT assembler option, the buffer pointer field points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

PROCESS

The assembler calls the exit with a request type of 5 (PROCESS) when the assembler is writing the associated data records, and it has a record for the exit to process. The address of the record read is in the buffer pointer field, and the length is in the buffer length field. The record length for variable-length records does not include the 4-byte length of the record descriptor word (RDW), and the buffer pointer field points at the ADATA header data, not the RDW. The assembler ignores all modifications to the ADATA records.

Figure 35 summarizes the ADATA exit processing.

Figure 35. ADATA Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Operation successful.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
5=PROCESS	0	Operation successful.

Note: The ADATA exit is not called for WRITE request type.

TERM Exit Processing

You can use the TERM exit to override the effect of the TERM assembler option. The exit does this by indicating to the assembler that it opens the terminal data set and does all terminal output processing. Then, as each terminal record is passed to the exit, the exit can decide whether to write the record, and where to write the record. For instance, the exit can write the terminal records to a different data set to which the assembler would normally write them.

The TERMINAL exit is not called if you specify the NOTERM assembler option. If you wish to process the terminal records in the exit but you do not want the assembler to write the records to the normal output data set, you can do one of the following:

- Instruct the assembler to discard the terminal records by setting the exit return code
- Suppress the terminal output as follows:

  MVS Provide a `//SYSTERM DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSTERM DUMMY` command.

The assembler calls the TERMINAL exit with the following request types:

OPEN

The assembler calls the exit with a request type of 1 (OPEN) at the start of the assembly.

The exit may set the return code in the Exit parameter list to one of the following:

0 Instructs the assembler to open the terminal data set, and supply the terminal output records to the exit in later PROCESS calls.

The exit can set the record length for the terminal data set by setting the reason code to 4 and the buffer length field. The buffer length field can be set to any value from 1 and 255. If the value is zero or greater than 255, the assembler issues message `ASMA404` and does not call the exit for any further processing.

4 Indicates that the exit writes the terminal records in later WRITE calls. If you wish to provide the assembler with the values for the system variables &SYSTERM_DSN, &SYSTERM_MEMBER and &SYSTERM_VOLUME, the exit must set the reason code to 4 and place the values in the data set name, member name, and volume serial fields of the exit-specific information block. The assembler also shows this information in the *Diagnostic Cross Reference and Assembler Summary* section of the listing, and includes it in the associated data file Output File Information record.

If you provide a character string in the *str6* suboption of the EXIT assembler option, the buffer pointer field points to the character string, and buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

CLOSE

The assembler calls the exit with a request type of 2 (CLOSE) at the end of the assembly. The exit should close any data sets it opened and release any storage that it acquired.

WRITE

The assembler calls the exit with a request type of 4 (WRITE) when the exit is writing the terminal records. The buffer pointer field points to the terminal record, and the buffer length contains the length of the record.

The user exit should maintain the absolute record number and the relative record number. These fields are set to zero before the OPEN request.

PROCESS

The assembler calls the exit with a request type of 5 (PROCESS) when the assembler is writing the terminal records, and it has a record for the exit to process. The address of the record is in the buffer pointer field, and the length is in the buffer length field.

The exit can set the return code in the Exit Parameter list to one of the following:

0 Indicates that the record has been accepted, and the assembler is to write it to the terminal data set. The exit may modify the record before it returns control to the assembler. The user exit may also insert extra records in the terminal by setting the reason code to 4. The assembler writes the current record and then calls the user exit with an empty buffer. The exit must place the additional terminal record in the area pointed to by the buffer pointer field. The exit can continue to supply additional records, by setting the reason code to 4. The exit must keep track of when the assembler calls it with an empty buffer, and ensure that it resets the reason code to zero to resume normal processing.

4 Instructs the assembler to discard the terminal record.

Although the user exit can insert or discard records, the assembler maintains the absolute record number and relative record number.

Figure 36 summarizes the TERM exit processing.

Figure 36. TERM Exit Processing Summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens terminal data set. If reason code=4, exit supplies listing line length.
	4	Exit writes terminal records. If reason code=4, exit supplies system variable symbols.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record.
5=PROCESS	0	Record accepted. Exit can modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record.
	4	Requests assembler to discard record.

Sample User Exits

Three sample exits are provided with High Level Assembler. They are described under:

- Appendix I, "Sample ADATA User Exit" on page 315
- Appendix J, "Sample LISTING User Exit" on page 322
- Appendix K, "Sample SOURCE User Exit" on page 324

User Exit Coding Example

Figure 37 on page 98 shows how to code a user exit. The exit is called `MYEXIT`. It uses all user exit types and all request types. It uses the field AXPUSER to anchor the storage it has acquired to make it reenterable. If the user exit does not need to be reenterable, this code is not required.

This user exit is not supplied with High Level Assembler.

The user exit does not show examples of how to open, read, write or close a data set when it is responsible for opening the data set. Instead, it provides source records from it's own storage, and writes output records to the operator using the WTO macro.

The user exit can be invoked as the following exit types.

SOURCE Exit - INEXIT: If you specify EXIT(INEXIT(MYEXIT)), the exit allows the assembler to open the input data set. The exit issues a WTO for each record read from the input data set.

If you specify EXIT(INEXIT(MYEXIT(EXIT))), the exit opens the input data set. It passes the following records to the assembler:

```
SMALL    TITLE 'Test the assembler exits'
         MACRO
         LITTLE
         BSM   0,14  Return
         MEND
         START
         OUTER
         LITTLE
         REPRO
This is to be written to the punch data set
         COPY  TINY
         END
```

LIBRARY Exit - LIBEXIT: If you specify EXIT(LIBEXIT(MYEXIT)), the exit allows the assembler to open the library data set. The exit issues a WTO for each record read from the library data set.

If you specify EXIT(LIBEXIT(MYEXIT(EXIT))), the exit opens the library data set. It passes the records for the following macro and COPY members to the assembler:

- Macro OUTER
- Macro INNER
- COPY member TINY
- COPY member TINY1

If you specify EXIT(LIBEXIT(MYEXIT(BOTH))), the exit and the assembler opens the library data sets. The exit passes the records for the following macro and COPY members to the assembler:

- Macro OUTER
- Macro INNER
- COPY member TINY
- COPY member TINY1

LISTING Exit - PRTEXIT: If you specify EXIT(PRTEXIT(MYEXIT)), the exit allows the assembler to open the listing data set. The exit issues a WTO for the first 80 characters of each listing record.

If you specify EXIT(PRTEXIT(MYEXIT(EXIT))), the exit opens the listing data set. The exit issues a WTO for the first 80 characters of each listing record passed to the exit.

OBJECT and PUNCH Exit - OBJEXIT: If you specify EXIT(OBJEXIT(MYEXIT)), the exit allows the assembler to open the object and punch data sets. The exit issues a WTO for each object record written to the object and punch data set.

If you specify EXIT(OBJEXIT(MYEXIT(EXIT))), the exit opens the object and punch data set. The exit issues a WTO for each object record passed to the exit.

ADATA Exit - ADEXIT: If you specify EXIT(ADEXIT(MYEXIT)), the exit issues a WTO for the first 80 characters of each record written to the associated data file.

TERM Exit - TRMEXIT: If you specify EXIT(TRMEXIT(MYEXIT)), the exit allows the assembler to open the terminal data set. The exit issues a WTO for the first 68 characters of each terminal record.

If you specify EXIT(TRMEXIT(MYEXIT(EXIT))), the exit opens the terminal data set. The exit issues a WTO for the first 68 characters of each terminal record passed to the exit.

```
MYEXIT   TITLE '- EXAMPLE OF A USER EXIT'
***********************************************************************
*                                                                     *
* This sample user exit demonstrates how to code a user exit.         *
* It has code to demonstrate the use of SOURCE, LIBRARY, LISTING,     *
* PUNCH, OBJECT, ADATA and TERM exits.                                *
*                                                                     *
* This user exit uses the field AXPUSER to anchor the storage it has  *
* acquired to make it reenterable. If the user exit does not need to  *
* be reenterable, this code is not required.                          *
*                                                                     *
* REGISTER USAGE:                                                     *
*    R0  - WORK                                                       *
*    R1  - WORK                                                       *
*    R2  - WORK                                                       *
*    R3  - WORK                                                       *
*    R4  - WORK                                                       *
*    R5  - POINTER TO DCB (MVS/CMS) ONLY                              *
*    R6  - POINTER TO SOURCE INFORMATION                              *
*    R7  - POINTER TO ERROR BUFFER                                    *
*    R8  - POINTER TO BUFFER                                          *
*    R9  - POINTER TO REQUEST INFORMATION                             *
*    R10 - POINTER TO ORIGINAL PASSED PARAMETER                       *
*    R11 - NOT USED.                                                  *
*    R12 - PROGRAM SECTION BASE REGISTER                              *
*    R13 - SAVEAREA AND DYNAMIC STORAGE AREA                          *
*    R14 - RETURN ADDRESS OF CALLING MODULE                           *
*    R15 - ENTRY POINT OF CALLED MODULE                               *
*                                                                     *
***********************************************************************
         PRINT NOGEN
         EJECT
```

Figure 37 (Part 1 of 18). Example of a User Exit

```
*********************************************************************** 00033000
* MYEXIT   Entry                                                      * 00034000
* - Save the registers.                                               * 00035000
* - Acquire the dynamic storage on the first entry and save the       * 00036000
*   address in AXPUSER.                                               * 00037000
* - Chain the save areas using the forward and backward pointers.     * 00038000
* - Address the data areas passed.                                    * 00039000
* - Process the required exit according to the 'Exit type' passed.    * 00040000
*********************************************************************** 00041000
MYEXIT   CSECT                                                          00042000
         STM   R14,R12,12(R13)          Save registers                  00043000
         LR    R12,R15                  Set up first base register      00044000
         USING MYEXIT,R12,R11                                           00045000
         LA    R11,2048(,R12)                                           00046000
         LA    R11,2048(,R11)           Set up second base register     00047000
         LR    PARMREG,R1               Save parameter list address     00048000
         USING AXPXITP,PARMREG                                          00049000
         L     REQREG,AXPRIP            Get address of exit parm list   00050000
         USING AXPRIL,REQREG                                            00051000
         L     R1,AXPUSER               Get address of user area        00052000
         LTR   R1,R1                    Is address present?             00053000
         BNZ   CHAIN                    Yes, use area                   00054000
         LA    0,WORKLEN                Otherwise, get length           00055000
         GETMAIN R,LV=(0)               and getmain storage             00056000
         ST    R1,AXPUSER               Save it for later               00057000
         XC    0(WORKLEN,R1),0(R1)      Clear area                      00058000
CHAIN    DS    0H                                                       00059000
         ST    R13,4(R1)                Save previous pointer           00060000
         ST    R1,8(R13)                Save next pointer               00061000
         LR    R13,R1                   Set savearea/workarea address   00062000
         USING WORKAREA,R13                                             00063000
         SPACE 1                                                        00064000
         L     BUFREG,AXPBUFP           Get address of buffer           00065000
         USING BUFF,BUFREG                                              00066000
         L     ERRREG,AXPERRP           Get address of error buffer     00067000
         USING ERRBUFF,ERRREG                                           00068000
         L     SRCREG,AXPSIP            Get address of source info      00069000
         USING AXPSIL,SRCREG                                            00070000
         L     DCBREG,AXPDCBP           Get address of DCB              00071000
         USING IHADCB,DCBREG                                            00072000
         SPACE 1                                                        00073000
         XC    AXPRETC,AXPRETC          Zero the return code            00074000
         L     R15,AXPTYPE              Load the exit type value (1-7)  00075000
         BCTR  R15,0                    Decrement by 1                  00076000
         SLL   R15,1                    Multiply by 2                   00077000
         LH    R15,EXITADDR(R15)        Index into address list         00078000
         AR    R15,R12                  Calculate the address           00079000
         BR    R15                      Branch to applicable routine    00080000
         SPACE 1                                                        00081000
EXITADDR DC    Y(SOURCE-MYEXIT)                                         00082000
         DC    Y(LIBRARY-MYEXIT)                                        00083000
         DC    Y(LISTING-MYEXIT)                                        00084000
         DC    Y(PUNCH-MYEXIT)                                          00085000
         DC    Y(OBJECT-MYEXIT)                                         00086000
         DC    Y(ADATA-MYEXIT)                                          00087000
         DC    Y(TERM-MYEXIT)                                           00087500
         DC    Y(*-*)                                                   00088000
         EJECT                                                          00089000
```

Figure 37 (Part 2 of 18). Example of a User Exit

```
*********************************************************************** 00090000
* MYEXIT   Exit                                                       * 00091000
* - Restore the callers register 13                                   * 00092000
* - Restore the registers and set the register 15 to zero.            * 00093000
* - Return to the caller.                                             * 00094000
*********************************************************************** 00095000
EXIT     DS    0H                                                       00096000
         MVC   LASTOP,AXPRTYP+3       Save last operation code          00097000
         L     R13,4(,R13)            Unchain save areas                00098000
EXIT2    DS    0H                                                       00099000
         LM    R14,R12,12(R13)        Restore callers registers         00100000
         LA    R15,0                  Set the return code               00101000
         BSM   R0,R14                 Return to caller                  00102000
         SPACE 1                                                        00103000
*********************************************************************** 00104000
* MYEXIT   - Free storage                                             * 00105000
* - Called on a CLOSE request.                                        * 00106000
* - Free the storage acquired and zero AXPUSER.                       * 00107000
* - Go to EXIT (after R13 is restored)                                * 00108000
*********************************************************************** 00109000
FREESTOR DS    0H                                                       00110000
         XC    AXPUSER,AXPUSER        Zero User field                   00111000
         LA    0,WORKLEN              Length of area to free            00112000
         LR    R1,R13                 Address of area to free           00113000
         L     R13,4(,R13)            Restore callers register 13       00114000
         FREEMAIN R,A=(1),LV=(0)      Free the storage acquired         00115000
         B     EXIT2                                                    00116000
         SPACE 1                                                        00117000
*********************************************************************** 00118000
* MYEXIT   - Logic error                                              * 00119000
* - If an error occurred, set up the error message in the buffer      * 00120000
*   and length in AXPERRL.  Set the severity code.                    * 00121000
* - Set the return code to 20.                                        * 00122000
* - Return to the caller.                                             * 00123000
*********************************************************************** 00124000
LOGICERR DS    0H                                                       00125000
         MVC   AXPRETC,=A(AXPCBAD)    Severe error occurred             00126000
         MVC   ERRBUFF(ERRMSGL),ERRMSG Set up error message             00127000
         MVC   AXPERRL,=A(ERRMSGL)    Set up error message length       00128000
         MVC   AXPSEVC,=A(20)         Set up error message severity     00129000
         B     EXIT                                                     00130000
         EJECT                                                          00131000
```

Figure 37 (Part 3 of 18). Example of a User Exit

```
*********************************************************************** 00132000
* SOURCE EXIT                                                         * 00133000
* - Process required request type                                     * 00134000
*********************************************************************** 00135000
SOURCE   DS    0H                                                       00136000
         L     R15,AXPRTYP            Get the request type value (1-5) 00137000
         BCTR  R15,0                  Decrement by 1                    00138000
         SLL   R15,1                  Multiply by 2                     00139000
         LH    R15,SOURCE_ADDR(R15)   Index into Address list           00140000
         AR    R15,R12                Calculate the address             00141000
         BR    R15                    Branch to applicable routine      00142000
SOURCE_ADDR DC Y(SOURCE_OPEN-MYEXIT)                                    00143000
         DC    Y(SOURCE_CLOSE-MYEXIT)                                   00144000
         DC    Y(SOURCE_READ-MYEXIT)                                    00145000
         DC    Y(SOURCE_WRITE-MYEXIT)                                   00146000
         DC    Y(SOURCE_PROCESS-MYEXIT)                                 00147000
         DC    Y(*-*)                                                   00148000
         SPACE 1                                                        00149000
*********************************************************************** 00150000
* SOURCE EXIT - Process OPEN request                                  * 00151000
* - Pick up character string if it is supplied.                       * 00152000
* - Set return code indicating whether the assembler or user exit     * 00153000
*   will open the primary input data set.                             * 00154000
* - Open data set if required.                                        * 00155000
*********************************************************************** 00156000
SOURCE_OPEN     DS    0H                                                00157000
         MVI   OPENPARM,C' '          Clear open parm                   00158000
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM                        00159000
         L     R1,AXPBUFL             Get the Buffer length             00160000
         LTR   R1,R1                  Is string length zero?            00161000
         BZ    SOURCE_NOSTR           Yes, no string passed             00162000
         BCTR  R1,0                   Decrement for execute             00163000
         EX    R1,MOVESTR             Move character string             00164000
SOURCE_NOSTR    DS    0H                                                00165000
         CLC   OPENPARM(8),=CL8'EXIT'  Will user exit read input?       00166000
         BE    SOURCE_OPEN_EXIT       Yes                               00167000
         MVC   AXPRETC,=A(0)          assembler to read primary input  00168000
         B     EXIT                   Return                            00169000
SOURCE_OPEN_EXIT DS   0H                                                00170000
         MVC   AXPRETC,=A(AXPCOPN)    User exit to read primary input  00171000
         LA    R1,SRC1                Address first source record       00172000
         ST    R1,CURR_PTR            Set up pointer                    00173000
         B     EXIT                   Return                            00174000
         SPACE 1                                                        00175000
*********************************************************************** 00176000
* SOURCE EXIT - Process CLOSE request                                 * 00177000
* - Close data set if required.                                       * 00178000
* - Free storage and return.                                          * 00179000
*********************************************************************** 00180000
SOURCE_CLOSE    DS    0H                                                00181000
         B     FREESTOR                                                 00182000
         SPACE 1                                                        00183000
```

Figure 37 (Part 4 of 18). Example of a User Exit

```
*********************************************************************
* SOURCE EXIT - Process READ request                                *
* - Provide source information on first read.                       *
* - Read primary input record and place in buffer.                  *
* - Set return code to 16 at end of file.                           *
*********************************************************************
SOURCE_READ     DS    0H
         CLI   LASTOP,AXPROPN          Was last operation OPEN?
         BNE   SOURCE_READ2
         MVC   AXPMEMN,=CL255'Member'
         MVC   AXPMEMT,=CL255'None'
         MVC   AXPDSN,=CL255'INPUT.data set.NAME'
         MVC   AXPVOL,=CL255'VOL001'
         MVC   AXPREAC,=A(AXPEISA)     Indicate source info available
         XC    AXPRELREC,AXPRELREC     Set Relative Record No. to 0
         XC    AXPABSREC,AXPABSREC     Set Absolute Record No. to 0
SOURCE_READ2    DS    0H
         L     R1,CURR_PTR             Get record address
         CLI   0(R1),X'FF'             Is it EOF?
         BE    SOURCE_EOF              Yes, set return code
         MVC   0(80,BUFREG),0(R1)
         LA    R1,80(,R1)
         ST    R1,CURR_PTR             Point to next source record
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for source record
         L     R1,AXPRELREC            Update
         LA    R1,1(R1)                 Relative Record
         ST    R1,AXPRELREC              Number
         L     R1,AXPABSREC            Update
         LA    R1,1(R1)                 Absolute Record
         ST    R1,AXPABSREC              Number
         B     EXIT
SOURCE_EOF      DS    0H
         MVC   AXPRETC,=A(AXPCEOD)     End of file on input
         B     EXIT
         SPACE 1
*********************************************************************
* SOURCE EXIT - Process WRITE request                               *
* - Not valid for SOURCE exit.                                      *
* - Set return code to 20 and set up error message.                 *
*********************************************************************
SOURCE_WRITE    DS    0H
         B     LOGICERR
         SPACE 1
*********************************************************************
* SOURCE EXIT - Process PROCESS request                             *
* - Exit may modify the record, have the assembler discard the      *
*   record or insert additional records by setting the return code  *
*   and/or reason code.                                             *
*********************************************************************
SOURCE_PROCESS  DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for source record
         B     EXIT
         EJECT
```

Figure 37 (Part 5 of 18). Example of a User Exit

```
*********************************************************************** 00231000
* LIBRARY EXIT                                                        * 00232000
* - Process required request type                                     * 00233000
*********************************************************************** 00234000
LIBRARY  DS    0H                                                       00235000
         L     R15,AXPRTYP            Get the request type value (1-8) 00236000
         BCTR  R15,0                  Decrement by 1                   00237000
         SLL   R15,1                  Multiply by 2                    00238000
         LH    R15,LIBRARY_ADDR(R15)  Index into Address list          00239000
         AR    R15,R12                Calculate the address            00240000
         BR    R15                    Branch to applicable routine     00241000
LIBRARY_ADDR DC Y(LIBRARY_OPEN-MYEXIT)                                  00242000
         DC    Y(LIBRARY_CLOSE-MYEXIT)                                  00243000
         DC    Y(LIBRARY_READ-MYEXIT)                                   00244000
         DC    Y(LIBRARY_WRITE-MYEXIT)                                  00245000
         DC    Y(LIBRARY_PR_MAC-MYEXIT)                                 00246000
         DC    Y(LIBRARY_PR_CPY-MYEXIT)                                 00247000
         DC    Y(LIBRARY_FIND_MAC-MYEXIT)                               00248000
         DC    Y(LIBRARY_FIND_CPY-MYEXIT)                               00249000
         DC    Y(*-*)                                                   00250000
         SPACE 1                                                        00251000
*********************************************************************** 00252000
* LIBRARY EXIT - Process OPEN request                                 * 00253000
* - Pick up character string if it is supplied.                       * 00254000
* - Set return code indicating whether the assembler, user exit or    * 00255000
*   both will process the library.                                    * 00255000
* - Open data set if required.                                        * 00257000
*********************************************************************** 00258000
LIBRARY_OPEN    DS    0H                                                00259000
         MVI   OPENPARM,C' '          Clear open parm                  00260000
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM                        00261000
         L     R1,AXPBUFL             Get the Buffer length            00262000
         LTR   R1,R1                  Is string length zero?           00263000
         BZ    LIBRARY_NOSTR          Yes, no string passed            00264000
         BCTR  R1,0                   Decrement for execute            00265000
         EX    R1,MOVESTR             Move character string            00266000
LIBRARY_NOSTR   DS    0H                                                00267000
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process library  00268000
         BE    LIBRARY_OPEN_EXIT      Yes                              00269000
         CLC   OPENPARM(4),=CL8'BOTH'  Will Both process library       00270000
         BE    LIBRARY_OPEN_BOTH      Yes                              00271000
         MVC   AXPRETC,=A(0)          assembler to process library     00272000
         B     EXIT                   Return                           00273000
LIBRARY_OPEN_EXIT DS   0H                                               00274000
         MVC   AXPRETC,=A(AXPCOPN)    User exit to process library     00275000
         B     EXIT                   Return                           00276000
LIBRARY_OPEN_BOTH DS   0H                                               00277000
         MVC   AXPRETC,=A(AXPCOPL)    Both to process library          00278000
         B     EXIT                   Return                           00279000
         SPACE 1                                                        00280000
*********************************************************************** 00281000
* LIBRARY EXIT - Process CLOSE request                                * 00282000
* - Close data set if required.                                       * 00283000
* - Free storage and return.                                          * 00284000
*********************************************************************** 00285000
LIBRARY_CLOSE    DS    0H                                               00286000
         B     FREESTOR                                                 00287000
         SPACE 1                                                        00288000
```

Figure 37 (Part 6 of 18). Example of a User Exit

```
*********************************************************************** 00289000
* LIBRARY EXIT - Process READ request                                 * 00290000
* - Read copy/macro source and place in buffer.                       * 00291000
* - Set return code to 16 at end of member.                           * 00292000
*********************************************************************** 00293000
LIBRARY_READ     DS    0H                                               00294000
         L     R1,CURR_PTR             Get record address               00295000
         CLI   0(R1),X'FF'             Is it EOF?                       00296000
         BE    LIBRARY_EOF             Yes, set return code             00297000
         MVC   0(80,BUFREG),0(R1)                                       00298000
         LA    R1,80(,R1)                                               00299000
         ST    R1,CURR_PTR             Point to next library record     00300000
         MVC   WTOL+4(80),0(BUFREG)                                     00301000
         WTO   MF=(E,WTOL)             Issue WTO for library record     00302000
         L     R1,AXPRELREC            Update                           00302100
         LA    R1,1(R1)                 Relative Record                 00302200
         ST    R1,AXPRELREC              Number                         00302300
         L     R1,AXPABSREC            Update                           00302400
         LA    R1,1(R1)                 Absolute Record                 00302500
         ST    R1,AXPABSREC              Number                         00302600
         B     EXIT                                                     00303000
LIBRARY_EOF      DS    0H                                               00304000
         XC    CURR_PTR,CURR_PTR       Zero pointer at EOD              00305000
         MVC   AXPRETC,=A(AXPCEOD)     End of file on input             00306000
         B     EXIT                                                     00307000
         SPACE 1                                                        00308000
*********************************************************************** 00309000
* LIBRARY EXIT - Process WRITE request                                * 00310000
* - Not valid for LIBRARY exit.                                       * 00311000
* - Set return code to 20 and set up error message.                   * 00312000
*********************************************************************** 00313000
LIBRARY_WRITE    DS    0H                                               00314000
         B     LOGICERR                                                 00315000
         SPACE 1                                                        00316000
*********************************************************************** 00317000
* LIBRARY EXIT - Process PROCESS MACRO/COPY request                   * 00318000
* - Exit may modify the record, have the assembler discard the        * 00319000
*   record or insert additional records by setting the return code    * 00320000
*   and/or reason code.                                               * 00321000
*********************************************************************** 00322000
LIBRARY_PR_MAC  DS    0H                                                00323000
LIBRARY_PR_CPY  DS    0H                                                00324000
         MVC   WTOL+4(80),0(BUFREG)                                     00325000
         WTO   MF=(E,WTOL)             Issue WTO for library record     00326000
         B     EXIT                                                     00327000
         SPACE 1                                                        00328000
```

Figure 37 (Part 7 of 18). Example of a User Exit

```
********************************************************************** 00329000
* LIBRARY EXIT - Process FIND MACRO/COPY request                     * 00330000
* - Search for the member.  Set the return code to indicate          * 00331000
*   whether the member was found.                                    * 00332000
* - If the member is found, the source information is returned.      * 00333000
********************************************************************** 00334000
LIBRARY_FIND_MAC  DS   0H                                              00335000
LIBRARY_FIND_CPY  DS   0H                                              00336000
         CLC   AXPOPTS,=A(AXPORES)     Is it a resume request?         00337000
         BE    LIBRARY_RESUME          Yes, resume member              00338000
         LA    R1,MACA1                                                00339000
         CLC   AXPMEMN(8),=CL8'OUTER'                                  00340000
         BE    LIBRARY_FOUND                                           00341000
         LA    R1,MACB1                                                00342000
         CLC   AXPMEMN(8),=CL8'INNER'                                  00343000
         BE    LIBRARY_FOUND                                           00344000
         LA    R1,CPYA1                                                00345000
         CLC   AXPMEMN(8),=CL8'TINY'                                   00346000
         BE    LIBRARY_FOUND                                           00347000
         LA    R1,CPYB1                                                00348000
         CLC   AXPMEMN(8),=CL8'TINY1'                                  00349000
         BE    LIBRARY_FOUND                                           00350000
         MVC   AXPRETC,=A(AXPCMNF)     Indicate member not found       00351000
         B     EXIT                                                    00352000
LIBRARY_FOUND     DS    0H                                             00353000
         CLC   AXPOPTS,=A(AXPONEST)    Is it a nested COPY/MACRO?      00354000
         BNE   LIBRARY_DSN             No, return DSN info             00355000
         L     R2,NEST_LVL             Get nesting level               00356000
         L     R0,CURR_PTR             Get current record pointer      00357000
         ST    R0,STACKPTR(R2)         Save ptr in nest stack          00358000
         L     R0,AXPRELREC            Get Relative Record No          00358100
         ST    R0,RELREC(R2)           Save in nest stack              00358200
         LA    R2,4(,R2)               Increment it                    00359000
         ST    R2,NEST_LVL             Save nesting level              00360000
LIBRARY_DSN       DS    0H                                             00361000
         ST    R1,CURR_PTR             Save current record pointer     00362000
         MVC   AXPMEMT,=CL255'None'                                    00363000
         MVC   AXPDSN,=CL255'LIBRARY.data set.NAME'                    00364000
         MVC   AXPVOL,=CL255'VOL002'                                   00365000
         MVC   AXPREAC,=A(AXPEISA)     Indicate source info available  00366000
         XC    AXPRELREC,AXPRELREC     Set Relative Record No. to 0    00366500
         B     EXIT                                                    00367000
LIBRARY_RESUME    DS    0H                                             00368000
         L     R2,NEST_LVL             Get nesting level               00369000
         S     R2,=F'4'                Decrement stack pointer         00370000
         ST    R2,NEST_LVL             Save updated nesting level      00371000
         L     R0,STACKPTR(R2)         Get saved record pointer        00372000
         ST    R0,CURR_PTR             Restore old record pointer      00373000
         L     R0,RELREC(R2)           Get saved Relative Record No    00373100
         ST    R0,AXPRELREC            Restore Relative Record No      00373200
         B     EXIT                                                    00374000
         EJECT                                                         00375000
```

Figure 37 (Part 8 of 18). Example of a User Exit

```
*********************************************************************
* LISTING EXIT                                                      *
* - Process required request type                                   *
*********************************************************************
LISTING  DS    0H
         L     R15,AXPRTYP             Get the request type value (1-5)
         BCTR  R15,0                   Decrement by 1
         SLL   R15,1                   Multiply by 2
         LH    R15,LISTING_ADDR(R15)   Index into Address list
         AR    R15,R12                 Calculate the address
         BR    R15                     Branch to applicable routine
LISTING_ADDR DC Y(LISTING_OPEN-MYEXIT)
         DC    Y(LISTING_CLOSE-MYEXIT)
         DC    Y(LISTING_READ-MYEXIT)
         DC    Y(LISTING_WRITE-MYEXIT)
         DC    Y(LISTING_PROCESS-MYEXIT)
         DC    Y(*-*)
         SPACE 1
*********************************************************************
* LISTING EXIT - Process OPEN request                               *
* - Pick up character string if it is supplied.                     *
* - Set return code indicating whether the assembler or the user exit *
*   will write the listing.                                         *
* - Open data set if required.                                      *
*********************************************************************
LISTING_OPEN    DS    0H
         MVI   OPENPARM,C' '           Clear open parm
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM
         L     R1,AXPBUFL              Get the Buffer length
         LTR   R1,R1                   Is string length zero?
         BZ    LISTING_NOSTR           Yes, no string passed
         BCTR  R1,0                    Decrement for execute
         EX    R1,MOVESTR              Move character string
LISTING_NOSTR   DS    0H
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process listing
         BE    LISTING_OPEN_EXIT       Yes
         MVC   AXPRETC,=A(0)           assembler to write listing
         B     EXIT                    Return
LISTING_OPEN_EXIT DS   0H
         MVC   AXPRETC,=A(AXPCOPN)     User exit to write listing
         MVC   AXPMEMN,=CL255' '
         MVC   AXPMEMT,=CL255' '
         MVC   AXPDSN,=CL255'LISTING.data set.NAME'
         MVC   AXPVOL,=CL255'VOL001'
         MVC   AXPREAC,=A(AXPEISA)     Indicate data set info available
         XC    AXPRELREC,AXPRELREC     Set Relative Record No. to 0
         XC    AXPABSREC,AXPABSREC     Set Absolute Record No. to 0
         B     EXIT                    Return
         SPACE 1
*********************************************************************
* LISTING EXIT - Process CLOSE request                              *
* - Close data set if required                                      *
* - Free storage and return.                                        *
*********************************************************************
LISTING_CLOSE    DS    0H
         B     FREESTOR
         SPACE 1
*********************************************************************
* LISTING EXIT - Process READ request                               *
* - Not valid for LISTING exit.                                     *
* - Set return code to 20 and set up error message.                 *
*********************************************************************
LISTING_READ     DS    0H
         B     LOGICERR
```

Figure 37 (Part 9 of 18). Example of a User Exit

```
*********************************************************************** 00433000
* LISTING EXIT - Process WRITE request                                * 00434000
* - Write the listing record passed.                                  * 00435000
*********************************************************************** 00436000
LISTING_WRITE    DS    0H                                               00437000
         MVC   WTOL+4(80),0(BUFREG)                                     00438000
         WTO   MF=(E,WTOL)            Issue WTO for listing record      00439000
         L     R1,AXPRELREC           Update                            00439100
         LA    R1,1(R1)                Relative Record                  00439200
         ST    R1,AXPRELREC             Number                          00439300
         L     R1,AXPABSREC           Update                            00439400
         LA    R1,1(R1)                Absolute Record                  00439500
         ST    R1,AXPABSREC             Number                          00439600
         B     EXIT                                                     00440000
         SPACE 1                                                        00441000
*********************************************************************** 00442000
* LISTING EXIT - Process PROCESS request                              * 00443000
* - Exit may modify the record, have the assembler discard the        * 00444000
*   record or insert additional records by setting the return code    * 00445000
*   and/or reason code.                                               * 00446000
*********************************************************************** 00447000
LISTING_PROCESS DS    0H                                                00448000
         MVC   WTOL+4(80),0(BUFREG)                                     00449000
         WTO   MF=(E,WTOL)            Issue WTO for listing record      00450000
         B     EXIT                                                     00451000
         EJECT                                                          00452000
```

Figure 37 (Part 10 of 18). Example of a User Exit

```
*********************************************************************** 00453000
* OBJECT EXIT                                                         * 00454000
* - Process required request type                                     * 00455000
*********************************************************************** 00456000
PUNCH    DS    0H                                                       00457000
OBJECT   DS    0H                                                       00458000
         L     R15,AXPRTYP            Get the request type value (1-5) 00459000
         BCTR  R15,0                  Decrement by 1                    00460000
         SLL   R15,1                  Multiply by 2                     00461000
         LH    R15,OBJECT_ADDR(R15)   Index into Address list           00462000
         AR    R15,R12                Calculate the address             00463000
         BR    R15                    Branch to applicable routine      00464000
OBJECT_ADDR DC Y(OBJECT_OPEN-MYEXIT)                                    00465000
         DC    Y(OBJECT_CLOSE-MYEXIT)                                   00466000
         DC    Y(OBJECT_READ-MYEXIT)                                    00467000
         DC    Y(OBJECT_WRITE-MYEXIT)                                   00468000
         DC    Y(OBJECT_PROCESS-MYEXIT)                                 00469000
         DC    Y(*-*)                                                   00470000
         SPACE 1                                                        00471000
*********************************************************************** 00472000
* OBJECT EXIT - Process OPEN request                                  * 00473000
* - Pick up character string if it is supplied.                       * 00474000
* - Set return code indicating whether the assembler or the user exit * 00475000
*   will write the object/punch records.                              * 00476000
* - Open data set if required                                         * 00477000
*********************************************************************** 00478000
OBJECT_OPEN    DS    0H                                                 00479000
         MVI   OPENPARM,C' '          Clear open parm                   00480000
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM                        00481000
         L     R1,AXPBUFL             Get the Buffer length             00482000
         LTR   R1,R1                  Is string length zero?            00483000
         BZ    OBJECT_NOSTR           Yes, no string passed             00484000
         BCTR  R1,0                   Decrement for execute             00485000
         EX    R1,MOVESTR             Move character string             00486000
OBJECT_NOSTR   DS    0H                                                 00487000
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process object    00488000
         BE    OBJECT_OPEN_EXIT       Yes                               00489000
         MVC   AXPRETC,=A(0)          assembler to write object/punch  00490000
         B     EXIT                   Return                            00491000
OBJECT_OPEN_EXIT DS   0H                                                00492000
         MVC   AXPRETC,=A(AXPCOPN)    User exit to write object/punch  00493000
         MVC   AXPMEMN,=CL255'Member'                                   00493100
         MVC   AXPMEMT,=CL255' '                                        00493200
         MVC   AXPDSN,=CL255'OBJECT.data set.NAME'                      00493300
         MVC   AXPVOL,=CL255'VOL001'                                    00493400
         MVC   AXPREAC,=A(AXPEISA)    Indicate data set info available 00493500
         XC    AXPRELREC,AXPRELREC    Set Relative Record No. to 0     00493600
         XC    AXPABSREC,AXPABSREC    Set Absolute Record No. to 0     00493700
         B     EXIT                   Return                            00494000
         SPACE 1                                                        00495000
*********************************************************************** 00496000
* OBJECT EXIT - Process CLOSE request                                 * 00497000
* - Close data set if required.                                       * 00498000
* - Free storage and return.                                          * 00499000
*********************************************************************** 00500000
OBJECT_CLOSE    DS    0H                                                00501000
         B     FREESTOR                                                 00502000
         SPACE 1                                                        00503000
```

Figure 37 (Part 11 of 18). Example of a User Exit

```
*********************************************************************** 00504000
* OBJECT EXIT - Process READ request                                  * 00505000
* - Not valid for OBJECT exit.                                        * 00506000
* - Set return code to 20 and set up error message.                   * 00507000
*********************************************************************** 00508000
OBJECT_READ     DS    0H                                                00509000
         B     LOGICERR                                                 00510000
*********************************************************************** 00511000
* OBJECT EXIT - Process WRITE request                                 * 00512000
* - Write the source record passed.                                   * 00513000
*********************************************************************** 00514000
OBJECT_WRITE    DS    0H                                                00515000
         MVC   WTOL+4(80),0(BUFREG)                                     00516000
         WTO   MF=(E,WTOL)             Issue WTO for object record      00517000
         L     R1,AXPRELREC            Update                           00517100
         LA    R1,1(R1)                 Relative Record                 00517200
         ST    R1,AXPRELREC              Number                         00517300
         L     R1,AXPABSREC            Update                           00517400
         LA    R1,1(R1)                 Absolute Record                 00517500
         ST    R1,AXPABSREC              Number                         00517600
         B     EXIT                                                     00518000
         SPACE 1                                                        00519000
*********************************************************************** 00520000
* OBJECT EXIT - Process PROCESS request                               * 00521000
* - Exit may modify the record, have the assembler discard the        * 00522000
*   record or insert additional records by setting the return code    * 00523000
*   and/or reason code.                                               * 00524000
*********************************************************************** 00525000
OBJECT_PROCESS DS    0H                                                 00526000
         MVC   WTOL+4(80),0(BUFREG)                                     00527000
         WTO   MF=(E,WTOL)             Issue WTO for object record      00528000
         B     EXIT                                                     00529000
         EJECT                                                          00530000
```

Figure 37 (Part 12 of 18). Example of a User Exit

```
***********************************************************************
* ADATA EXIT                                                          *
* - Process required request type                                     *
***********************************************************************
ADATA    DS    0H
         L     R15,AXPRTYP            Get the request type value (1-5)
         BCTR  R15,0                  Decrement by 1
         SLL   R15,1                  Multiply by 2
         LH    R15,ADATA_ADDR(R15)    Index into Address list
         AR    R15,R12                Calculate the address
         BR    R15                    Branch to applicable routine
ADATA_ADDR DC  Y(ADATA_OPEN-MYEXIT)
         DC    Y(ADATA_CLOSE-MYEXIT)
         DC    Y(ADATA_READ-MYEXIT)
         DC    Y(ADATA_WRITE-MYEXIT)
         DC    Y(ADATA_PROCESS-MYEXIT)
         DC    Y(*-*)
         SPACE 1
***********************************************************************
* ADATA EXIT - Process OPEN request                                   *
* - Pick up character string if it is supplied.                       *
***********************************************************************
ADATA_OPEN    DS    0H
         MVI   OPENPARM,C' '          Clear open parm
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM
         L     R1,AXPBUFL             Get the Buffer length
         LTR   R1,R1                  Is string length zero?
         BZ    ADATA_NOSTR            Yes, no string passed
         BCTR  R1,0                   Decrement for execute
         EX    R1,MOVESTR             Move character string
ADATA_NOSTR   DS    0H
         B     EXIT                   Return
         SPACE 1
***********************************************************************
* ADATA EXIT - Process CLOSE request                                  *
* - Close data set if required.                                       *
* - Free storage and return.                                          *
***********************************************************************
ADATA_CLOSE    DS    0H
         B     FREESTOR
         SPACE 1
***********************************************************************
* ADATA EXIT - Process READ request                                   *
* - Not valid for ADATA exit.                                         *
* - Set return code to 20 and set up error message.                   *
***********************************************************************
ADATA_READ     DS    0H
         B     LOGICERR
***********************************************************************
* ADATA EXIT - Process WRITE request                                  *
* - Not valid for ADATA exit.                                         *
* - Set return code to 20 and set up error message.                   *
***********************************************************************
ADATA_WRITE    DS    0H
         B     LOGICERR
         SPACE 1
```

Figure 37 (Part 13 of 18). Example of a User Exit

```
***********************************************************************
* ADATA EXIT - Process PROCESS request                                 *
* - Exit may check the record but it may not modify the record,       *
*   discard the record or insert additional records.                  *
***********************************************************************
ADATA_PROCESS DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for ADATA record
         B     EXIT
         EJECT
```

Figure 37 (Part 14 of 18). Example of a User Exit

```
*********************************************************************
* TERM EXIT                                                         *
* - Process required request type                                   *
*********************************************************************
TERM     DS    0H
         L     R15,AXPRTYP            Get the request type value (1-5)
         BCTR  R15,0                  Decrement by 1
         SLL   R15,1                  Multiply by 2
         LH    R15,TERM_ADDR(R15)     Index into Address list
         AR    R15,R12                Calculate the address
         BR    R15                    Branch to applicable routine
TERM_ADDR DC   Y(TERM_OPEN-MYEXIT)
         DC    Y(TERM_CLOSE-MYEXIT)
         DC    Y(TERM_READ-MYEXIT)
         DC    Y(TERM_WRITE-MYEXIT)
         DC    Y(TERM_PROCESS-MYEXIT)
         DC    Y(*-*)
         SPACE 1
*********************************************************************
* TERM EXIT - Process OPEN request                                  *
* - Pick up character string if it is supplied.                     *
* - Set return code indicating whether the assembler or the user exit *
*   will write the terminal records.                                *
* - Open data set if required.                                      *
*********************************************************************
TERM_OPEN       DS    0H
         MVI   OPENPARM,C' '          Clear open parm
         MVC   OPENPARM+1(L'OPENPARM-1),OPENPARM
         L     R1,AXPBUFL             Get the Buffer length
         LTR   R1,R1                  Is string length zero?
         BZ    TERM_NOSTR             Yes, no string passed
         BCTR  R1,0                   Decrement for execute
         EX    R1,MOVESTR             Move character string
TERM_NOSTR      DS    0H
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process records?
         BE    TERM_OPEN_EXIT         Yes
         MVC   AXPRETC,=A(0)          assembler to write records
         B     EXIT                   Return
TERM_OPEN_EXIT DS      0H
         MVC   AXPRETC,=A(AXPCOPN)    User exit to write records
         MVC   AXPMEMN,=CL255' '
         MVC   AXPMEMT,=CL255' '
         MVC   AXPDSN,=CL255'TERM.data set.NAME'
         MVC   AXPVOL,=CL255'VOL001'
         MVC   AXPREAC,=A(AXPEISA)    Indicate data set info available
         XC    AXPRELREC,AXPRELREC    Set Relative Record No. to 0
         XC    AXPABSREC,AXPABSREC    Set Absolute Record No. to 0
         B     EXIT                   Return
         SPACE 1
*********************************************************************
* TERM EXIT - Process CLOSE request                                 *
* - Close data set if required.                                     *
* - Free storage and return.                                        *
*********************************************************************
TERM_CLOSE       DS    0H
         B     FREESTOR
         SPACE 1
*********************************************************************
* TERM EXIT    - Process READ request                               *
* - Not valid for TERM exit.                                        *
* - Set return code to 20 and set up error message.                 *
*********************************************************************
TERM_READ        DS    0H
         B     LOGICERR
```

Figure 37 (Part 15 of 18). Example of a User Exit

```
***********************************************************************
* TERM EXIT - Process WRITE request                                   *
* - Write the terminal record passed.                                 *
***********************************************************************
TERM_WRITE       DS    0H
         MVC   WTOL+4(68),0(BUFREG)
         WTO   MF=(E,WTOL)            Issue WTO for terminal record
         L     R1,AXPRELREC           Update
         LA    R1,1(R1)                Relative Record
         ST    R1,AXPRELREC             Number
         L     R1,AXPABSREC           Update
         LA    R1,1(R1)                Absolute Record
         ST    R1,AXPABSREC             Number
         B     EXIT
         SPACE 1
***********************************************************************
* TERM EXIT - Process PROCESS request                                 *
* - Exit may modify the record, have the assembler discard the        *
*   record or insert additional records by setting the return code    *
*   and/or reason code.                                               *
***********************************************************************
TERM_PROCESS DS      0H
         MVC   WTOL+4(68),0(BUFREG)
         WTO   MF=(E,WTOL)            Issue WTO for terminal record
         B     EXIT
```

Figure 37 (Part 16 of 18). Example of a User Exit

```
ERRMSG   DC    C'Invalid EXIT type or Request type passed to exit'      00686000
ERRMSGL  EQU   *-ERRMSG                                                 00687000
WTOL     WTO   '1234567890123456789012345678901234567890123456789012345X00688000
               67890123456789012345678901234567890',MF=L                00689000
MOVESTR  MVC   OPENPARM(*-*),0(BUFREG) Move character string            00690000
         SPACE 1                                                        00691000
SRC1     DC    CL80'SMALL    TITLE ''Test the assembler exits'''        00692000
SRC2     DC    CL80'         MACRO'                                     00693000
SRC3     DC    CL80'         LITTLE'                                    00694000
SRC4     DC    CL80'         BSM   0,14  Return'                        00695000
SRC5     DC    CL80'         MEND'                                      00696000
SRC6     DC    CL80'         START'                                     00697000
SRC7     DC    CL80'         OUTER'                                     00698000
SRC8     DC    CL80'         LITTLE'                                    00699000
SRC9     DC    CL80'         REPRO'                                     00700000
SRC10    DC    CL80'This is to be written to the punch data set'        00701000
SRC11    DC    CL80'         COPY  TINY'                                00702000
SRC12    DC    CL80'         END'                                       00714000
SRCEND   DC    X'FF'         END OF SOURCE STMTS                        00715000
         SPACE 1                                                        00716000
MACA1    DC    CL80'         MACRO'                                     00717000
MACA2    DC    CL80'         OUTER'                                     00718000
MACA3    DC    CL80'         XR    15,15'                               00719000
MACA4    DC    CL80'         INNER'                                     00720000
MACA5    DC    CL80'         LTR   15,15'                               00721000
MACA6    DC    CL80'         MEND'                                      00722000
MACAEND  DC    X'FF'         END OF MACRO STMTS                         00723000
         SPACE 1                                                        00724000
MACB1    DC    CL80'         MACRO'                                     00725000
MACB2    DC    CL80'         INNER'                                     00726000
MACB3    DC    CL80'         LR    12,15'                               00727000
MACB4    DC    CL80'         MEND'                                      00728000
MACBEND  DC    X'FF'         END OF MACRO STMTS                         00729000
         SPACE 1                                                        00730000
CPYA1    DC    CL80'TINY     DSECT               LINE 1 TINY'           00731000
CPYA2    DC    CL80'         DS    C''TINY''       LINE 2 TINY'         00732000
CPYA3    DC    CL80'         COPY  TINY1         LINE 3 TINY'           00733000
CPYA4    DC    CL80'         DS    CL10''TINY''    LINE 4 TINY'         00734000
CPYA5    DC    CL80'         DS    CL80          LINE 5 TINY'           00735000
CPYEND   DC    X'FF'         END OF COPY STMTS                          00736000
CPYB1    DC    CL80'TINY1    DSECT                 LINE 1 TINY1'        00737000
CPYB2    DC    CL80'         DS    C''TINY1''        LINE 2 TINY1'      00738000
CPYB3    DC    CL80'         DS    CL10''TINY1''     LINE 3 TINY1'      00739000
CPYBEND  DC    X'FF'         END OF COPY STMTS                          00740000
         SPACE 1                                                        00741000
```

Figure 37 (Part 17 of 18). Example of a User Exit

```
R0       EQU   0                                                     00742000
R1       EQU   1                                                     00743000
R2       EQU   2                                                     00744000
R3       EQU   3                                                     00745000
R4       EQU   4                                                     00746000
R5       EQU   5                                                     00747000
R6       EQU   6                                                     00748000
R7       EQU   7                                                     00749000
R8       EQU   8                                                     00750000
R9       EQU   9                                                     00751000
R10      EQU   10                                                    00752000
R11      EQU   11                                                    00753000
R12      EQU   12                                                    00754000
R13      EQU   13                                                    00755000
R14      EQU   14                                                    00756000
R15      EQU   15                                                    00757000
DCBREG   EQU   5                   Address of DCB                    00758000
SRCREG   EQU   6                   Address of Source Information     00759000
ERRREG   EQU   7                   Address of Error Buffer           00760000
BUFREG   EQU   8                   Address of buffer                 00761000
REQREG   EQU   9                   Address of request information    00762000
PARMREG  EQU   10                  Address or parameter              00763000
         LTORG ,                                                     00764000
         SPACE 1                                                     00765000
         DCBD  DSORG=PS,DEVD=DA                                      00766000
         SPACE 1                                                     00767000
         ASMAXITP ,                Mapping for exit parameter list   00768000
         SPACE 1                                                     00769000
BUFF     DSECT ,                                                     00770000
         DS    CL255               Record buffer                     00771000
         SPACE 1                                                     00772000
ERRBUFF  DSECT ,                                                     00773000
         DS    CL255               Error message buffer              00774000
         SPACE 1                                                     00775000
WORKAREA DSECT                                                       00776000
SAVEAREA DS    18F                 Save area                         00777000
OPENPARM DS    CL64                Character string passed at open time  00778000
LASTOP   DS    X                   Previous request type             00779000
CURR_PTR DS    A                   Current record pointer            00780000
NEST_LVL DS    F                   Current nesting level             00781000
STACKPTR DS    10A                 Stack to save record pointers     00782000
RELREC   DS    10A                 Stack to save relative record no  00783000
WORKLEN  EQU   *-WORKAREA                                            00784000
         END   MYEXIT                                                00785000
```

Figure 37 (Part 18 of 18). Example of a User Exit

Chapter 5. Providing External Functions

Two conditional assembly instructions, SETAF and SETCF, let you call routines written in a programming language that conforms to standard OS Linkage conventions. The assembler calls the external function load module and passes the address of an external function parameter list in Register 1. Each differently named external function called in the same assembly is provided with a separate parameter list.

The SETAF instruction calls an external function module, and passes to the module any number of parameters containing arithmetic values. The SET symbol in the instruction is assigned the fullword value returned by the external function module.

The SETCF instruction calls an external function module, and passes to the module any number of parameters containing character values. The SET symbol in the instruction is assigned the character string value returned by the external function module. The character string value can be up to 255 characters long.

This chapter describes the external function processing requirements, the linkage conventions for generating an external function module, and the contents of the parameter list the assembler passes to the module.

External Function Processing

The assembler calls an external function each time it processes a SETAF or SETCF instruction. The assembler loads the external function module when the first call to the module is encountered. The assembler must be able to locate the external function module as follows:

Under MVS: The external function must be a link-edited load module in a partitioned data set that is in the standard search sequence. The external function can also be located in the Link Pack Area (LPA).

Under CMS: The external function must have a file type of MODULE and be located on one of the accessed disks. To generate the module use the CMS LOAD and GENMOD commands. When the LOAD command is issued, specify the RLDSAVE option to make the module relocatable. If RLDSAVE is not specified, the assembler program or data storage might be overlaid during execution.

The assembler expects the external function module to be generated in 31-bit addressing mode (AMODE 31). Only one copy of the load module is loaded, so it must be serially reusable.

Using the SIZE Option to Reserve Storage: External function modules are loaded by the assembler during the assembly, which is after the assembler completes initialization. Therefore, you should allow enough virtual storage in the address space the assembler runs in, so the external function modules can be loaded, and for any storage your external function might acquire. You can reserve storage for your external function modules by reducing the amount of storage the assembler uses. Use the SIZE assembler option to control the amount of storage the assembler uses.

Linkage Conventions

External function modules are called by the assembler using standard OS Linkage conventions. The external function can be written in any language that:

- Uses standard OS linkage conventions
- Can be called many times using the module (or phase) entry point
- Retains storage for variables across invocations and does not require a run-time environment to be maintained across invocations

See the specific programming language *Programmer's Guide* to determine if you can use the programming language to write an external function for the High Level Assembler.

The contents of the registers upon entry to the external function are as follows:

Register 0	Undefined
Register 1	Address of external function parameter list
Register 2 through 12	Undefined
Register 13	Address of the 72 byte register save area
Register 14	Return address
Register 15	Address of entry point of external function

External Function Parameter List

The assembler passes a parameter list to the external function module. Register 1 points to the parameter list, and macro ASMAEFNP maps the parameter list. Figure 38 on page 118 shows the SETAF parameter list, and Figure 39 on page 119 shows the SETCF parameter list. A separate copy of the external function parameter list is passed to each external function. The sections following the figures describe each of the parameters in detail.

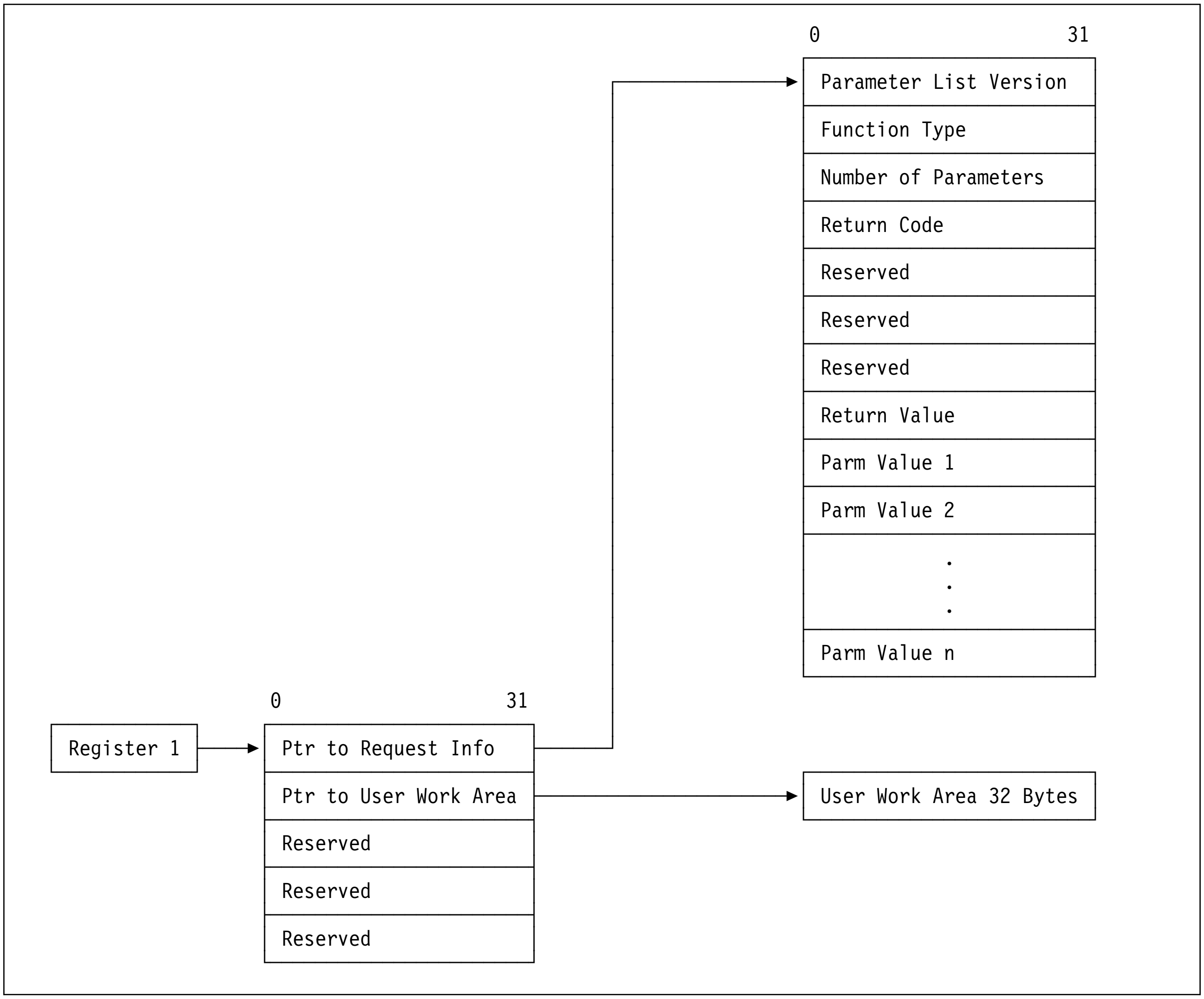

Figure 38. SETAF External Function Parameter List Format

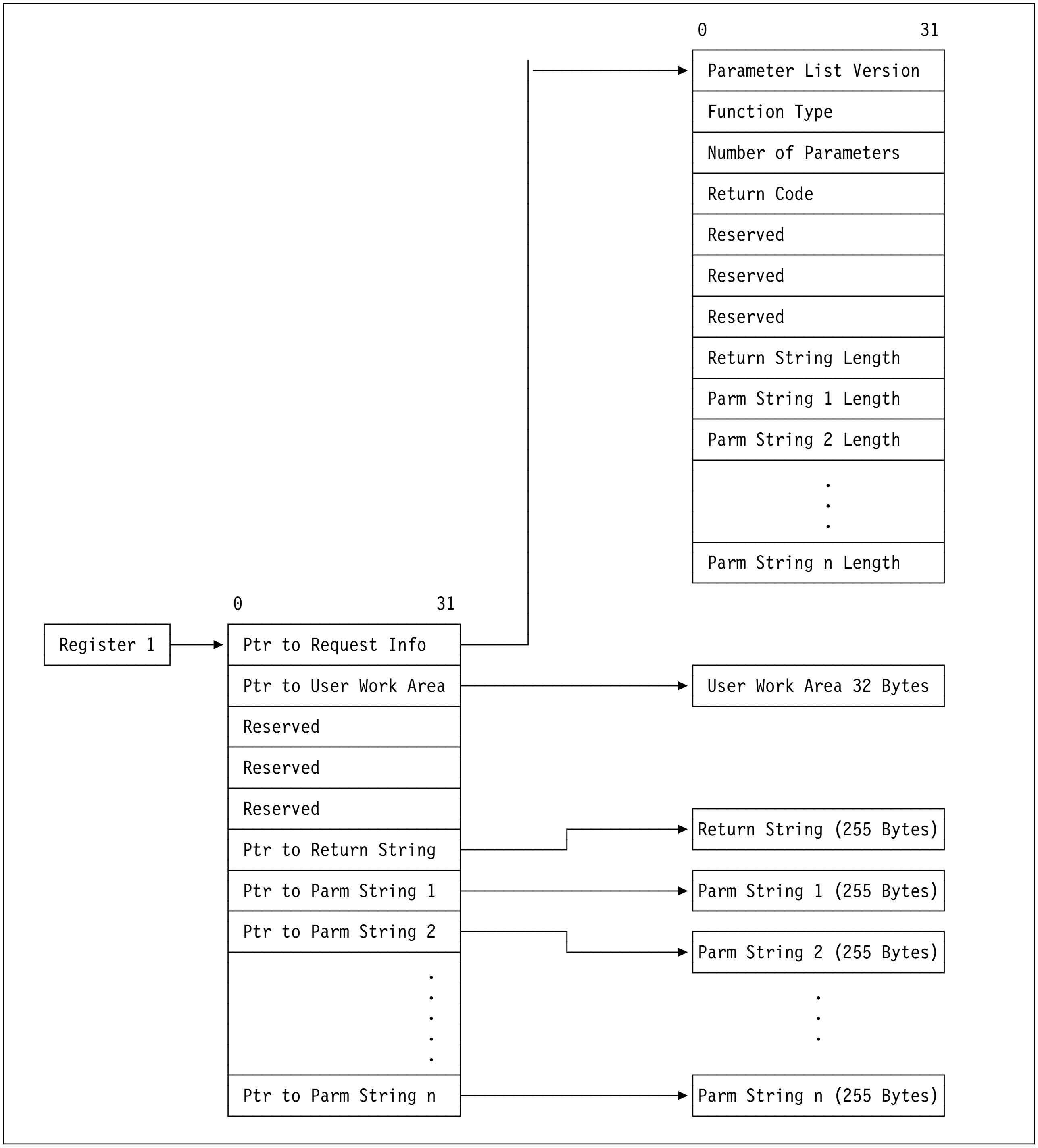

Figure 39. SETCF External Function Parameter List Format

The external function parameter list consists of the following addresses:

Request Information List

Pointer to a list of binary fullword items that describe the external function request.

The assembler sets this pointer which is always valid.

The Request Information List consists of the following fields:

Parameter List Version

A fullword identifying which version of the parameter list is provided to the external function. Only one value is allowed in this field:

1 (for Version 1)

Function Type

A fullword, set by the assembler to indicate the function type:

1 SETAF function

2 SETCF function

Number of Parameters

A fullword indicating the number of parameters provided on the call to this external function.

The assembler always sets this field.

Return Code

A fullword, set by the external function, indicating success or failure of the operation, and action to be taken by the assembler on return from the external function

Figure 40. External Function Return Codes

RC=0	RC>0
Operation successful. Value or string returned.	Operation failed. Request assembler to terminate immediately.

When the return code is greater than 0 the assembler issues diagnostic error message ASMA941.

Reserved

This storage is reserved for future use by IBM. The external function should not use this field, nor should it rely on the contents of this field, as the contents of this field are liable to change without notice.

SETAF Function

When the function type is SETAF, the following fields are provided:

Return Value
: A fullword, set by the external function. This field is set to zero by the assembler before the external function call.

Parm Value *n*
: A fullword, set by the assembler, containing the value of the parameter passed to the external function.

The Number of Parameters field indicates the number of Parm Value fields in the Request Information List.

SETCF Function

When the function type is SETCF, the following fields are provided, and contain the length of their related strings.

Return String Length
: An unsigned fullword, set by the external function, containing the length of the string pointed to by the Return String Pointer field.

  The assembler uses this field as the length of the returned string.

  If the length is greater than 255, it is reset to 255 by the assembler. The consequence of returning a string longer than 255 bytes is unpredictable.

Parm String *n* Length
: An unsigned fullword, set by the assembler, containing the length of the string pointed to by the Parm String *n* Pointer field.

  The external function should use this length to determine the length of the Parm String *n* passed by the assembler.

  The assembler sets this field to a value between 0 and 255 inclusive.

  The Number of Parameters field indicates the number of Parm String *n* Length fields in the Request Information List.

User Work Area

Address of the User Work Area.

The assembler provides four double words of storage for use by the external function. This storage is double-word aligned and the assembler initializes it to zero for the first call to the external function.

It can be used by the external function to store information (such as the address of acquired storage) between calls. The contents of this storage area is preserved across all calls until the assembly completes. The assembler does not use or modify the work area.

Reserved

This storage is reserved for future use by IBM. The external function should not use this field, nor should it rely on the contents of this field, as the contents of this field are liable to change without notice.

Return String

Address of the string returned by the external function.

The assembler always sets this pointer before invoking an external function. The external function can put up to 255 bytes of character string data into the area addressed by this field.

Parm String

Address of string number *n* passed to the external function.

The assembler always sets this pointer before invoking an external function. The length of the string pointed to by this field is contained in the Parm String *n* Length field.

The Number of Parameters field in the Request Information List indicates the number of Parm String *n* Pointers in the External Function Parameter List.

Chapter 6. Diagnosing Assembly Errors

The diagnostic facilities for High Level Assembler include:

- Diagnostic messages for assembly errors
- A macro trace and dump facility (MHELP)
- Messages and dumps issued by the assembler if it ends abnormally.
- Diagnostic or explanatory messages issued by the source program or by macro definitions (MNOTEs)

This chapter provides an overview of these facilities. The assembly error diagnostic messages and abnormal assembly termination messages are described in detail in Appendix G, "High Level Assembler Messages" on page 271.

Assembly Error Diagnostic Messages

High Level Assembler prints most error messages in the listing immediately following the statement in error. It also prints the total number of flagged statements and their statement numbers in the *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing.

The messages do not follow the statement in error when:

- Errors are detected during editing of macro definitions read from a library. A message for such an error appears after the first call in the source program to that macro definition. You can, however, bring the macro definition into the source program with a COPY statement or using the LIBMAC assembler option. The editing error messages then follow immediately after the statements in error.
- Errors are detected by the lookahead function of the assembler. (For attribute references, look-ahead processing scans for symbols defined on statements after the one being assembled.) Messages for these errors appear after the statements in which they occur. The messages may also appear at the point at which lookahead was called.
- Errors are detected on conditional assembler statements during macro generation or MHELP testing. Such a message follows the most recently generated statement or MHELP output statement.

A typical error diagnostic message is:

```
ASMA057E *** ERROR *** UNDEFINED OPERATION CODE — xxxxxxxx
```

The term ***ERROR*** is part of the message if the severity code is 8 or greater. The term **WARNING** is part of the message if the severity code is 0 to 4, inclusively.

A copy of a segment of the statement in error, represented above by *xxxxxxxx*, is appended to the end of many messages. Normally this segment begins at the bad character or term. For some errors, however, the segment begins after the bad character or term.

If a diagnostic message follows a statement generated by a macro definition, the following items might be appended to the error message:

- The number of the model statement in which the error occurred, or the first five characters of the macro name
- The SET symbol, parameter number, or value string associated with the error

Macro Parameters: References to macro parameters are by number, instead of by name. For example, the 44th parameter is shown as PARAM00044. The assembler uses numbers 0 to 40 for for the standard system parameters as follows:

```
PARAM00000 =  &SYSNDX
PARAM00001 =  &SYSECT
PARAM00002 =  &SYSLOC
PARAM00003 =  &SYSTIME
PARAM00004 =  &SYSDATE
PARAM00005 =  &SYSASM
PARAM00006 =  &SYSVER
PARAM00007 =  &SYSDATC
PARAM00008 =  &SYSJOB
PARAM00009 =  &SYSSTEP
PARAM00010 =  &SYSSTYP
PARAM00011 =  &SYSSTMT
PARAM00012 =  &SYSNEST
PARAM00013 =  &SYSSEQF
PARAM00014 =  &SYSOPT_DBCS
PARAM00015 =  &SYSOPT_OPTABLE
PARAM00016 =  &SYSOPT_RENT
PARAM00017 =  &SYSTEM_ID
PARAM00018 =  &SYSIN_DSN
PARAM00019 =  &SYSIN_MEMBER
PARAM00020 =  &SYSIN_VOLUME
PARAM00021 =  &SYSLIB_DSN
PARAM00022 =  &SYSLIB_MEMBER
PARAM00023 =  &SYSLIB_VOLUME
PARAM00024 =  &SYSPRINT_DSN
PARAM00025 =  &SYSPRINT_MEMBER
PARAM00026 =  &SYSPRINT_VOLUME
PARAM00027 =  &SYSTERM_DSN
PARAM00028 =  &SYSTERM_MEMBER
PARAM00029 =  &SYSTERM_VOLUME
PARAM00030 =  &SYSPUNCH_DSN
PARAM00031 =  &SYSPUNCH_MEMBER
PARAM00032 =  &SYSPUNCH_VOLUME
PARAM00033 =  &SYSLIN_DSN
PARAM00034 =  &SYSLIN_MEMBER
PARAM00035 =  &SYSLIN_VOLUME
PARAM00036 =  &SYSADATA_DSN
PARAM00037 =  &SYSADATA_MEMBER
PARAM00038 =  &SYSADATA_VOLUME
PARAM00039 =  &SYSPARM
PARAM00040 =  Name Field Parameter
```

After these, the keyword parameters are numbered in the order defined in the macro definition, followed by positional parameters. When there are no keyword parameters in the macro definition, PARAM00041 refers to the first positional parameter.

Conditional Assembly: If a diagnostic message follows a conditional assembly statement in the source program, the following items are appended to the error message:

- The word 'OPENC', meaning 'open code'
- The SET symbol or value string associated with the error

Multiple Messages: Several messages can be issued for a single statement or even for a single error within a statement. This happens because each statement is usually evaluated on more than one level (for example, term level, expression level, and operand level) or by more than one phase of the assembler. Each level or phase can diagnose errors; therefore, most or all of the errors in the statement are flagged. Occasionally, duplicate error messages may occur. This is a normal result of the error detection process.

Figure 41 on page 126 is an example of High Level Assembler handling of error messages, and includes message ASMA435I to show the effect of the FLAG(RECORD) assembler option.

MNOTE Statements

An MNOTE statement is included in a macro definition or in the source program. It causes the assembler to generate an inline error or informational message.

An MNOTE appears in the listing as follows:

ASMA254I ***MNOTE*** *severity code*, *message*

Unless it has a severity code of * or the severity code is omitted, the statement number of the MNOTE is listed in the diagnostic cross-reference.

```
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                          HLASM R2.0  1995/03/26 16.24
                                     1 ***********************************************************************
                                     2 *             SAMPLE ERROR DIAGNOSTIC MESSAGES                        *
                                     3 *          IN SOURCE PROGRAM (OPEN CODE) AND GENERATED BY MACRO CALLS *
                                     4 ***********************************************************************
000000                               6 A        CSECT
000000 0000 0000           00000     7          STM   14,U2,12(13(
   ASMA044E *** ERROR *** Undefined symbol - U2
   ASMA029E *** ERROR *** Incorrect register or mask specification
   ASMA179S *** ERROR *** Delimiter error, expected right parenthesis
   ASMA435I ** WARNING ** Record 7 in CJMTEST  ASSEMBLE A1 on volume: A49191
000004 05C0                          8          BALR  12,0
                 R:C  00006          9          USING *,12
000006 0000 0000           00000    10          ST    13,SAVE+4
   ASMA044E *** ERROR *** Undefined symbol - SAVE
   ASMA435I ** WARNING ** Record 10 in CJMTEST  ASSEMBLE A1 on volume: A49191
                                    11          OPEN   (CRDIN,(INPUT),CRDOUT,(OUTPUT)
   ASMA088E *** ERROR *** Unbalanced parentheses in macro call operand - OPEN /(CRDIN,(INPUT),CRDOUT,(OUTPUT)
   ASMA435I ** WARNING ** Record 11 in CJMTEST  ASSEMBLE A1 on volume: A49191
00000A 0700                         12+         CNOP  0,4                              ALIGN LIST TO FULLWORD      01-OPEN
00000C 4110 C00E           00014    13+         LA    1,*+8                            LOAD R1 W/LIST ADR @V6PXJRU 01-OPEN
000010 47F0 C00E           00014    14+         B     *+4                              BRANCH AROUND LIST @V6PXJRU 01-OPEN
   ASMA254I *** MNOTE ***           15+     12,***  IHB001  DCB OPERAND REQ'D-NOT SPECIFIED                    02-IHBER
                                    16          DROP  11
   ASMA045W ** WARNING ** Register or label not previously used - 11
   ASMA435I ** WARNING ** Record 12 in CJMTEST  ASSEMBLE A1 on volume: A49191
                                    18 ***********************************************************************
                                    19 *     EDITING AND GENERATION ERRORS AND MNOTES FROM A LIBRARY MACRO   *
                                    20 ***********************************************************************
                                    22          LOADR REG1=10,REG2=8,WOOSHA,SUMA
000014 58A0 C02E           00034    23+         L     10,WOOSHA                                                    01-LOADR
000018 5880 C032           00038    24+         L     8,SUMA                                                       01-LOADR
                                    25          LOADR REG1=25,REG2=8,WOOSHA,MAINY
00001C 0000 0000           00000    26+         L     25,WOOSHA                                                    01-LOADR
   ASMA029E *** ERROR *** Incorrect register or mask specification
   ASMA435I ** WARNING ** Record 5 in MYMAC    MACLIB   A1(LOADR) on volume: A49191
000020 0000 0000           00000    27+         L     8,MAINY                                                      01-LOADR
   ASMA044E *** ERROR *** Undefined symbol - MAINY
   ASMA435I ** WARNING ** Record 6 in MYMAC    MACLIB   A1(LOADR) on volume: A49191
                                    28          LOADR REG2=10,SUMA,MAINY
   ASMA254I *** MNOTE ***           29+     36,YOU LEFT OUT THE FIRST REGISTER                                     01-LOADR
```

Figure 41 (Part 1 of 2). Sample Error Diagnostic Messages

```
                                    30 ***********************************************************************
                                    31 *     SAMPLE IN-LINE MACRO DEFINITION                                 *
                                    32 ***********************************************************************
                                    34          MACRO
                                    35 &NAME    LOADR &REG1=,&REG2=,&OP1,&OP2
                                    36 &R(1)    SETA  &REG1,&REG2
                                    37          AIF   (T'&REG1 EQ 'O').ERR
                                    38          L     &R(1),&OP1
                                    39          L     &R(2),&OP2
                                    40          MEXIT
                                    41 .ERR     MNOTE 36,'YOU LEFT OUT THE FIRST REGISTER'
                                                                                                  Page    4
  Active Usings: A+X'6',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                         HLASM R2.0  1995/03/26 16.24
                                    42          MEND
                                    44 ***********************************************************************
                                    45 *     SAMPLE MACRO CALLS WITH GENERATION ERRORS AND MNOTES            *
                                    46 ***********************************************************************
                                    48          LOADR REG1=10,REG2=8,WOOSHA,SUMA
000024 58A0 C02E            00034   49+         L     10,WOOSHA                                         01-00038
000028 5880 C032            00038   50+         L     8,SUMA                                            01-00039
                                    51          LOADR REG1=25,REG2=8,WOOSHA,&MAINY
   ASMA003E *** ERROR *** Undeclared variable symbol; default=0, null, or type=U - OPENC/MAINY
   ASMA435I ** WARNING ** Record 40 in CJMTEST  ASSEMBLE A1 on volume: A49191
00002C 0000 0000            00000   52+         L     25,WOOSHA                                         01-00038
   ASMA029E *** ERROR *** Incorrect register or mask specification
   ASMA435I ** WARNING ** Record 45 in CJMTEST  ASSEMBLE A1 on volume: A49191
000030 0000 0000            00000   53+         L     8,                                                01-00039
   ASMA074E *** ERROR *** Illegal syntax in expression -
   ASMA431W ** WARNING ** Continuation statement may be in error - continuation column is blank.
   ASMA435I ** WARNING ** Record 46 in CJMTEST  ASSEMBLE A1 on volume: A49191
                                    54          LOADR REG2=8,SUMA,MAINY
   ASMA254I *** MNOTE ***           55+     36,YOU LEFT OUT THE FIRST REGISTER                          01-00041
000034                              56 WOOSHA   DS    F
000038                              57 SUMA     DS    F
                                    58          END
```

Figure 41 (Part 2 of 2). Sample Error Diagnostic Messages

Suppression of Error Messages and MNOTE Statements

Optionally, you can suppress error messages and MNOTE statements below a specified severity level by specifying the assembler option FLAG(*n*) (where *n* is the lowest severity message that the assembler issues).

Reference Information for Statements in Error

The FLAG(RECORD) assembler option instructs the assembler to issue message ASMA435I after the last error message for each statement in error. This message shows reference information, including the data set name, and member name (if applicable), and the input record number of the statement in error. When you specify this option, the assembler includes reference information with the flagged statements in the *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing. The reference information includes:

- The member name (if applicable)
- The input record number of the statement in error
- The input dataset concatenation value

Abnormal Assembly Termination

Whenever the assembly cannot complete, High Level Assembler provides a message and, in some cases, a specially formatted dump for diagnostic information. This might indicate an assembler malfunction or it might indicate a programmer error. The statement causing the error is identified and, if possible, the assembly listing up to the point of the error is printed. Appendix G, "High Level Assembler Messages" on page 271 describes the abnormal termination messages. The messages give enough information to enable you (1) to correct the error and reassemble your program, or (2) to determine that the error is an assembler malfunction.

MHELP—Macro Trace Facility

The MHELP instruction controls a set of trace and dump facilities. You select options by specifying an absolute expression in the MHELP operand field. MHELP statements can occur anywhere in open code or in macro definitions. MHELP options remain in effect until superseded by another MHELP statement.

Format of MHELP:

```
Name        Operation     Operand

            MHELP         Absolute expression, binary or decimal options
```

The operands are:

B'1' or 1. Macro Call Trace
B'10' or 2. Macro Branch Trace
B'100' or 4. Macro AIF Dump
B'1000' or 8. Macro Exit Dump
B'10000' or 16. Macro Entry Dump
B'100000' or 32. Global Suppression
B'1000000' or 64. Macro Hex Dump
B'10000000' or 128. Suppression
Other values Control on &SYSNDX

Refer to Appendix F, "MHELP Sample Macro Trace and Dump" on page 262 for complete details about this facility.

Part 2. Developing Assembler Programs under MVS

Chapter 7. Assembling your Program under MVS

This chapter describes how to invoke the assembler under MVS. It describes:

- The input to the assembler
- The output from the assembler
- How to invoke the assembler under MVS and TSO
- How to invoke the assembler dynamically from a program
- How to assemble multiple source programs using the BATCH option
- The data sets used by the assembler
- The assembler return codes
- The cataloged procedures of job control language supplied by IBM

Input to the Assembler

As input, the assembler accepts a program written in the assembler language as defined in the *Language Reference*. This program is referred to as a source module. Some statements in the source module (macro or COPY instructions) may cause additional input to be obtained from a macro library.

Input can also be obtained from user exits. See Chapter 4, "Providing User Exits" on page 66 for more information.

Output from the Assembler

The output from the assembler can consist of an object module, a program listing, terminal messages and an associated data file. The object module can be written to a data set residing on a direct access device or a magnetic tape. If you specify the XOBJECT assembler option, the assembler produces an extended object format module. Both formats of the object module are written to the same data set, however only one format can be produced at a time. From that data set, the object module can be read and processed by the linkage editor, the batch loader or the DFSMS/MVS binder. See Appendix C, "Object Deck Output" on page 210 for the format of the object module. The format of the extended object format module is described in *DFSMS/MVS Version 1 Release 3 Program Management*, SC26-4916.

The program listing shows all the statements in the module, both in source and machine language format, and gives other important information about the assembly, such as error messages and cross reference information. The listing is described in detail in Chapter 2, "Using the Assembler Listing" on page 7.

Invoking the Assembler under MVS

The JCL for running an assembly includes:

- A job description
- A statement to run the assembler
- Definitions for the data sets needed

The simplest way to assemble your program under MVS is to code JCL that uses the cataloged procedure shown in Figure 42 on page 132.

```
//jobname  JOB  accountno,progrname,MSGLEVEL=1  1
//stepname EXEC ASMAC                           2
//SYSIN    DD   *                               3
⋮
assembler source statements
⋮
/*
```

Figure 42. JCL for Assembly, using Cataloged Procedure

1 Identifies the beginning of your job to the operating system. *jobname* is the name you assign to the job. *accountno* specifies the account to which your job is charged, and *progrname* is the name of the programmer responsible for the job. `MSGLEVEL=1` specifies that the job control statements connected with this job are to be listed. Check what parameters are required at your installation and how they must be specified.

2 Calls the cataloged procedure ASMAC. As a result, a number of job control statements are included in the job from the procedure library. ASMAC is described under "Cataloged Procedure for Assembly (ASMAC)" on page 155; an expanded job stream is shown there.

3 Specifies that the assembler language source program follows immediately after this statement.

These statements cause the assembler to assemble your program, produce a listing and write an object module to the SYSLIN data set. If you do not want an object module written to the SYSLIN data set, use the following job control statements to assemble the program:

```
//jobname  JOB  accountno,progrname,MSGLEVEL=1
//stepname EXEC ASMAC,PARM=NOOBJECT
//SYSIN    DD   *
⋮
assembler source statements
⋮
/*
```

Figure 43. JCL for Assembly, using Cataloged Procedure, with NOOBJECT

Assembler Options: The second parameter (`PARM`) specifies the assembler option `NOOBJECT`, which tells the assembler not to write the object module to SYSLIN. For a full discussion of assembler options, see Chapter 3, "Controlling your Assembly with Options" on page 34.

Using your own JCL: The cataloged procedures might not comply with your data processing requirements. Figure 44 on page 133 shows sample job control statements that you can use instead to assemble your program.

```
//ASMJOB   JOB  1,MSGLEVEL=1
//ASSEMBLY EXEC PGM=ASMA90,PARM=OBJECT
//SYSUT1   DD   UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD   SYSOUT=A
//SYSTERM  DD   SYSOUT=A
//SYSLIN   DD   DSNAME=PROG.OBJ,DISP=OLD
//SYSPUNCH DD   DSNAME=PROG.DECK,DISP=OLD
//SYSADATA DD   DSNAME=PROG.ADATA,DISP=OLD
//SYSIN    DD   DSNAME=PROG.SOURCE,DISP=SHR
```

Figure 44. JCL for Assembly

Refer to "Related Publications (MVS)" on page 331 for a list of JCL manuals that describe additional techniques for specifying job control statements and overriding catalogued procedures.

Invoking the Assembler under TSO

Under TSO, you can use TSO commands, or command lists (CLIST), or ISPF to assemble your program. Figure 45 shows how to allocate the data sets and assemble the source program using the ALLOCATE and CALL commands. The commands are shown in bold text.

```
READY
ALLOCATE FILE(SYSUT1) CYLINDERS SPACE(1 1) REUSE
READY
ALLOCATE FILE(SYSPRINT) DATASET(*) REUSE
READY
ALLOCATE FILE(SYSTERM) DATASET(*) REUSE
READY
ALLOCATE FILE(SYSLIN) DATASET(PROG.OBJ) NEW TRACKS SPACE(3,3)
    BLKSIZE(80) LRECL(80) RECFM(F B) CATALOG REUSE
READY
ALLOCATE FILE(SYSADATA) DATASET(PROG.ADATA) NEW CYLINDERS
    SPACE(1 1) BLKSIZE(8192) LRECL(8188) RECFM(V B)
    REUSE CATALOG
READY
ALLOCATE FILE(SYSIN) DATASET(PROG.ASSEMBLE) SHR REUSE
READY
CALL 'SYS1.LINKLIB(ASMA90)' 'ADATA,LIST(133),OBJECT,TERM'
⋮
Assembler listing and messages
⋮
READY
FREE FILE(SYSADATA,SYSUT1,SYSPRINT,SYSTERM,SYSLIN,SYSIN)
READY
```

Figure 45. Assembling under TSO

You can enter ALLOCATE commands in any order, however, you must enter all of them before you start the assembly. Figure 46 shows the data sets you must allocate when you specify particular assembler options.

Figure 46. Assembler Options and Data Sets Required

Option Specified	Data Sets Required
Any	SYSUT1 and SYSIN
LIST	SYSPRINT
TERM	SYSTERM
OBJECT or XOBJECT	SYSLIN
DECK	SYSPUNCH
ADATA	SYSADATA

Exit Option: If you specify the EXIT option, the user exit program module must be in a partitioned data set that is in the standard search sequence, including the Link Pack Area (LPA).

Invoking the Assembler Dynamically

You can invoke High Level Assembler from a running program using the CALL, LINK, XCTL, or ATTACH system macro instructions.

When you use CALL, LINK, or ATTACH, you can supply:

- The assembler options
- The ddnames of the data sets to be used during processing

If you use XCTL, you cannot pass options to the assembler; the assembler uses the installation default options. Figure 47 shows how to invoke the assembler dynamically.

```
Name      Operation      Operand

symbol    CALL           ASMA90,(optionlist[,ddnamelist]),VL
          LINK|ATTACH    EP=ASMA90,PARAM=(optionlist[,ddnamelist]),VL=1
```

Figure 47. Invoking the Assembler Dynamically

ASMA90
is the load module name and entry point to invoke the assembler. ASMA90 may be invoked in either 24-bit or 31-bit addressing mode.

EP
specifies the symbolic name of the assembler load module and entry point.

PARAM
specifies, as a sublist, address parameters to be passed from the program to the assembler. The first word in the address parameter list contains the address of the option list. The second word contains the address of the ddname list.

optionlist
specifies the address of a variable-length list containing the options. The address of an option list must be provided even if no options are required.

The option list must begin on a halfword boundary. The first two bytes contain the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form, with each field separated from the next by a comma. No blanks should appear in the list, except within the string specified for the EXIT or SYSPARM options, as long as the string is enclosed within single quotes.

ddnamelist
specifies the address of a variable-length list containing alternative ddnames for the data sets used during assembler processing. If standard ddnames are used, this operand can be omitted.

The ddname list must begin on a halfword boundary. The first two bytes contain the number of bytes in the remainder of the list. Each name of less than 8 bytes must be left-justified and padded to 8 bytes with blanks. If an alternative ddname is omitted, the standard name is assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the 8-byte entries in the ddname list is as follows:

Entry	Alternative
1	SYSLIN
2	Not applicable
3	Not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	Not applicable
10	Not applicable
11	Not applicable
12	SYSTERM
13	Not applicable
14	Not applicable
15	Not applicable
16	SYSADATA

Overriding ddname: Any overriding ddname specified when High Level Assembler was installed, occupies the corresponding position in the above list. The overriding ddname can also be overridden during invocation. For example, if SYSWORK1 replaced SYSUT1, it occupies position 8 in the above list. SYSWORK1 can be overridden by another name during invocation.

VL specifies that the sign bit is to be set to 1 in the last word of the parameter address list. `VL` must be specified for the CALL macro and `VL=1` for the LINK or ATTACH macros.

```
DYNAMICM CSECT
DYNAMICM RMODE  24
DYNAMICM AMODE  ANY
BEGIN    SAVE   (14,12)
         USING  BEGIN,15
         ST     13,SAVEAREA+4
         LA     13,SAVEAREA
         CALL   ASMA90,(OPTIONS),VL
         L      13,SAVEAREA+4
         RETURN (14,12)
SAVEAREA DS     18F
OPTIONS  DC     Y(OPTIONSL)
OPTS     DC     C'XREF(SHORT)'
OPTIONSL EQU    *-OPTS
         END
```

Figure 48. Sample Program to Call the Assembler Dynamically

Batch Assembling

A sequence of separate assembler programs may be assembled with a single invocation of the assembler when the BATCH option is specified. The object programs produced from this assembly may be linked into either a single program module or separate program modules.

When the BATCH option is specified, each assembler program in the sequence must be terminated by an END statement, including the last program in the batch. If an END statement is omitted, the program will be assembled with the next program in the sequence. If the END statement is omitted from the last program in the sequence, an END statement will be generated by the assembler.

If you want to produce more than one program module, a NAME control statement must be written for each one. The NAME statement must be written after the object module. The following example shows how to create two program modules SECT1 and SECT2.

```
SECT1    CSECT          Start of first load module
         .
         .
         Source instructions
         .
         .
         END            End of first load module
         PUNCH ' NAME SECT1(R)'
         END
SECT2    CSECT          Start of second load module
         .
         .
         Source instructions
         .
         .
         END            End of second load module
         PUNCH ' NAME SECT2(R)'
         END
```

Input and Output Data Sets

Depending on the options in effect, High Level Assembler requires the following data sets, as shown in Figure 49:

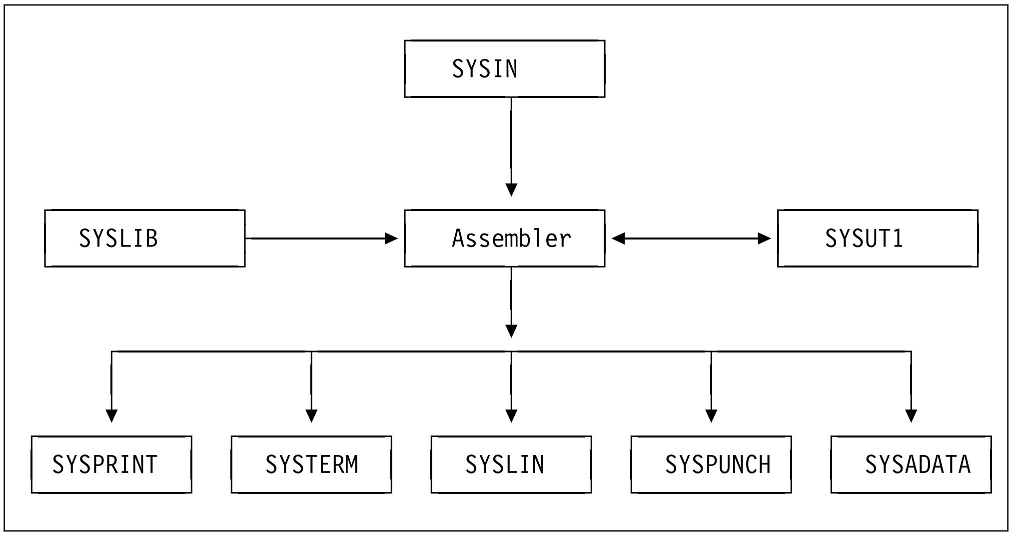

Figure 49. High Level Assembler Files

You can override the ddnames during installation or when invoking the assembler dynamically (see "Invoking the Assembler Dynamically" on page 134).

High Level Assembler requires the following data sets:

SYSUT1 — A work data set used as intermediate external storage when processing the source program. This data set is used when there is not enough main storage available to assemble in-storage.

If the value specified for the SIZE option is large enough, an in-storage assembly is done and the work data set SYSUT1 can be omitted, although a warning message is issued.

SYSIN — An input data set containing the source statements to be processed.

In addition, the following six data sets might be required:

SYSLIB — A data set containing macro definitions (for macro definitions not defined in the source program), source code to be called through COPY assembler instructions, or both.

SYSPRINT — A data set containing the assembly listing (if the LIST option is in effect).

SYSTERM — A data set containing a condensed form of SYSPRINT, principally flagged statements and their error messages (only if the TERM option is in effect).

SYSPUNCH — A data set containing object module output (only if the DECK option is in effect).

SYSLIN — A data set containing object module output usually for the linkage editor, loader, or binder (only if the OBJECT option or XOBJECT option is in effect).

SYSADATA A data set containing associated data output (only if the ADATA option is in effect).

The data sets listed above are described on page 140. Figure 50 describes the characteristics of these datasets, including the characteristics set by the assembler and those you can over ride. The standard ddname that defines the data set appears as the heading for each data set description.

Figure 50. Assembler Data Set Characteristics

Data Set	**Access Method**	**Logical Record Length (LRECL)**	**Block Size (BLKSIZE)**	**Record Format (RECFM)**
SYSUT1	BSAM	Same as BLKSIZE	4	F
SYSIN	QSAM	80	5	7
SYSLIB	BPAM	80	6	7
SYSPRINT	QSAM	1	5 10	8
SYSTERM	QSAM	2	5 10	9
SYSPUNCH	QSAM	80	5	7
SYSLIN	QSAM	3	5	7
SYSADATA	QSAM	8188	8192 or greater. 10	VB

Notes to Figure 50:

1 If you specify EXIT(PRTEXIT) and the user exit specifies the logical record length, the logical record length returned is used. If EXIT(PRTEXIT) has not been specified or the user exit does not specify a record length, the record length from the DD statement or data set label is used if present. Otherwise, the record length defaults to 133.

The minimum record length allowed for SYSPRINT is 121, and the maximum allowed is 255.

2 If you specify EXIT(TRMEXIT) and the user exit specifies the logical record length, the logical record length returned is used. If EXIT(TRMEXIT) has not been specified or the user exit does not specify a record length, the record length from the DD statement or data set label is used if present. If not present, the record length defaults to the record length for SYSPRINT (if the LIST option is in effect) or 133 otherwise.

The maximum record length allowed for SYSTERM is 255.

3 If you specify the OBJECT option the logical record length for SYSLIN must be 80. If you specify the XOBJECT option the object module can be generated with either fixed-length records of 80 bytes, or variable-length records up to 8212 bytes.

Hierarchical File System: If you wish to copy the object data set to a file in a Hierarchical File System, for example under MVS OpenEdition*, the object data set must be created with fixed-length records.

4 You can specify a block size (BLKSIZE) between 2008 and 32760 bytes on the DD statement or in the data set label. The BLKSIZE should be a multiple of 8. If it is not, it is rounded to the next lower multiple of 8. If you do not specify BLKSIZE, the assembler sets the block size to 4088.

5 If specified, the BLKSIZE must equal the LRECL or be a multiple of the LRECL. If BLKSIZE is not specified, it is set to LRECL.

Refer to the applicable *Linkage Editor and Loader* manual, or *DFSMS/MVS Program Management* manual, for the block size requirements of SYSPUNCH and SYSLIN, if you use them as input to the linkage editor, or DFSMS/MVS Binder.

6 The BLKSIZE on the DD statement or the data set label must be equal to or be a multiple of the LRECL.

7 Set by the assembler to F or FB if necessary.

8 Set by the assembler to F or FB if necessary. If the DD statement or data set label specifies machine or ASA control characters, the ASA option is set or reset accordingly. If machine or ASA control characters are not specified on the DD statement or data set label, the record format is set to FA (or FBA) if the ASA option is specified or FM (or FBM) otherwise.

9 Set by the assembler to F or FB if necessary. The record format is set to FA (or FBA) if the ASA option is specified or FM (or FBM) otherwise.

10 High Level Assembler supports MVS/DFP* System-Determined Blocksize (SDB) for all output data sets except SYSLIN and SYSPUNCH.

System-Determined Blocksize is applicable when all of the following conditions are true:

- The operating system is MVS/ESA with a MVS/DFP level of 3.1 or higher.
- The data set is NOT allocated to SYSOUT.
- A block size of zero is specified or the blocksize is not specified in the JCL.
- A record length (LRECL) is specified.
- A record format (RECFM) is specified.
- A data set organization (DSORG) is specified.

If these conditions are met, MVS/DFP selects the appropriate blocksize for a new data set depending on the device type selected for output.

If the System-Determined Blocksize feature is not available, and your JCL omits the blocksize, or specifies a blocksize of zero, the assembler uses the logical record length as the blocksize.

Work Data Set: SYSUT1

The assembler uses this work data set as an intermediate external storage device when processing the source program. The input/output device assigned to this data set must be a direct-access device. The assembler does not support multi-volume utility data sets.

This data set is only used if there is insufficient virtual storage allocated to assemble the program in storage.

Specifying the Source Data Set: SYSIN

Define the data sets that contain your source code with the SYSIN DD statement.

```
//SYSIN    DD   DSN=datasetname,DISP=SHR
```

This data set contains the input to the assembler—the assembler language source statements to be processed.

You can place your assembler source code in the input stream. To do this, use this SYSIN DD statement:

```
//SYSIN    DD   *
```

When you use the (*) DD parameter, the source code must follow the DD statement. If another job step follows the assembly, the EXEC statement for that step must follow the last source statement, or end-of-file (/*) statement.

The IBM-supplied High Level Assembler procedures do not contain the SYSIN DD statement. The DD statement for SYSIN must be provided in the input stream:

```
//STEP1    EXEC ASMAC
//SYSIN    DD   *
⋮
assembler source statements
⋮
/*
```

Specifying Macro and Copy Code Libraries: SYSLIB

Define the partitioned data sets that contain your macro or copy members with the SYSLIB DD statement:

```
//SYSLIB   DD   DSN=SYS1.MACLIB,DISP=SHR
```

From this data set, the assembler obtains macro definitions and assembler language statements to be called by the COPY assembler instruction. Each macro definition or sequence of assembler language statements is a separate member in a partitioned data set. The member name is the operation code used to invoke the macro in a macro instruction, or the operand name in a COPY instruction.

The data set can be defined as SYS1.MACLIB, or your private macro definition or COPY library. SYS1.MACLIB contains macro definitions for the system macro instructions provided by IBM. Your private library may be concatenated with SYS1.MACLIB. The two libraries must have the same logical record length (80 bytes), but the blocking factors may be different. The applicable *JCL Reference* explains the concatenation of data sets.

Specifying the Listing Data Set: SYSPRINT

Define the data set that contains your listing output with the SYSPRINT DD statement.

```
//SYSPRINT DD   SYSOUT=A
```

The assembler uses this data set to produce a listing. You can direct output to a printer, a magnetic tape, or a direct-access storage device. The assembler uses ASA or machine control characters for this data set according to the ASA option.

Directing assembler messages to your terminal: SYSTERM

Define the data set that contains your terminal message's output with the SYSTERM DD statement.

```
//SYSTERM  DD   SYSOUT=A
```

Under TSO, the terminal messages can be sent to your terminal by using the following ALLOC statement.

```
ALLOC F(SYSTERM) DA(*)
```

This data set is used by the assembler to store a condensed form of SYSPRINT containing flagged statements and their associated error messages. It is intended for output to a terminal, but can also be routed to a printer, a magnetic tape, or a direct-access storage device. Depending on the ASA option, the assembler uses ASA or machine control characters to skip to a new line for this data set.

Specifying Object Code Data Sets: SYSLIN and SYSPUNCH

Define the data set that contains your object output with the SYSLIN and SYSPUNCH DD statements. When the OBJECT or XOBJECT option is in effect, the object module is written to SYSLIN. When the DECK option is in effect, the object module is written to SYSPUNCH. When both OBJECT and DECK options are in effect, the object module is written to both SYSLIN and SYSPUNCH.

You can direct the SYSLIN data set to either a card punch or an intermediate storage device capable of sequential access.

```
//SYSLIN   DD   DSN=dsname,UNIT=SYSDA,
//              SPACE=(subparms),DISP=(MOD,PASS)
```

You can direct the SYSPUNCH data set to either a card punch or an intermediate storage device capable of sequential access.

```
//SYSPUNCH DD   SYSOUT=B
```

Specifying the Associated Data Data Set: SYSADATA

Define the data set that contains your associated data output with the SYSADATA DD statement.

```
//SYSADATA DD   DSN=dsname,UNIT=SYSDA,
//              SPACE=(subparms),DISP=(MOD,PASS)
```

The associated data data set contains information regarding the assembly. It provides information for use by symbolic debugging and cross-reference tools. The SYSADATA data set must be directed to an intermediate storage device capable of sequential access.

Return Codes

High Level Assembler issues return codes for use with the COND parameter of the JOB and EXEC job control language statements. The COND parameter enables you to skip or to run a job step, depending on the results (indicated by the return code) of a previous job step. It is explained in the applicable *JCL Reference*.

The return code issued by the assembler is the highest severity code that is associated with any error detected in the assembly or with any MNOTE message produced by the source program or macro instructions. The return code can be controlled by the FLAG(*n*) assembler option described on page 43. See Appendix G, "High Level Assembler Messages" on page 271 for a listing of the assembler errors and their severity codes.

Chapter 8. Linking and Running your Program under MVS

The output from an assembly is an *object module*. An object module is a relocatable module of machine code that is not executable.

Before an object module can be executed, you must use one of the following components to convert it into executable machine code:

- The linkage editor and loader component of MVS/DFP
- A program management component of DFSMS/MVS

This section introduces these components, and helps you prepare your program for execution.

Using MVS/DFP to Process Object Modules

You use the *linkage editor* and *loader* component of MVS/DFP to convert object modules into executable programs and store them in program libraries. When a load module is to be executed, *program fetch* prepares the module for execution by loading it into virtual storage.

Refer to *MVS/DFP Linkage Editor and Loader*, SC26-4564 for details about using the linkage editor and loader. Figure 51 shows how the linkage editor and loader components are used to prepare an executable program.

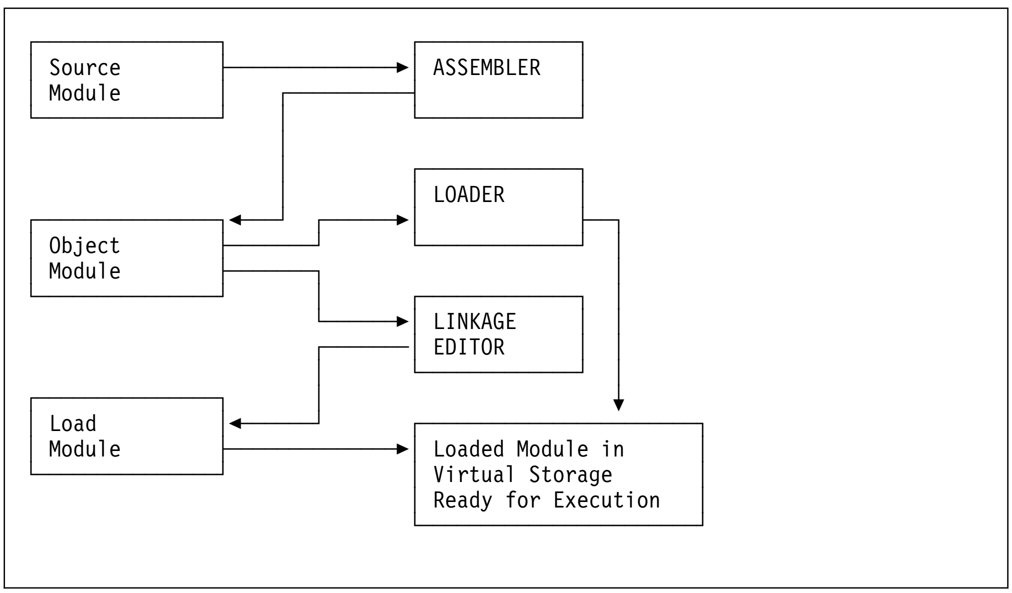

Figure 51. Using the Linkage Editor and Loader

Using DFSMS/MVS to Process Object Modules

You use the program management components of DFSMS/MVS to convert object modules into executable programs, store them in program libraries, and load them into virtual storage for execution. You can use the program management *binder* and *loader* to perform these tasks. These components can also be used in conjunction with the *linkage editor*. A load module produced by the linkage editor can be accepted as input by the binder, or can be loaded into storage for execution by the loader.

Refer to *DFSMS/MVS Program Management*, SC26-4916 for details about program management services. Figure 52 on page 145 shows how the program management components work together, and how each one is used to prepare an executable program.

The Program Management Binder

The binder converts object modules into an executable program unit that can either be read directly into virtual storage for execution, or stored in a program library. Executable program units can either be load modules, or *program objects*. You can use the binder to:

- Convert object or load modules, or program objects, into a program object and store it in a PDSE program library.
- Convert object or load modules, or program objects, into a load module and store it in a partitioned data set program library.
- Convert object or load modules, or program objects, into an executable program in virtual storage and execute the program.

For the remainder of this section, the binder is referred to as the *linker*, unless otherwise stated.

The Linkage Editor

The linkage editor converts object modules into a load module that is stored in a partitioned data set program library. A load module is loaded into storage for execution by program fetch (MVS/DFP), or the program management loader (DFSMS/MVS).

For the remainder of this section, the linkage editor is referred to as the *linker*, unless otherwise stated.

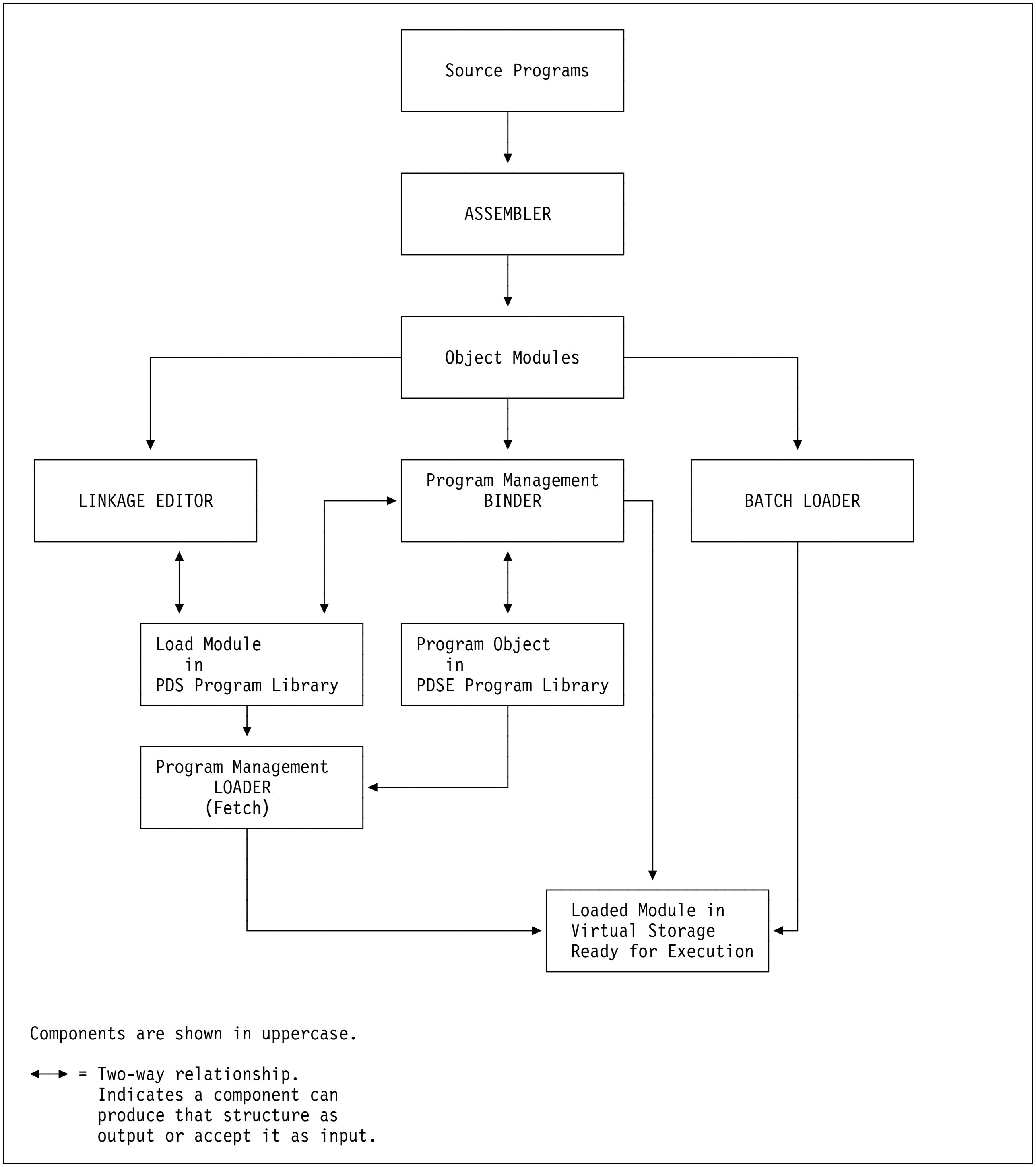

Figure 52. Using the Program Management Components

The Loader

The loader component of MVS/DFP, and the batch loader component of DFSMS/MVS perform the same task; therefore the batch loader is hereafter referred to as the *loader*, unless otherwise stated.

The loader combines the basic editing and loading services that can also be provided by the linkage editor and program fetch into one step. The loader accepts object modules and load modules, and loads them into virtual storage for execution. The loader does not produce load modules that can be stored in program libraries.

To keep a load module for later execution, use the linkage editor or binder.

Creating a Load Module

The linker processes your object module (assembled source program) and prepares it for execution. The processed object module becomes a load module or program object.

Optionally, the linker can process more than one object module, or load module, and convert them into one or more load modules, or program objects, by using the NAME control statement. See “Batch Assembling” on page 136 for an example that uses the NAME control statement.

Creating a Load Module under MVS

Figure 53 shows the general job control for creating a load module or program object.

```
//jobname  JOB  acctno,name,MSGLEVEL=1
⋮
//stepname EXEC PGM=HEWL,PARM=(options)
//SYSPRINT DD   SYSOUT=A
//SYSLMOD  DD   DSN=&&name(member),UNIT=SYSDA,
//              DISP=(NEW,PASS),SPACE=(subparms)
//SYSLIB   DD   DSN=dsname,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(subparms)
//SYSLIN   DD   DSN=MYOBJ,DISP=SHR
```

Figure 53. Sample job control for creating a load module

The SYSUT1 DD statement is used by the linkage editor, and ignored by the binder.

High Level Assembler provides cataloged procedures for the following:

- Assembly and link
- Assembly, link, and go (to execute your program)
- Assembly and go using the loader.

See “Using Cataloged Procedures” on page 155.

Creating a Load Module under TSO

You can invoke the linker under TSO (Time Sharing Option) with the LINK and LOADGO commands.

The LINK command creates a program module and saves it in either a partitioned data set or PDSE program library. If you run the LINK command in a system with DFSMS/MVS, you can use the BINDER and NOBINDER option on the LINK command to control whether your object module is linked using the binder or the MVS/DFP linkage editor.

The LOADGO command creates and executes a program module. The module is not saved in a program library.

Examples Using the LINK Command: If your assembly produced an object module in a data set called `PROGRAM1.OBJ`, issue the following LINK command at your terminal:

```
LINK PROGRAM1
```

The program module is placed by default in member TEMPNAME of a partitioned data set, or PDSE program library called *userid*.PROGRAM1.LOAD. If you want to put the program module in a different data set, issue the following LINK command:

```
LINK PROGRAM1 LOAD(data-set-name(member-name))
```

where *data-set-name* is a program library, and and *member-name* is the name of the program module.

The following example shows how to link two object modules and place the resulting program module in member TEMPNAME of the *userid*.LM.LOAD data set:

```
LINK PROGRAM1,PROGRAM2 LOAD(LM)
```

If your program refers to other modules, that is external references, you can instruct the linker to search for them by including the LIB parameter on the LINK command. The LIB parameter specifies one or more names of library data sets to search. For example:

```
LINK PROGRAM1 LIB('SALESLIB.LIB.SUBRT2')
```

This request searches library SALESLIB.LIB.SUBRT2.

You can also specify link options on the LINK and LOADGO commands. See “Specifying Linker Options Using the TSO LINK Command” on page 151.

Linker options are discussed in “Linker Processing Options” on page 150.

For more information about using the LINK and LOADGO commands, see the *TSO/E Command Reference*.

Input to the Linker

Your input to the linker can be:

- One or more object modules
- Linker control statements (that you can generate using the PUNCH assembler statement)

- Previously linked program modules you want to combine into one load module

The *primary* input to the linker can be:

- A sequential data set
- A member of a partitioned data set
- A member of a PDSE (if you are using the binder to link your program)
- Concatenated data sets of any combination of the above

The primary input data set can contain object modules, linker control statements, and linked program modules.

You specify the primary input data set with the SYSLIN DD statement.

Secondary input to the linker can consist of object modules or program modules that are not part of the primary input data set but are included explicitly or automatically in the program module using the *automatic library call* process.

An automatic call library contains modules that you can use as secondary input to the linkage editor to resolve external symbols left undefined after all primary input has been processed.

The automatic call library may be in the form of:

- Libraries containing object modules, with or without linkage editor control statements

- Libraries containing linked program modules

Secondary input for the linkage editor is composed of either all object modules or all load modules, but it cannot contain both types. Secondary input for the binder can be any combination of object modules, load modules libraries, and program object libraries.

You specify the secondary input data sets with a SYSLIB DD statement and, if the data sets are object modules, add the LIBRARY and INCLUDE control statements. If you have multiple secondary input data sets, concatenate them as follows:

```
//SYSLIB   DD   DSNAME=ORDERLIB,DISP=SHR
//         DD   DSNAME=SALESLIB,DISP=SHR
```

In this case, both the partitioned data set (library) named ORDERLIB and SALESLIB are available as the automatic call library. The LIBRARY control statement has the effect of concatenating any specified member names with the automatic call library.

Data Sets for Linker Processing

You need the following data sets for linker processing. Others may be necessary if you have several additional libraries or object modules. If you need additional libraries and object modules, include a DD statement for them in your JCL. Figure 54 summarizes the data sets for you need for linking.

Figure 54. Data Sets Used for Linking

DD name	Type	Function
SYSLIN[1]	Input	Primary input data, normally the output of the assembler
SYSPRINT[1]	Output	Diagnostic messages Informative messages Module map Cross reference list
SYSLMOD[1]	Output	Output data set for the program module
SYSUT1[1]	Utility	Work data set. Not used by the binder.
SYSLIB	Library	Automatic call library
SYSTERM[2]	Output	Numbered error or warning messages
User specified[3]		Additional object modules and program modules

Notes:

1 Required data set
2 Required if TERM option is specified
3 Optional data set

Additional Object Modules as Input

You can use the INCLUDE and LIBRARY control statements to:

1. Specify additional object modules you want included in the program module (INCLUDE statement).
2. Specify additional libraries to search for object modules to include in the program module (LIBRARY statement). This statement has the effect of concatenating any specified member names with the automatic call library.

Figure 55 shows an example that uses the INCLUDE and LIBRARY control statements.

```
⋮
//SYSLIN   DD   DSNAME=&&GOFILE,DISP=(SHR,DELETE)
//         DD   *
 INCLUDE MYLIB(ASMLIB,ASSMPGM)
 LIBRARY ADDLIB(COBREGN0)
/*
```

Figure 55. INCLUDE and LIBRARY control statements

Data sets you specify on the INCLUDE statement are processed as the linker encounters the statement. In contrast, data sets you specify on the LIBRARY statement are used only when there are unresolved references after all the other input is processed.

Output from the Linker

SYSLMOD and SYSPRINT are the data sets used for linker output. The output varies depending on the options you select, as shown in Figure 56.

Figure 56. Options for Controlling Linker Output

To Get This Output	Use This Option
A map of the program modules generated by the linker.	MAP
A cross-reference list of data variables	XREF
Informative messages	Default
Diagnostic messages	Default
Listing of the linker control statements	LIST
One or more program modules (which you must assign to a library)	Default

You always receive diagnostic and informative messages as the result of linking. You can get the other output items by specifying options in the PARM parameter of the EXEC statement in your JCL.

The program modules are written to the data set defined by the SYSLMOD DD statement in your JCL. Diagnostic output is written to the data set defined by the SYSPRINT DD statement.

Linker Processing Options

Linker options can be specified in either of two ways:

- In your JCL
- When you invoke the LINK or LOADGO command under TSO

Figure 57 describes some of these options.

Figure 57 (Page 1 of 2). Link Processing Options

Option	Action	Comments
LET	Lets you specify the severity level of an error, to control whether the linker marks the program module as non-executable.	The LET option is used differently between the linkage editor and the binder.
MAP NOMAP	Use MAP if you want to get a map of the generated program modules. NOMAP suppresses this map listing.	The map of the program module gives the length and location (absolute addresses) of the main program and all subprograms. NOMAP is the default.
NCAL	When you use the no automatic library call option (NCAL), the linker does not search for library members to resolve external references.	If you specify NCAL, you don't need to use the LIBRARY statement, and you don't need to supply the SYSLIB DD statement.

Figure 57 (Page 2 of 2). Link Processing Options

Option	Action	Comments
RENT NORENT	The RENT option indicates to the linker that the object module is reenterable and can be used by more than one task at a time. This type of module cannot be modified by itself or any other module when it is running. The assembler RENT option can be used to assist in determining whether the object module is reentant. NORENT indicates that the object module is not reentrant.	The assembler RENT option and linker RENT option are independent of each other. NORENT is the default linker option.
AMODE 24 | 31 | ANY	Use AMODE (addressing mode) to override the default AMODE attribute established by the assembler.	See "AMODE and RMODE Attributes" on page 152.
RMODE 24 | ANY	Use RMODE (residence mode) to override the default RMODE attribute established by the assembler.	See "AMODE and RMODE Attributes" on page 152.
PRINT	When you use the TSO commands LINK or LOADGO, the PRINT option specifies where to print diagnostic messages and the module map. PRINT is also an option of the loader, and controls whether diagnostic messages are produced.	See also "Specifying Linker Options Using the TSO LINK Command" on page 151.

Specifying Linker Options through JCL

In your link JCL, use the PARM statement to specify options:

```
PARM=(linker-options)
PARM.stepname=('linker-options')
```

linker-options
: A list of linker options (see Figure 57 on page 150). Separate the options with commas.

stepname
: The name of the step in the cataloged procedure that contains the PARM statement.

Specifying Linker Options Using the TSO LINK Command

You specify linker options on the LINK and LOADGO commands. The following example shows you how to specify the LET, MAP, and NOCALL options when you issue the LINK command:

```
LINK PROGRAM1 LET LET NOCALL
```

You can use the PRINT option to display the module map at your terminal:

```
LINK PROGRAM1 MAP PRINT(*)
```

The * indicates that the output from the linker is displayed at your terminal. NOPRINT suppresses any messages.

AMODE and RMODE Attributes

Every program that runs in MVS/ESA is assigned two attributes, an AMODE (addressing mode) and an RMODE (residency mode).

AMODE specifies the addressing mode in which the program is designed to receive control. Generally, the program is also designed to run in that mode, although a program can switch modes and can have different AMODE attributes for different entry points within a program module.

MVS/ESA uses a program's AMODE attribute to determine whether a program invoked using ATTACH, LINK, or XCTL is to receive control in 24-bit or 31-bit addressing mode.

RMODE indicates where the program can reside in virtual storage.

MVS/ESA uses the RMODE attribute to determine whether a program must be loaded into virtual storage below 16 megabytes, or can reside anywhere in virtual storage (above or below 16 megabytes).

Valid AMODE and RMODE specifications are:

Attribute	Meaning
AMODE=24	24-bit addressing mode
AMODE=31	31-bit addressing mode
AMODE=ANY	Either 24-bit or 31-bit addressing mode
RMODE=24	The module must reside in virtual storage below 16 megabytes. Use RMODE=24 for programs that have 24-bit dependencies.
RMODE=ANY	Indicates that the module can reside anywhere in storage, which includes addresses above the 16 megabyte line.

If you don't specify the AMODE or RMODE in the assembler program or when you link the program, both AMODE and RMODE default to 24.

Overriding the Defaults

The following examples show you how to override the default AMODE and RMODE values:

- Using the EXEC JCL statement:

```
//LKED     EXEC PGM=IEWBLINK,
//              PARM='AMODE=31,RMODE=ANY'
```

- Using the TSO commands LINK or LOADGO

```
LINK PROGRAM1 AMODE(31) RMODE(ANY)

or

LOADGO PROGRAM1 AMODE(31) RMODE(ANY)
```

You can also use linker control statements to override the default AMODE and RMODE values.

Detecting Linker Errors

The linker produces a listing in the data set defined by the SYSPRINT DD statement. The listing includes any informational or diagnostic messages issued by the linker. You should check the load map to make sure that all the modules you expected were included.

When linking your program, do not be concerned if you get messages about unresolved, "weak" external references. For example if you obtain the following results:

Figure 58. Linker Output for Unresolved External References

Location	Refers to Symbol	In Control Section
6A8	WXTRNNAM	$UNRESOLVED(W)
6AC	EXTRNNAM	$UNRESOLVED

WXTRNNAM is a "weak" external reference; you should not be concerned about it.

EXTRNNAM is a "strong" external reference, which you should resolve for the program module to run correctly.

Running your Assembled Program

When you've completed the preparatory work for your assembler program (designing, coding, assembling, and linking), the program is ready to run.

You can use cataloged procedures to combine the assemble, link, and go procedures in your programs. See "Using Cataloged Procedures" on page 155.

Running your Assembled Program in Batch

The general job control to run your program in batch is as follows:

```
//stepname EXEC PGM=progname[,PARM='user-parameters']
//STEPLIB  DD   DSN=library.dsname,DISP=SHR
//ddname   DD   (parameters for user-specified data sets)
⋮
```

Running your Assembled Program under TSO

You use the CALL command to run your program under TSO, as follows:

```
CALL 'CJM.LIB.LOAD(PROGRAM1)'
```

If you omit the descriptive qualifier (LOAD) and the member name (PROGRAM1), the system assumes LOAD and TEMPNAME, respectively. If your program module is in the data set CJM.LIB.LOAD(TEMPNAME), and your TSO userid is CJM, enter:

```
CALL LIB
```

Chapter 9. MVS System Services and Programming Considerations

This chapter describes some of the MVS system services and program development facilities that assist you in developing your assembler program. It provides the following information:

- Adding definitions to a macro library
- Using catalogued procedures
- Overriding statements in catalogued procedures
- Saving and restoring general register contents
- Ending program execution
- Accessing execution parameters
- Combining object modules to form a single program module
- Modifying program modules

Adding Definitions to a Macro Library

You can add macro definitions, and members containing assembler source statements that can be read by a COPY instruction to a macro library. You can use the system utility IEBUPDTE for this purpose. You can find the details of IEBUPDTE and its control statements in the *MVS/DFP 3.3 Utilities*, SC26-4559. Figure 59 shows how a new macro definition, NEWMAC, is added to the system library, SYS1.MACLIB.

```
//CATMAC   JOB        1,MSGLEVEL=1
//STEP1    EXEC       PGM=IEBUPDTE,PARM=MOD
//SYSUT1   DD         DSNAME=SYS1.MACLIB,DISP=OLD                          1
//SYSUT2   DD         DSNAME=SYS1.MACLIB,DISP=OLD                          1
//SYSPRINT DD         SYSOUT=A                                             2
//SYSIN    DD         DATA
./         ADD        LIST=ALL,NAME=NEWMAC,LEVEL=01,SOURCE=0               3
           MACRO                                                           4
           NEWMAC &OP1,&OP2
           LCLA   &PAR1,&PAR2

⋮
           MEND
./         ENDUP
/*
```

Figure 59. Macro Library Addition procedure

Notes to Figure 59:

1 The SYSUT1 and SYSUT2 DD statements indicate that SYS1.MACLIB, an existing program library, is to be updated.

2 Output from the IEBUPDTE program is printed on the Class A output device (specified by SYSPRINT).

3 The utility control statement, ./ ADD, and the macro definition follow the SYSIN statement. The ./ ADD statement specifies that the statements following it are to be added to the macro library under the name NEWMAC. When you include macro definitions in the library, the name specified in the NAME parameter of the ./ ADD statement must be the same as the operation code of the prototype statement of the macro definition.

4 Following the ADD utility control statement is the macro definition itself.

Using Cataloged Procedures

Often you use the same set of job control statements repeatedly, for example, to specify the assembly, linking, and running of many different programs. To save programming time and to reduce the possibility of error, standard sets of EXEC and DD statements can be prepared once and cataloged in a procedure library. Such a set of statements is termed a *cataloged procedure* and can be invoked by one of the following statements:

```
//stepname EXEC procname
//stepname EXEC PROC=procname
```

The specified procedure (*procname*) is read from the procedure library (SYS1.PROCLIB) and merged with the job control statements that follow this EXEC statement.

This section describes four IBM cataloged procedures: a procedure for assembling (ASMAC); a procedure for assembling and linking (ASMACL); a procedure for assembling, linking, and running (ASMACLG); and a procedure for assembling and running the loader (ASMACG).

Cataloged Procedure for Assembly (ASMAC)

This procedure consists of one job step: assembly. Use the name ASMAC to call this procedure. The result of running this procedure is an object module written to SYSPUNCH and an assembler listing. (See “Invoking the Assembler under MVS” on page 131 for more details and another example.)

In the following example, input is provided in the input stream:

```
//jobname  JOB
//stepname EXEC PROC=ASMAC
//SYSIN    DD   *
⋮
assembler source statements
⋮
/* (delimiter statement)
```

The statements of the ASMAC procedure are read from the procedure library and merged into the input stream.

Figure 60 on page 156 shows the statements that make up the ASMAC procedure.

```
//ASMAC    PROC
//*
//***   ASMAC
//*
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER AND CAN BE USED
//* TO ASSEMBLE PROGRAMS.
//*
//C        EXEC PGM=ASMA90,PARM=(OBJECT,NODECK)                              1
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR                                      2
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),UNIT=VIO,         3
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*                                                      4
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),UNIT=VIO,              5
//             DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FBS,BUFNO=1)
```

Figure 60. Cataloged Procedure for Assembly (ASMAC)

Notes to Figure 60:

1 PARM= or COND= parameters can be added to this statement by the EXEC statement that calls the procedure (see "Overriding Statements in Cataloged Procedures" on page 162). The system name ASMA90 identifies High Level Assembler.

2 This statement identifies the macro library data set. The data set name SYS1.MACLIB is an IBM designation.

3 This statement specifies the assembler work data set. The device class name used here, VIO, represents a direct-access unit. The I/O unit assigned to this name is specified by the installation when the operating system is generated. A unit name such as 3390 or SYSDA can be substituted for VIO.

4 This statement defines the standard system output class, SYSOUT=*, as the destination for the assembler listing.

5 This statement describes the data set that contains the object module produced by the assembler.

Cataloged Procedure for Assembly and Link (ASMACL)

This procedure consists of two job steps: assembly and link. Use the name ASMACL to call this procedure. This procedure produces an assembler listing, the linker listing, and a program module.

The following example shows input to the assembler in the input job stream. SYSLIN contains the output from the assembly step and the input to the link step. It can be concatenated with additional input to the linker as shown in the example. This additional input can be linker control statements or other object modules.

An example of the statements entered in the input stream to use this procedure is:

```
//jobname       JOB
//stepname      EXEC PROC=ASMACL
//C.SYSIN       DD  *
⋮
assembler source statements
⋮
/*
//L.SYSIN       DD  *
⋮
object module or linker control statements
/*
```

//L.SYSIN is necessary only if the linker is to combine modules or read editor control information from the job stream.

Figure 61 shows the statements that make up the ASMACL procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMACL   PROC
//*
//***   ASMACL
//*
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER, LINKS THE
//* NEWLY ASSEMBLED PROGRAM
//*
//C        EXEC PGM=ASMA90,PARM=(OBJECT,NODECK)
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),UNIT=VIO,
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),UNIT=VIO,          1
//             DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FBS,BUFNO=1)
//L        EXEC PGM=HEWL,PARM='MAP,LET,LIST,NCAL',COND=(8,LT,C)          2
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)                              3
//         DD  DDNAME=SYSIN                                             4
//SYSLMOD  DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(1,1,1)),             5
//             DSN=&&GOSET(GO)
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,    6
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*                                                 7
```

Figure 61. Cataloged Procedure for Assembling and Linking (ASMACL)

Notes to Figure 61:

1 In this procedure, the SYSLIN DD statement describes a temporary data set, the object module, which is passed to the linker.

2 This statement runs the linker. The linker options in the PARM field cause the linker to produce a cross-reference table, a module map, and a list of all control statements processed by the linker. The NCAL option suppresses the automatic library call function of the linker.

3 This statement identifies the linker input data set as the same one (SYSLIN) produced as output from the assembler.

4 This statement is used to concatenate any input to the linker from the input stream (object decks, linker control statements, or both) with the input from the assembler.

5 This statement specifies the linker output data set (the program load module). As specified, the data set is deleted at the end of the job. If it is required to retain the program module, the DSN parameter must be respecified and a DISP parameter added. See “Overriding Statements in Cataloged Procedures” on page 162. If you want to retain the output of the linker the DSN parameter must specify a library name and a member name at which the program module is to be placed. The DISP parameter must specify either KEEP or CATLG.

6 This statement specifies the work data set for the linker.

7 This statement identifies the standard output class as the destination for the linker listing.

Cataloged Procedure for Assembly, Link, and Run (ASMACLG)

This procedure consists of three job steps: assembly, link, and run. Use the name ASMACLG to call this procedure. It produces an assembler listing, an object module, and a linker listing.

The statements entered in the input stream to use this procedure are:

```
//jobname         JOB
//stepname        EXEC PROC=ASMACLG
//C.SYSIN         DD  *
⋮
assembler source statements
⋮
/*
//L.SYSIN         DD  *
⋮
object module or linker control statements
⋮
/*
//G.ddname        DD   (parameters)
//G.ddname        DD   (parameters)
//G.ddname        DD   *
⋮
program input
⋮
/*
```

//L.SYSIN is necessary only if the linker is to combine modules or read linker control information from the job stream.

//G.ddname statements are included only if necessary.

Figure 62 shows the statements that make up the ASMACLG procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMACLG PROC
//*
//***   ASMACLG
//*
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER, LINKS THE
//* NEWLY ASSEMBLED PROGRAM AND RUNS THE PROGRAM AFTER
//* THE LINK IS ACCOMPLISHED.
//*
//C        EXEC PGM=ASMA90,PARM=(OBJECT,NODECK)
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),UNIT=VIO,
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),UNIT=VIO,
//             DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FBS,BUFNO=1)
//L        EXEC PGM=HEWL,PARM='MAP,LET,LIST,NCAL',COND=(8,LT,C)         1
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(1,1,1)),             2
//             DSN=&&GOSET(GO)
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(1024,(120,120),,,ROUND),UNIT=VIO,
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//G        EXEC PGM=*.L.SYSLMOD,COND=((8,LT,C),(8,LT,L))                3
```

Figure 62. Cataloged Procedure for Assembly, Link, and Run (ASMACLG)

Notes to Figure 62:

1 The LET linker option specified in this statement causes the linker to mark the program module as executable even if errors are encountered during processing.

2 The output of the linker is specified as a member of a temporary data set, residing on a direct-access device, and is to be passed to a following job step.

3 This statement runs the assembled and linker program. The notation *.L.SYSLMOD identifies the program to be run as being in the data set described in job step L by the DD statement named SYSLMOD.

Cataloged Procedure for Assembly and Run (ASMACG)

This procedure consists of two job steps: assembly and run using the loader. Program modules for program libraries are not produced.

Enter these statements in the input stream to use this procedure:

```
//jobname       JOB
//stepname      EXEC PROC=ASMACG
//C.SYSIN       DD  *
⋮
assembler source statements
⋮
/*
//G.ddname      DD   (parameters)
//G.ddname      DD   (parameters)
//G.ddname      DD   *
⋮
program input
⋮
/*
```

//G.ddname statements are included only if necessary.

Figure 63 shows the statements that make up the ASMACG procedure. Only those statements not previously discussed are explained in the figure.

Use the name ASMACG to call this procedure. Assembler and loader listings are produced. See Figure 63.

```
//ASMACG   PROC
//*
//***   ASMACG
//*
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER AND WILL USE
//* THE LOADER PROGRAM TO RUN THE NEWLY ASSEMBLED PROGRAM.
//*
//C        EXEC PGM=ASMA90,PARM=(OBJECT,NODECK)
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),UNIT=VIO,
//             DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSPUNCH DD  SYSOUT=B
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),UNIT=VIO,
//             DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FBS,BUFNO=1)
//G        EXEC PGM=LOADER,PARM='MAP,LET,PRINT,NOCALL',COND=(8,LT,C)      1
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)                                 2
//         DD  DDNAME=SYSIN
//SYSLOUT  DD  SYSOUT=*                                                    3
```

Figure 63. Cataloged Procedure for Assembly and Running using the Loader (ASMACG)

Notes to Figure 63:

1 This statement runs the loader. The loader options in the PARM field cause the loader to produce a map and print the map and diagnostics. The NOCALL option is the same as NCAL for the linker, and the LET option is the same as for the linker.

2 This statement defines the loader input data set as the same one produced as output by the assembler.

3 This statement identifies the standard output class as the destination for the loader listing.

Overriding Statements in Cataloged Procedures

You can override any parameter in a cataloged procedure except the PGM= parameter in the EXEC statement. Overriding of statements or fields is effective only for the duration of the job step in which the statements appear. The statements, as stored in the procedure library of the system, remain unchanged.

To respecify, add, or nullify statements, include statements in the input stream that contain the required changes and identify the statements to be overridden.

EXEC Statements

Any EXEC parameter (except PGM) can be overridden. For example, the PARM= and COND= parameters can be added or, if present, respecified, by including them in the EXEC statement calling the procedure. The JCL notation to specify these parameters is:

```
//ASM          EXEC   PROC=ASMACLG,PARM.C=(NOOBJECT),COND.L=(8,LT,stepname.c)
```

stepname identifies the EXEC statement within the catalogued procedure (ASMACLG) to which the modification applies.

If the procedure consists of more than one job step, a PARM.procstepname= or COND.procstepname= parameter can be entered for each step. The entries must be in order (PARM.procstepname1=, PARM.procstepname2=, etc.).

DD Statements

All parameters in the operand field of DD statements can be overridden by including in the input stream (following the EXEC statement calling the procedure) a DD statement with the notation //*procstepname.ddname* in the name field. *procstepname* refers to the job step in which the statement identified by *ddname* appears.

If more than one DD statement in a procedure is to be overridden, the overriding statements must be in the same order as the statements in the procedure.

Examples of Cataloged Procedures

1. In the assembly procedure ASMAC (Figure 60 on page 156), you might want to suppress the object module to SYSPUNCH and respecify the UNIT= and SPACE= parameters of data set SYSUT1. In this case, the following statements are required:

   ```
   //stepname     EXEC   PROC=ASMAC,
   //                    PARM=NODECK
   //SYSUT1       DD     UNIT=3390,
   //                    SPACE=(4096,(300,40))
   //SYSIN        DD     *
   ⋮
   assembler source statements
   ⋮
   /*
   ```

2. In procedure ASMACLG (Figure 62 on page 160), you might want to suppress the assembler listing, and add the COND= parameter to the EXEC statement that invokes the linker. In this case, the EXEC statement in the input stream are:

```
//stepname    EXEC   PROC=ASMACLG,
//                   PARM.C=(NOLIST,OBJECT),
//                   COND.L=(8,LT,stepname.C)
```

For this run of procedure ASMACLG, no assembler listing is produced, and running of the linker job step //L would be suppressed if the return code issued by the assembler (step C) were greater than 8.

When you override the PARM field in a procedure, the whole PARM field is overridden. Thus, in this example, overriding the LIST parameter effectively deletes PARM=(OBJECT,NODECK). PARM=(OBJECT,NODECK) must be repeated in the override statement; otherwise, the assembler default values are used.

3. The following example shows how to use the procedure ASMACL (Figure 61 on page 158) to:

   1 Read input from a nonlabeled 9-track tape in unit 282 that has a standard blocking factor of 10.

   2 Put the output listing on a tape labeled TAPE10, with a data set name of PROG1 and a blocking factor of 5.

   3 Block the SYSLIN output of the assembler and use it as input to the linker with a blocking factor of 10.

   4 Link the module only if there are no errors in the assembly (COND=0).

   5 Link onto a previously allocated and cataloged data set USER.LIBRARY with a member name of PROG.

```
//jobname      JOB
//stepname     EXEC   PROC=ASMACL,
//                    COND.L=(0,NE,stepname.C)                       4
//C.SYSPRINT   DD     DSNAME=PROG1,UNIT=TAPE,                        2
//                    VOLUME=SER=TAPE10,DCB=(BLKSIZE=665)
//C.SYSLIN     DD     DCB=(BLKSIZE=800)                              3
//C.SYSIN      DD     UNIT=282,LABEL=(,NL),                          1
//                    DCB=(RECFM=FBS,BLKSIZE=800)
//L.SYSLIN     DD     DCB=stepname.C.SYSLIN                          3
//L.SYSLMOD    DD     DSNAME=USER.LIBRARY(PROG),DISP=OLD             5
/*
```

The order of appearance of overriding ddnames for job step C corresponds to the order of ddnames in the procedure; that is, SYSPRINT precedes SYSLIN within step C. The ddname C.SYSIN was placed last because SYSIN does not occur at all within step C. These points are covered in the applicable *JCL Reference*.

4. The following example shows assembly of two programs, link of the two object modules produced by the assemblies into one program module, and running the generated program. The input stream appears as follows:

```
//stepname1      EXEC    PROC=ASMAC,PARM=OBJECT
//SYSIN          DD      *
⋮
assembler source statements for program 1
⋮
/*
//stepname2      EXEC    PROC=ASMACLG
//C.SYSIN        DD      *
⋮
assembler source statements for program 2
⋮
/*
//L.SYSIN        DD      *
                 ENTRY PROG
/*
//G.ddname       DD      dd statements for G step
```

The applicable *JCL Reference* provides additional descriptions of overriding techniques.

Operating System Programming Conventions

Assembler programs executing under MVS must follow a set of programming conventions to save and restore registers, and access execution parameters. These conventions are described in the following sections.

Saving and Restoring General Register Contents

A program should save the values contained in the general registers when it receives control and, upon completion, restore to the general registers these same values. Thus, as control is passed from the operating system to a program and, in turn, to a subprogram, the status of the registers used by each program is preserved. This is done through use of the SAVE and RETURN system macro instructions.

Saving Register Contents: The SAVE macro instruction should be the first statement in the program. It stores the contents of registers 14, 15, and 0 through 12 in an area provided by the program that passes control. When a program is given control, register 13 contains the address of an area in which the general register contents should be saved.

If the program calls any subprograms, or uses any operating system services other than GETMAIN, FREEMAIN, ATTACH, and XCTL, it must first save the contents of register 13 and then load the address of an 18-fullword save area into register 13. This save area is in the program and is used by any subprograms or operating system services called by the program.

Restoring Register Contents: At completion, the program restores the contents of general registers 14, 15, and 0 through 12 by use of the RETURN system macro instruction (which also indicates program completion). The contents of register 13 must be restored before issuing the RETURN macro instruction.

Example: The coding sequence that follows shows the basic process of saving and restoring the contents of the registers. A complete discussion of the SAVE and RETURN macro instructions and the saving and restoring of registers is contained in the *MVS/ESA Programming: Assembler Services Reference*.

```
Name      Operation      Operand

BEGIN     SAVE           (14,12)
          USING          BEGIN,15
          .
          .
          ST             13,SAVEBLK+4
          LA             13,SAVEBLK
          .
program function source statements
          .
          L              13,SAVEBLK+4
          RETURN         (14,12)
SAVEBLK   DC             18F'0'
          .
          .
          END
```

Ending Program Execution

You indicate completion of an assembler language source program by using the RETURN system macro instruction to pass control from the terminating program to the program that initiated it. The initiating program might be the operating system or, if a subprogram issued the RETURN, the program that called the subprogram.

In addition to indicating program completion and restoring register contents, the RETURN macro instruction can also pass a return code—a condition indicator that can be used by the program receiving control.

If the program returns to the operating system, the return code can be compared against the condition stated in the COND= parameter of the JOB or EXEC statement.

If the program returns to another program, the return code is available in general register 15, and can be used as required. Your program should restore register 13 before issuing the RETURN macro instruction.

The RETURN system macro instruction is discussed in detail in the *MVS/ESA Programming: Assembler Services Reference*.

Accessing Execution Parameters (MVS)

You access information in the PARM field of an EXEC statement by refering to the contents of general register 1. When control is given to the program, general register 1 contains the address of a fullword which, in turn, contains the address of the data area containing the information.

The data area consists of a halfword containing the count (in binary) of the number of information characters, followed by the information field. The information field is aligned to a fullword boundary. Figure 64 on page 166 shows how the PARM field information is structured.

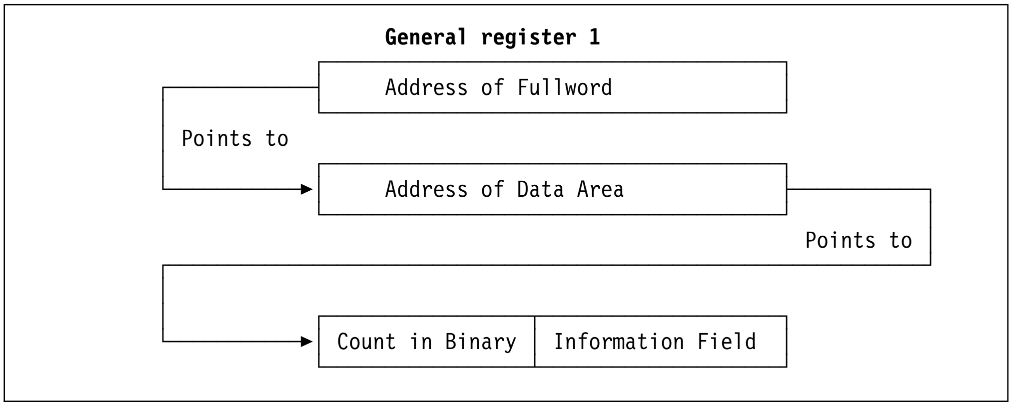

Figure 64. Access to PARM Field

Object Module Linkage

You can combine two or more object modules, whether generated by the assembler or by another language processor, to produce a single load module. The object modules can be combined by the linkage editor, or DFSMS/MVS binder, provided each object module conforms to the data formats and the required linkage conventions. This makes it possible for you to use different programming languages for different parts of your program, allowing each part to be written in the language best suited for it. Use the CALL system macro instruction to link an assembler language main program to subprograms produced by another language processor. Refer to the *MVS/ESA Programming: Assembler Services Reference* for details about linkage conventions and the CALL system macro instruction.

Figure 65 on page 167 is an example of statements used to establish the assembler language program linkage to subprograms. See the applicable language programmer's guide for information about calling the language from an assembler language program.

If any input/output operations are done by called subprograms, supply the correct DD statements for the data sets used by the subprograms. See the applicable language programmer's guide for an explanation of the DD statements and special data set record formats used for the language.

```
ENTRPT   SAVE        (14,12)
         LR          12,15
         USING       ENTRPT,12
         ST          13,SVAREA+4
         LA          15,SVAREA
         ST          15,8(,13)
         LR          13,15

⋮
         CALL        subprogram-name,(V1,V2,V3),VL

⋮
         L           13,SVAREA+4
         RETURN      (14,12)
SVAREA   DC          18F'0'
V1       DC          CL5'Data1'
V2       DC          CL5'Data2'
V3       DC          CL5'Data3'
         END
```

Figure 65. Sample Assembler Linkage Statements for calling Subprograms

Modifying Program Modules

If the editing functions of the linker are used to modify a program module, the entry point to the program module must be restated when the program module is reprocessed by the linker. Otherwise, the first byte of the first control section processed by the linker becomes the entry point. To enable restatement of the original entry point, or designation of a new entry point, the entry point must have been identified originally as an external symbol; that is, it must have appeared as an entry in the external symbol dictionary. The assembler automatically identifies external symbols if the entry point is the name of a control section or START statement; otherwise, you must use an assembler ENTRY statement to identify the entry point as an external symbol.

When a new object module is added to or replaces part of the load module, the entry point is restated in one of three ways:

- By placing the entry point symbol in the operand field of an EXTRN statement and an END statement in the new object module
- By using an END statement in the new object module to designate a new entry point in the new object module
- By using a linker ENTRY statement to designate either the original entry point or a new entry point for the program module

Further discussion of program module entry points is contained in the applicable *Linkage Editor and Loader* manual, or *DF/SMS Program Management* manual.

Part 3. Developing Assembler Programs under CMS

Chapter 10. Assembling Your Program under CMS

This chapter describes how to invoke the assembler under CMS (Conversational Monitor System). It describes:

- The input to the assembler
- The output from the assembler
- How to gain access to the High Level Assembler product files
- How to invoke the assembler under CMS
- How to assemble multiple source programs using the BATCH option
- Special options for invoking the assembler under CMS
- The data sets used by the assembler
- The assembler return codes
- Special diagnostic messages when invoking the assembler under CMS

To use this section effectively, you should be familiar with the assembler language described in the *Language Reference*.

The assembler language program can be run under control of CMS. For more information about CMS, refer to the applicable *CP Command Reference for General Users* and *CMS Command and Macro Reference*.

Input to the Assembler

As input, the assembler accepts a program written in the assembler language as defined in the *Language Reference*. This program is referred to as a source module. Some statements in the source module (macro or COPY instructions) can cause additional input to be obtained from a macro library.

Input can also be obtained from user exits. See Chapter 4, “Providing User Exits” on page 66 for more information.

Output from the Assembler

The output from the assembler can consist of an object module, a program listing, terminal messages and an associated data file. The object module is stored on your virtual disk in a TEXT file. You can bring it into your virtual storage and run it by using the CMS LOAD and START commands. The program listing lists all the statements in the module, both in source and machine language format, and gives other important information about the assembly, such as error messages. The listing is described in detail in Chapter 2, “Using the Assembler Listing” on page 7.

Accessing the Assembler

To access the High Level Assembler under CMS, you must first link to the mini-disk containing the assembler by issuing the CP LINK command. You must then issue the ACCESS command to assign a file mode, and make the mini-disk available to CMS. For example:

```
CP LINK PRODUCT 194 198 RR PASSWORD
ACCESS 198 B
```

In this example, you have linked to disk 194 of the virtual machine that contains the High Level Assembler product, and whose user ID is PRODUCT. You have defined disk 194 as 198 to your VM session. You have read access to the disk (RR) and you specified the read-share password for the 194 disk (PASSWORD).

After you linked to the 194 disk as 198, you accessed the 198 disk as disk B on your system. After you have access to the product disk, you can invoke the assembler using the ASMAHL command. (see "Invoking the Assembler under CMS")

If High Level Assembler is stored on your A-disk, or another disk to which you already have access, you can omit the CP LINK and ACCESS commands. If High Level Assembler is not on a disk that you have accessed, you can put the CP LINK and ACCESS commands into your PROFILE EXEC, which issues them for you each time you log on. For more information on the CP LINK and ACCESS commands, see the applicable *CP Command Reference* for your VM environment, as listed under "Related Publications (VM)" on page 332.

Invoking the Assembler under CMS

Use the ASMAHL command to invoke and control assembly of assembler source programs under CMS.

The format of the ASMAHL command is:

```
►►──ASMAHL──┬──────────┬──┬────────────────────────────┬──►◄
            └─filename─┘  │     ┌───────┐              │
                          │     │   ┌─,─┘              │
                          │     ▼   │                  │
                          └─(───option──┴──┬─────┬─────┘
                                           └─)───┘
```

where:

filename is the name of your assembler source program.

Use one of the three methods available for specifying your assembler source program. See "Specifying the Source File: SYSIN" on page 175 for details on each of these methods.

option represents one or more assembler options, separated by a blank or comma, that you want in effect during assembly. These assembler options are equivalent to the options you would specify on the PARM parameter of an EXEC job control statement, if you were invoking the assembler under MVS.

A complete list and discussion of assembler options can be found under Chapter 3, "Controlling your Assembly with Options" on page 34.

The assembler options in effect are determined by the default options that were set when High Level Assembler was installed, and by the options you specify with the ASMAHL command. There are also several assembler options that can only be specified when running under CMS; see "Controlling Your Assembly" on page 172.

Synonym for ASMAHL Command: Your installation might have created a synonym for ASMAHL when High Level Assembler was installed. See your system programmer for the specific command name.

Batch Assembling

You can assembler a sequence of separate assembler programs with a single invocation of the assembler, using the BATCH option. The object programs produced from this assembly can be link-edited into either a single load module or separate load modules.

When the BATCH option is specified, each assembler program in the sequence must be terminated by an END statement, including the last program in the batch. If an END statement is omitted, the program assembles with the next program in the sequence. If the END statement is omitted from the last program in the sequence, the assembler generates an END statement.

If separate load modules are to be produced, you must write a NAME linkage editor control statement for each load module. The NAME statement must be written at the end of the load module. The following example shows how to create two load modules SECT1 and SECT2.

```
SECT1    CSECT          Start of first load module
         .
         .
         Source instructions
         .
         .
         END            End of first load module
         PUNCH ' NAME SECT1(R)'
         END
SECT2    CSECT          Start of second load module
         .
         .
         Source instructions
         .
         .
         END            End of second load module
         PUNCH ' NAME SECT2(R)'
         END
```

If separate TEXT files are required, you must issue two separate ASMAHL commands.

Controlling Your Assembly

The assembly options are specified on the ASMAHL command after the left parenthesis. The options that can be specified to control your assembly are described inChapter 3, "Controlling your Assembly with Options" on page 34 .

Under CMS, there are additional options that can be specified. These are described in Chapter 3, "Controlling your Assembly with Options" on page 34, and consist of:

ERASE Deletes LISTING, TEXT, and SYSADATA files before the assembly begins.

LINECOUN Specifies the number of lines to be printed on each page of the assembler listing.

NOSEG Specifies that the assembler load modules are loaded from disk. (The default is to load the modules from the Logical Saved Segment (LSEG), but if the LSEG is not available then load the modules from disk).

PRINT Directs the assembler listing to the virtual printer, instead of to disk.

SEG Specifies that the assembler load modules are loaded from the Logical Saved Segment (LSEG). (The default is to load the modules from the LSEG, but if the LSEG is not available then load the modules from disk).

SYSPARM A question mark (?) can be specified in the SYSPARM string, which instructs the assembler to prompt you for a character string at your terminal.

Input and Output Files

Depending on the options in effect, High Level Assembler requires the following files, as shown in Figure 66:

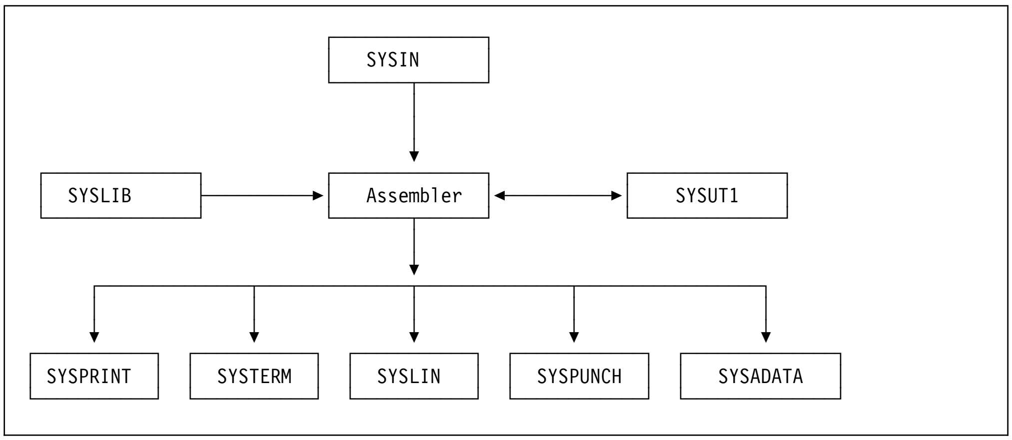

Figure 66. High Level Assembler Files

The ddnames can be overridden during installation.

High Level Assembler requires the following files:

SYSUT1 A work file used as intermediate external storage when processing the source program. This file is used when there is not enough main storage available to assemble in-storage.

If the value specified for the SIZE option is large enough, an in-storage assembly is done and the work file SYSUT1 can be omitted, though a warning message is issued.

SYSIN An input file containing the source statements to be processed.

In addition, the following six files might be required:

SYSLIB A file containing macro definitions (for macro definitions not defined in the source program), source code to be called for through COPY assembler instructions, or both.

SYSPRINT — A file containing the assembly listing (if the LIST option is in effect).

SYSTERM — A file containing essentially a condensed form of SYSPRINT, principally error flagged statements and their error messages (only if the TERM option is in effect).

SYSPUNCH — A file containing object module output, usually for punching (only if the DECK option is in effect).

SYSLIN — A file containing object module output usually for the linkage editor (only if the OBJECT option is in effect). Alternatively, you can specify the XOBJECT option to instruct the assembler to write the extended object format to this file.

SYSADATA — A file containing associated data output (only if the ADATA option is in effect).

The files listed above are described in the text following Figure 67. The characteristics of these files, those set by the assembler and those you can override, are shown in Figure 67.

Figure 67. Assembler File Characteristics

File	Access Method	Logical Record Length (LRECL)	Block Size (BLKSIZE)	Record Format (RECFM)
SYSUT1	BSAM	Same as BLKSIZE	3	F
SYSIN	QSAM	80	4	6 9
SYSLIB	BPAM	80	5	6 9
SYSPRINT	QSAM	1	4	7 9
SYSTERM	QSAM	2	4	8 9
SYSPUNCH	QSAM	80	4	6 9
SYSLIN	QSAM	80	4	6 9
SYSADATA	QSAM	8188	8192 or greater	VB

Notes to Figure 67:

1 If you specify EXIT(PRTEXIT) and the user exit specifies the logical record length, the logical record length returned is used. If you do not specify EXIT(PRTEXIT) or the user exit does not specify a record length, the record length from the FILEDEF command or file label is used, if present. Otherwise, the record length defaults to 133.

The minimum length allowed for SYSPRINT is 121, and the maximum allowed is 255.

2 If you specify EXIT(TRMEXIT) and the user exit specifies the logical record length, the logical record length returned is used. If you do not specify EXIT(TRMEXIT) or the user exit does not specify a record length, the record length from the FILEDEF command or file label is used, if present. If not present, the record length defaults to the record length for SYSPRINT (if the LIST option is in effect) or 133 otherwise.

The maximum record length allowed for SYSTERM is 255.

3 You can specify a block size (BLKSIZE) between 2008 and 32760 bytes on the FILEDEF command or in the file label. The BLKSIZE should be a multiple of 8. If it is not, it is rounded to the next lower multiple of 8. If you do not specify BLKSIZE, the assembler sets the block size to 4088.

4 If specified, the BLKSIZE must equal the LRECL or be a multiple of the LRECL. If BLKSIZE is not specified, it is set to LRECL.

5 The BLKSIZE on the FILEDEF command or the file label must equal the LRECL or be a multiple of the LRECL.

6 Set by the assembler to F or FB if necessary.

7 Set by the assembler to F or FB if necessary. If the FILEDEF command or file label specifies machine or ASA control characters, the ASA option will be set or reset accordingly. If machine or ASA control characters are not specified on the FILEDEF command or file label, the record format is set to FA (or FBA) if the ASA option is specified or FM (or FBM) otherwise.

8 Set by the assembler to F or FB if necessary. The record format is set to FA (or FBA) if the ASA option is specified or FM (or

9 You can specify B, S, or T, or any combination of these.

Work file: SYSUT1

The assembler uses this work file as an intermediate external storage device when processing the source program. The input/output device assigned to this file must be a direct-access device.

If no SYSUT1 FILEDEF command is issued before the ASMAHL command is issued, the following FILEDEF command is issued by the ASMAHL command:

```
FILEDEF SYSUT1 DISK fn SYSUT1 m4 (BLOCK 4088
```

where *fn* is the filename specified on the ASMAHL command, and the file mode *m*4 is set to use the read/write disk with the most available space. For example, if three read/write disks were accessed as the A, B, and D disks, and if the D disk had the most available space, then *m*4 would be set to 'D4' for use during the assembly.

This data set is only used if there is insufficient virtual storage allocated to assemble the program in storage.

Specifying the Source File: SYSIN

Use one of the following methods for specifying your assembler source program:

- Specify the filename of the assembler source program on the ASMAHL command line.
- Issue a FILEDEF for SYSIN before issuing the ASMAHL command.
- Supply source statements from a user-supplied module by using the EXIT assembler option.

Specify the Filename on the Command Line: Using this method, you specify the filename of your assembler source program on the ASMAHL command line. For example:

```
ASMAHL PROG1 (LIST,XREF(SHORT))
```

assembles the source program named PROG1 using the assembler options LIST and XREF(SHORT). The source program must have a filetype of ASSEMBLE. The ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSIN DISK PROG1 ASSEMBLE * (RECFM FB LRECL 80 BLOCK 16000
```

Issue a FILEDEF for SYSIN: Another method you can use to specify the assembler source program is to issue a FILEDEF for SYSIN before the assembly. The assembler then assembles the program specified in the FILEDEF. For example:

```
FILEDEF SYSIN DISK PROG2 ASSEMBLE A
ASMAHL (LIST,XREF)
```

Assembles the program named PROG2, using the options specified on the ASMAHL command line. When you issue a FILEDEF for SYSIN, the source program you specify with the FILEDEF is the one used for input by the assembler.

If the FILEDEF for SYSIN is issued and the FILEDEF specifies a DISK file, the filename on the ASMAHL command is optional. If the filename is specified on the ASMAHL command, the filename must match the filename of the file specified on the FILEDEF. Additionally, when using a FILEDEF, the file type need not be ASSEMBLE.

You can read MVS data sets and VSE files as CMS files by defining those data with the FILEDEF command. For example,

```
FILEDEF SYSIN DISK OSDS ASSEMBLE fm DSN OS DATASET (options...
```

You can also assemble a member of an OS partitioned data set or a CMS MACLIB by using the MEMBER parameter of the FILEDEF command. When you specify member parameter, the member name is used as the filename for the LISTING, TEXT and SYSADATA files.

If you want to assemble a source file that is in your CMS virtual reader, issue the following FILEDEF command:

```
FILEDEF SYSIN READER
```

and then issue the ASMAHL command. You must specify the filename on the ASMAHL command. The filename is used as the file name of the LISTING, TEXT and SYSADATA files.

Similarly, if you have a tape containing an assembler input file that you want to assemble, you must issue the following command before issuing the ASMAHL command:

```
FILEDEF SYSIN TAPn (RECFM F LRECL 80 BLOCK 80
```

If the blocksize of the file were 800 bytes, you could specify BLOCK 800 as in the preceding FILEDEF.

If the FILEDEF command specifies a tape file, the filename must be specified on the ASMAHL command. The filename is used as the filename of the LISTING, TEXT and SYSADATA files.

Make sure that any attributes specified for a file conform to the attributes expected by the assembler for the device.

Specify Source Using the EXIT Option: If you are using an input user exit to provide source statements to the assembler, the FILEDEF for SYSIN is not required. For example:

```
ASMAHL PROG2 (EXIT(INEXIT(INMOD1('ABCD'))),LIST,XREF(SHORT))
```

assembles the source statements provided by the input user module named `INMOD1` using the character string `ABCD`, and also the assembler options LIST and XREF(SHORT). (For specific details on using the EXIT assembler option, see page 41).

Specify the filename on the ASMAHL command or a FILEDEF for SYSIN before issuing the ASMAHL command as described above. This is required even if the assembler does not read the input file. The filename specified on the ASMAHL command or from the FILEDEF for SYSIN is used as the filename of the LISTING, TEXT and SYSADATA files.

If you specify the INEXIT option, the ASMAHL command does not check whether the input file exists. If the SOURCE user exit instructs the assembler to open the primary input file, the open fails if the file does not exist.

Specifying Macro and Copy Code Libraries: SYSLIB

If you don't issue SYSLIB FILEDEF before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSLIB DISK CMSLIB MACLIB * (RECFM FB LRECL 80 BLOCK 8000
```

Use the GLOBAL command to identify which CMS libraries are to be searched for macro definitions and COPY code. Private libraries and CMSLIB can be concatenated with each other in any order by the GLOBAL command. The format of this command is described in the applicable *CMS Command and Macro Reference*.

You can concatenate a CMS MACLIB with an OS partitioned data set. When this is required, the library with the largest blocksize must be specified first as in the following example.

```
FILEDEF SYSLIB DISK MYLIB   MACLIB M DSN ATR005.MACLIB
FILEDEF SYSLIB DISK OSMACRO MACLIB S (CONCAT
GLOBAL MACLIB MYLIB OSMACRO
```

Specifying the Listing File: SYSPRINT

If you specify the PRINT option, and you don't issue SYSPRINT FILEDEF before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSPRINT PRINTER
```

If you specify the DISK option, which is the default, and you don't issue SYSPRINT FILEDEF before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSPRINT DISK fn LISTING m1 (RECFM FB BLOCK 13300
```

where *fn* is the filename specified on the ASMAHL command. If the assembler source file (SYSIN input) is *not* on disk or is on a read-only disk, the file mode *m* is set to the first available read/write disk. If the source file is on a read/write disk, the mode letter *m* is set to the mode of that read/write disk. For example, if the source file were on a read/write B disk, the file mode *m*1 would be set to 'B1'.

You can issue a FILEDEF command for SYSPRINT before the ASMAHL command to direct the listing to the terminal, printer or a disk file. See "PRINT" on page 55 for details about the CMS options for SYSPRINT.

Directing assembler messages to your terminal: SYSTERM

If you don't issue a SYSTERM FILEDEF command before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSTERM TERMINAL
```

You can issue a FILEDEF command for SYSTERM before the ASMAHL command to direct the listing to the terminal, printer or a disk file.

Specifying Object Code Files: SYSLIN and SYSPUNCH

If you don't issue a SYSPUNCH or SYSLIN FILEDEF command before the ASMAHL command, the ASMAHL command issues the following FILEDEF commands:

```
FILEDEF SYSPUNCH PUNCH
FILEDEF SYSLIN DISK fn TEXT m1 (RECFM FB LRECL 80 BLOCK 16000
```

where *fn* is the filename specified on the ASMAHL command. If the assembler source file (SYSIN input) is *not* on disk or is on a read-only disk, the file mode *m* is set to the first available read/write disk. If the source file is on a read/write disk, the mode letter *m* is set to the mode of that read/write disk. For example, if the source file were on a read/write B disk, the file mode *m*1 would be set to 'B1'.

You can issue a FILEDEF command for SYSPUNCH or SYSLIN before the ASMAHL command is issued to direct the object output to the punch or a disk file.

Specifying the Associated Data File: SYSADATA

If you don't issue a SYSADATA FILEDEF command before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSADATA DISK fn SYSADATA m1 (RECFM VB LRECL 8188 BLOC
```

where *fn* is the filename specified on the ASMAHL command, and if the assembler source file (SYSIN input) is *not* on disk or is on a read-only disk, the file mode *m* is set to the first read/write disk. If the source file is on a read/write disk, the mode letter *m* is set to the mode of that read/write disk. For example, if the source file were on a read/write B disk, the file mode *m*1 would be set to 'B1'.

A FILEDEF command for SYSADATA can be issued before the ASMAHL command is issued to direct the associated data output to a different file.

Return Codes

High Level Assembler issues return codes that are returned to the caller. If High Level Assembler is called from an EXEC, the EXEC can check the return code.

The return code issued by the assembler is the highest severity code that is associated with any error detected in the assembly or with any MNOTE message produced by the source program or macro instructions. The return code can be controlled by the FLAG(*n*) assembler option described on page 43. See Appendix G, “High Level Assembler Messages” on page 271 for a listing of the assembler errors and their severity codes.

Diagnostic Messages Written by CMS

If an error occurs during the running of the ASMAHL command, a message might be written at the terminal and, at completion of the command, register 15 contains a nonzero return code.

Two types of messages might be issued:

- Messages that are issued by the assembler (see Appendix G, “High Level Assembler Messages” on page 271)
- Messages that are issued by the ASMAHL command processor (see “ASMAHL Command Error Messages” on page 310)

The messages issued by the ASMAHL command processor are in two parts: a message code and the message text. The message code is in the form 'ASMACMS*nnnt*', where ASMACMS indicates that the message was generated by the ASMAHL command program, *nnn* is the number of the message, and *t* is the type of message. The message text describes the error condition.

You can use the CP command SET EMSG to control what part of the diagnostic message to display. Figure 68 shows the SET EMSG options you can specify, and how they affect the message display.

Figure 68. CP SET EMSG Command Options

SET EMSG Option	Part of Message Displayed
CODE	Displays the message code only.
OFF	Suppresses the entire message text and message code.
ON	Displays the entire message text and the message code.
TEXT	Displays the message text only.

Refer to the applicable *CP Command Reference for General Users* for details about the CP SET command.

When you specify the TERM assembler option, diagnostic messages are written to the terminal in the form ASMA*nnns*. Errors detected by the ASMAHL command program, which terminate the command before High Level Assembler is called, result in error messages (type E).

Chapter 11. Running your Program under CMS

There are three ways to run your assembler program under any level of CMS:

- Using the CMS LOAD and START commands
- Using the CMS GENMOD command to create a program module and then using the module filename to cause the module to be run
- Using the CMS LKED and OSRUN commands

Any of these three methods can be used under the control of the CMS batch facility.

Using the CMS LOAD and START Commands

After you have assembled your program, you can run the object program in the TEXT file produced by the assembler. The TEXT file produced is relocatable and can be run merely by loading it into virtual storage with the LOAD command and using the START command to begin running. For example, if you have assembled a source program named CREATE, you have a file named CREATE TEXT. Use the LOAD command to load your program into storage, and then use the START command to run the program:

```
LOAD CREATE
START
```

In this example, the file CREATE TEXT contains the object code from the assembly.

The CMS START command can be used to pass user-defined parameters. For a complete description of the START command, see the applicable *CMS Command Reference* for your VM environment, as listed under “Related Publications (VM)” on page 332.

Using the CMS GENMOD Command

When your programs are debugged and tested, you can use the LOAD and INCLUDE commands, in conjunction with the GENMOD command, to create program modules. A module is a relocatable or non-relocatable object program whose external references have been resolved. In CMS, these files must have a filetype of MODULE.

To create a program module, load the TEXT files or TXTLIB members into storage and issue the GENMOD command:

```
LOAD CREATE ANALYZE PRINT
GENMOD PROCESS
```

In this example, CREATE, ANALYZE, and PRINT are TEXT files that you are combining into a module named PROCESS; PROCESS is the filename you are assigning the module, which has a filetype of MODULE. If you use the name of an existing MODULE file, the old one is replaced.

From then on, any time you want to run the program composed of the object files CREATE, ANALYZE, and PRINT, enter:

```
PROCESS
```

If PROCESS requires input files, output files, or both, you must define these files before PROCESS can run correctly.

For more information on creating program modules, see the applicable *CMS User's Guide* for your particular VM environment, as listed under “Related Publications (VM)” on page 332.

Using the CMS LKED and OSRUN Commands

A LOADLIB is another type of library available to you under CMS. LOADLIBs, like MACLIBs and TXTLIBs, are in CMS-simulated partitioned dataset formats. Unlike TXTLIBs, which contain object programs that need to be link-edited when they are loaded, LOADLIBs contain programs that have already been link-edited, thus saving the overhead of the link-editing process every time the program is loaded. You can load the members of TXTLIBs by both CMS loading facilities (LOAD or INCLUDE command) and certain OS macros (such as LINK, LOAD, ATTACH, or XCTL), but you can only load the members of LOADLIBs that use these OS macros.

Use the LKED command to create a CMS LOADLIB. For example:

```
FILEDEF SYSLIB DISK USERTXT TXTLIB *
LKED TESTFILE
```

This example takes a CMS TEXT file with the filename of TESTFILE and creates a file named TESTFILE LOADLIB, using the SYSLIB to resolve external references. TESTFILE LOADLIB is a CMS-simulated partitioned data set containing one member named TESTFILE.

To use the OSRUN command to run TESTFILE, first use the GLOBAL command to identify which libraries are to be searched when processing subsequent CMS commands. For example:

```
GLOBAL LOADLIB TESTFILE
OSRUN TESTFILE
```

The OSRUN command causes the TESTFILE member of the TESTFILE LOADLIB to be loaded, relocated, and run.

User parameters can be added on the line with the OSRUN command, but they are passed in OS format. For a complete description of the OSRUN command, see the applicable *CMS Command Reference* for your particular VM environment, as listed under “Related Publications (VM)” on page 332.

Using the CMS Batch Facility

The CMS batch facility provides a way of submitting jobs for batch processing in CMS, and can be used to run an assembler program. You can use this facility when:

- You have a job that takes a lot of time, and you want to be able to use your terminal for other work while the job is running.
- You do not have access to a terminal.

The CMS batch facility is really a virtual machine, generated and controlled by the system operator, who logs onto VM using the batch user ID and invokes the CMSBATCH command. All jobs submitted for batch processing are spooled to the user ID of this virtual machine, which runs the jobs sequentially. To use the CMS batch facility at your location, you must contact the system operator to learn the user ID of the batch virtual machine.

You can run High Level Assembler under the control of the CMS batch facility. Terminal input can be read from the console stack. In order to prevent your batch job from being cancelled, make sure that stacked input is available if your program requests input from the terminal. For further information on using the CMS batch facility, see the applicable *CMS User's Guide* for your particular VM environment, as listed under “Related Publications (VM)” on page 332.

Chapter 12. CMS System Services and Programming Considerations

This chapter describes some of the CMS system services and program development facilities that assist you in developing your assembler program. It provides the following information:

- Assembler macros supported by CMS
- Adding definitions to a macro library
- Saving and restoring general register contents
- Ending program execution
- Passing parameters to your assembler language program

Using Macros

Assembler Macros Supported by CMS

There are several CMS assembler macros you can use in assembler programs. Among the services provided by these macros are the ability to write a record to disk, to read a record from disk, to write lines to a virtual printer, and so on. All the CMS assembler macros are described in the applicable *CMS Command and Macro Reference*, listed under "Related Publications (VM)" on page 332.

Adding Definitions to a Macro Library

Macro definitions, and members containing assembler source statements that can be read by a COPY instruction, can be added to a macro library. Use the CMS MACLIB command to create and modify CMS macro libraries. In the following example, a macro with a filename of NEWMAC and filetype of MACRO is added to the MACLIB with a filename of MYLIB.

```
MACLIB ADD MYLIB NEWMAC
```

Details of this command are described in the applicable *CMS Command and Macro Reference*, listed under "Related Publications (VM)" on page 332.

Operating System Programming Conventions

Assembler programs executing under CMS must follow a set of programming conventions to save and restore registers, and access execution parameters. These conventions are described in the following sections.

Saving and Restoring General Register Contents

A program should save the values contained in the general registers when it receives control and, upon completion, restore to the general registers these same values. Thus, as control is passed from the operating system to a program and, in turn, to a subprogram, the status of the registers used by each program is preserved. This is done through use of the SAVE and RETURN system macro instructions.

Saving Register Contents: The SAVE macro instruction should be the first statement in the program. It stores the contents of registers 14, 15, and 0 through 12 in an area provided by the program that passes control. When a program is given control, register 13 contains the address of an area in which the general register contents should be saved.

If the program calls any subprograms, or uses any operating system services other than GETMAIN, FREEMAIN, ATTACH, and XCTL, it must first save the contents of register 13 and then load the address of an 18-fullword save area into register 13. This save area is in the program and is used by any subprograms or operating system services called by the program.

Restoring Register Contents: At completion, the program restores the contents of general registers 14, 15, and 0 through 12 by use of the RETURN system macro instruction (which also indicates program completion). The contents of register 13 must be restored before issuing the RETURN macro instruction.

Example: The coding sequence that follows shows the basic process of saving and restoring the contents of the registers. See the *VM/ESA CMS Application Development Reference for Assembler* for further information about the SAVE and RETURN macros.

```
Name      Operation      Operand

CSECTNAM  SAVE           (14,12)
          USING          CSECTNAM,15
          .
          .
          ST             13,SAVEAREA+4
          LA             13,SAVEAREA
          .
program function source statements
          .
          L              13,SAVEAREA+4
          RETURN         (14,12)
SAVEAREA  DC             18F'0'
          .
          .
          END
```

Ending Program Execution

You indicate completion of an assembler language source program by using the RETURN system macro instruction to pass control from the terminating program to the program that initiated it. The initiating program may be the operating system or, if a subprogram issued the RETURN, the program that called the subprogram.

In addition to indicating program completion and restoring register contents, the RETURN macro instruction may also pass a return code—a condition indicator that may be used by the program receiving control.

If the program returns to the operating system, the return code can be compared against the condition stated in the COND= parameter of the JOB or EXEC statement.

If return is to another program, the return code is available in general register 15, and may be used as required. Your program should restore register 13 before issuing the RETURN macro instruction.

The RETURN system macro instruction is discussed in detail in the *VM/ESA CMS Application Development Reference for Assembler*.

Passing Parameters to your Assembler Language Program

In CMS, you can pass parameters to an assembler language program by means of the START command. The statement below shows how to pass parameters to your program using the CMS START command:

```
START MYJOB PARM1 PARM2
```

The parameters must be no longer than 8 characters each, and must be separated by blanks.

CMS creates a list of the parameters that are passed to the program when it is run. The address of the parameters is passed in register 1. The parameter list for the command above is:

```
PLIST    DS    0D
         DC    CL8'MYJOB'
         DC    CL8'PARM1'
         DC    CL8'PARM2'
         DC    8X'FF'
```

where the list is terminated by hexadecimal FFs.

If your program is started using the CMS OSRUN command, the parameters are passed in the same way as described in “Accessing Execution Parameters (MVS)” on page 165.

If a module was created using the CMS GENMOD command and run using the MODULE name, the parameters are passed in extended parameter list format. The address of the parameter list is passed in register 0.

The format of the extended parameter list is:

Offset	**Field**
0	Address of command name
4	Address of beginning of options
8	Address of end of options
12	User word
16	Reserved

Appendixes

Appendix A. Earlier Assembler Compatibility and Migration

This section compares the High Level Assembler to the earlier assemblers, Assembler H Version 2 and DOS/VSE Assembler. This section can be used to determine the changes that might be required to migrate your assembler programs to High Level Assembler. This section also lists the new facilities that are available with High Level Assembler that you can use for new and existing assembler programs.

Comparison of Instruction Set and Assembler Instructions

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Instruction set			
S/370 instructions	Yes	Yes	Yes
XA instructions	No	Yes	Yes
ESA instructions	No	Yes	Yes
Vector instructions	No	Yes	Yes
DOS operation code table	No	No	The DOS operation code table is designed specifically for assembling programs previously assembled using the DOS/VSE assembler. Some machine instructions and assembler instructions are not included in this operation code table. See “OPTABLE” on page 51 for further details.
Data definition statements			
CCW	Yes	Yes	Yes
CCW0	No	Yes	Yes
CCW1	No	Yes	Yes
DC	Yes	Yes	Yes
DS	Yes	Yes	Yes
Symbols used in the DC or DS expression need not be defined before they are used	No	Yes	Yes
Q-type Constant	No	Yes	Yes
S-type Constant	No	Yes	Yes
Number of nominal values for Binary and Hexadecimal constants	One	Multiple	Multiple
Program control statements			

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
ADATA	No	No	Yes
CNOP	Name entry can have sequence symbol or blank	Name entry can have any symbol or blank	Name entry can have any symbol or blank
COPY	Nesting depth limited to 3	Nesting depth not limited	Nesting depth not limited
EQU	Value operand only	Value, length attribute and type attribute operands.	Value, length attribute and type attribute operands.
END	END statement must be supplied.	Multiple END statements are allowed. If the END statement is omitted, the assembler generates an END statement.	Multiple END statements are allowed. If the END statement is omitted, the assembler generates an END statement.
EXITCTL	No	No	Yes
ICTL	Yes	Yes	Yes
ISEQ	Yes	Yes	Yes
LTORG	Yes	Yes	Yes
OPSYN	No	Yes	Yes
ORG	Name entry can have sequence symbol or blank	Name entry can have any symbol or blank	Name entry can have any symbol or blank
POP	No	Yes	Yes, with NOPRINT operand
PUNCH	Yes	Yes	Yes
PUSH	No	Yes	Yes, with NOPRINT operand
REPRO	Yes	Yes	Yes
Listing control statements			
CEJECT	No	No	Yes
EJECT	Yes	Yes	Yes
PRINT	Yes	Yes	Yes, with NOPRINT, MCALL, NOMCALL, MSOURCE, NOMSOURCE, UHEAD, and NOUHEAD operands
SPACE	Yes	Yes	Yes
TITLE	Up to 4 characters in name (if not a sequence symbol)	Up to 8 characters in name (if not a sequence symbol)	Up to 8 characters in name (if not a sequence symbol)
Base register assignment			
DROP	Yes	Yes	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
USING	Yes, ordinary USING	Yes, ordinary USING	Yes, ordinary, labeled, and dependent USINGs
Program sectioning and linking			
ALIAS	No	No	Yes
AMODE	No	Yes	Yes
CATTR	No	No	Yes
COM	Only unnamed common control sections are allowed	Yes	Yes
CSECT	Only named control sections are allowed	Yes	Yes
CXD	No	Yes	Yes
DSECT	Yes	Yes	Yes
DXD	No	Yes	Yes
ENTRY	The maximum number of symbols that can be identified by the ENTRY instruction is 200	Yes	Yes
EXTRN	Yes	Yes	Yes
RMODE	No	Yes	Yes
RSECT	No	Yes	Yes, with automatic checking for reenterability
START	Only named control sections are allowed	Yes	Yes
WXTRN	Yes	Yes	Yes

Comparison of Macro and Conditional Assembly Statements

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Macro definition			
MACRO	Yes	Yes	Yes
MEXIT	Yes	Yes	Yes
MEND	Yes	Yes	Yes
Conditional assembly			
ACTR	Yes	Yes	Yes
AEJECT	No	No	Yes
AGO	Yes	Yes	Yes
AIF	Yes	Yes	Yes
ANOP	Yes	Yes	Yes
AREAD	No	Yes	Yes, including CLOCKB and CLOCKD operands
ASPACE	No	No	Yes
GBLA	Yes	Yes	Yes
GBLB	Yes	Yes	Yes
GBLC	Yes	Yes	Yes
LCLA	Yes	Yes	Yes
LCLB	Yes	Yes	Yes
LCLC	Yes	Yes	Yes
MNOTE	Not allowed in open code	Allowed in open code	Allowed in open code
MHELP	No	Yes	Yes
SETA	Yes	Yes	Yes
SETB	Yes	Yes	Yes
SETC	Yes	Yes	Yes
SETAF	No	No	Yes
SETCF	No	No	Yes
System variable symbols			
&SYSLIST	Yes	Yes	Yes
&SYSNDX	Up to maximum of 9999	Up to maximum of 9999999	Up to maximum of 9999999
&SYSECT	Yes	Yes	Yes
&SYSLOC	No	Yes	Yes
&SYSTIME	No	Yes	Yes
&SYSDATE	No	Yes	Yes
&SYSASM	No	No	Yes
&SYSVER	No	No	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
&SYSDATC	No	No	Yes
&SYSJOB	No	No	Yes
&SYSSTEP	No	No	Yes
&SYSSTYP	No	No	Yes
&SYSSTMT	No	No	Yes
&SYSNEST	No	No	Yes
&SYSSEQF	No	No	Yes
&SYSOPT_DBCS	No	No	Yes
&SYSOPT_OPTABLE	No	No	Yes
&SYSOPT_RENT	No	No	Yes
&SYSTEM_ID	No	No	Yes
&SYSIN_DSN	No	No	Yes
&SYSIN_MEMBER	No	No	Yes
&SYSIN_VOLUME	No	No	Yes
&SYSLIB_DSN	No	No	Yes
&SYSLIB_MEMBER	No	No	Yes
&SYSLIB_VOLUME	No	No	Yes
&SYSPARM	Yes	Yes	Yes
&SYSPRINT_DSN	No	No	Yes
&SYSPRINT_MEMBER	No	No	Yes
&SYSPRINT_VOLUME	No	No	Yes
&SYSTERM_DSN	No	No	Yes
&SYSTERM_MEMBER	No	No	Yes
&SYSTERM_VOLUME	No	No	Yes
&SYSPUNCH_DSN	No	No	Yes
&SYSPUNCH_MEMBER	No	No	Yes
&SYSPUNCH_VOLUME	No	No	Yes
&SYSLIN_DSN	No	No	Yes
&SYSLIN_MEMBER	No	No	Yes
&SYSLIN_VOLUME	No	No	Yes
&SYSADATA_DSN	No	No	Yes
&SYSADATA_MEMBER	No	No	Yes
&SYSADATA_VOLUME	No	No	Yes
Symbol attributes			
Defined attribute	No	Yes	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Type attribute	An ordinary symbol outside a macro cannot be used as an operand of the T' inside a macro and cannot be used to determine the type of a SETA or SETB variable. Only allowed in conditional assembly instructions and not allowed for literals.	Yes; only allowed in conditional assembly instructions and not allowed for literals.	Yes; allowed in conditional assembly, assembler, and machine instructions and are allowed for previously defined literals.
Length attribute	Yes; allowed in conditional assembly, assembler, and machine instructions and not allowed for literals.	Yes; allowed in conditional assembly, assembler, and machine instructions and not allowed for literals.	Yes; allowed in conditional assembly, assembler, and machine instructions and are allowed for previously defined literals.
Scaling attribute	Yes; only allowed in conditional assembly instructions and not allowed for literals.	Yes; only allowed in conditional assembly instructions and not allowed for literals.	Yes; allowed in conditional assembly, assembler, and machine instructions and are allowed for previously defined literals.
Integer attribute	Yes; only allowed in conditional assembly instructions and not allowed for literals.	Yes; only allowed in conditional assembly instructions and not allowed for literals.	Yes; allowed in conditional assembly, assembler, and machine instructions and are allowed for previously defined literals.
Count attribute	Can only be used to determine the length of a macro instruction operand.	Yes	Yes
Number attribute	Yes	Can be applied to SETx variables	Can be applied to SETx variables
Operation Code Data attribute	No	No	Yes
Type and Count attribute for system variable symbols	No	Yes	Yes
Type attribute for SETA symbols that are defined via LCLA or GBLA but are not set (via SETA)	Not applicable	Returns a value of '00'	Returns a value of 'N'
Type attribute for SETB symbols that are defined via LCLB or GBLB but are not set (via SETB)	Not applicable	Issues an error message	Returns a value of 'N'

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Type attribute for macro instruction operands with a value of a previously used literal	Not applicable	Returns a value of 'U'	Returns the Type attribute of the constant defined by the literal

Comparison of Macro and Conditional Assembly

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
External Function calls using high level programming language	No	No	Yes
Built-In Functions for SETA, SETB, and SETC expressions	No	No	Yes
Substring length value The second subscript value of the substring notation can be specified as an (*).	No	No	Yes
Library macros in source format	No, library macros must be stored in edited format	Yes	Yes
Macro definitions can appear anywhere in your source module.	No, they must be at the start of the source file.	Yes	Yes
Editing macro definitions Use conditional assembly statement to avoid editing of macros.	No	Yes	Yes
Redefining macros A macro definition can be redefined at any point in the source code.	No	Yes	Yes
Nesting macro definitions Allow both inner macro instructions and inner macro definitions.	No	Yes	Yes
Generated macro instruction operation codes Macro instruction operation codes can be generated by substitution.	No	Yes	Yes
Multilevel sublists in macro instruction operands Multilevel sublists (sublists within sublists) are permitted in macro instruction operands and in keyword default values in prototype statements.	No	Yes	Yes
DBCS language support Double-byte data is supported by the macro language.	No	Yes	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Macro names, variable symbols (including the ampersand) and sequence symbols (including the period) can be up to a maximum of 63 characters.	No, limited to 8 characters.	Yes	Yes
Comments (both ordinary comments beginning with '*' and internal macro comments beginning with '.*') can be inserted between the macro header and the prototype and, for library macros, before the macro header.	No	Yes	Yes
Any mnemonic operation code of the Universal character set, or any assembler operation code, can be defined as a macro instruction.	No	Yes	Yes
Any instruction, except ICTL, is permitted within a macro definition.	No	Yes	Yes
AIF statements The AIF statement can include a string of logical expressions and related sequence symbols.	No	Yes	Yes
AGO statements The AGO statement can contain computed branch sequence information.	No	Yes	Yes
SETx statements The SETA, SETB and SETC statements can assign lists or arrays of values to subscripted SET symbols.	No	Yes	Yes
SET symbol format and definition changes • Either a macro definition or open code can contain more than one declaration for a given SET symbol, as long as only one is encountered during a given macro expansion or conditional assembly. • A SET symbol that has not been declared in a LCLx or GBLx statement is implicitly declared by appearing in the name field of a SETx statement. • A SET symbol can be defined as an array of values by adding a subscript after it, when it is declared, either explicitly or implicitly.	No	Yes	Yes
Created SET symbols SET symbols may be created during the generation of a macro.	No	Yes	Yes
Using SETC variables in arithmetic expressions You can use a SETC variable as an arithmetic term if its character string value represents a valid self-defining term.	No	Yes	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Forward attribute references If an attribute reference is made to a symbol that has not yet been encountered, the assembler scans the source code either until it finds the referenced symbol in the name field of a statement in open code, or until it reaches the end of the source module.	No	Yes	Yes
Attribute reference using SETC variables You can take an attribute reference for a symbol specified as: • The name of the ordinary symbol itself • The name of a symbolic parameter whose value is name of the ordinary symbol • The name of a SETC symbol whose value is the name of the ordinary symbol	No	Yes	Yes
Number attributes for SET symbols The number attribute can be applied to SETx variables to determine the highest subscript value of a SET symbol array to which a value has been assigned in a SETx instruction.	No	Yes	Yes
Alternate format in conditional assembly The alternate format allows a group of operands to be spread over several lines of code.	No	Yes	Yes
Maximum number of symbolic parameters and macro instruction operands	200	No fixed maximum	No fixed maximum
Mixing positional and keyword symbolic parameters and macro instruction operands	All positional parameters or operands must come first.	Keyword parameters or operands can be interspersed among positional parameters or operands.	Keyword parameters or operands can be interspersed among positional parameters or operands.

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
SET symbol declaration	Declaration of local symbols must immediately precede declaration of global symbols. Declaration of global and local symbols must immediately follow prototype statement if in macro definition. Declaration of global and local symbols must immediately follow source macro definitions, if in open code.	Declaration of local and global symbols can be mixed. Declaration of global and local symbols does not need to immediately follow prototype statement if in macro definition. Declaration of global and local symbols does not need to immediately follow source macro definitions, if in open code.	Declaration of local and global symbols can be mixed. Declaration of global and local symbols does not need to immediately follow prototype statement if in macro definition. Declaration of global and local symbols does not need to immediately follow source macro definitions, if in open code.
Maximum dimension for subscripted SET Symbols	4095	Not limited	Not limited
Duplication factor allowed in SETC instruction	No	Yes	Yes
Dynamically extended variable SET symbols	No	Yes	Yes
Number of terms in arithmetic expressions in conditional assembly	Up to 16	Not limited	Not limited
Levels of parentheses in arithmetic expressions in conditional assembly	Up to 5	Not limited	Not limited
MNOTE with error in macro is flagged at each invocation	Yes	No	No
Blank lines treated as equivalent to ASPACE 1.	No	No	Yes
Name entry of macro instruction must be a valid symbol	Yes	Yes	No
Ampersand preceding the SET symbols being declared is optional	No	No	Yes
Predefined absolute symbols allowed in arithmetic expression	No	No	Yes
Predefined absolute symbols allowed in SETx instruction	No	No	Yes
Type attribute of CNOP Label is set to 'I'	No, set to 'J'	No, set to 'J'	Yes
Type, length, scaling and integer attribute allowed for ordinary symbols, SETC symbols and literals in open code	No	No	Yes
Sublists assigned to SETC symbols can be passed to macro definitions and be recognised as sublists	No	No	Yes

Comparison of Language Features

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Macro comment statements allowed in open code	No	No	Yes
EQU instruction extension Symbols appearing in the first operand of the EQU instruction need not have been previously defined.	No	Yes	Yes
CNOP instruction extension There is no restriction that symbols in the operand field of a CNOP instruction must have been previously defined.	No	Yes	Yes
COPY instruction extension Any number of 'nestings', COPY statements within code that have been brought into your program by another COPY statement, is permitted.	No, nesting depth limited to 3	Yes	Yes

COPY instruction processed immediately
 	
COPY members are read immediately after a COPY statement is encountered in the source, regardless of whether or not conditional processing requires it, as in the following example:
 	
`       AGO    .LABEL`
`       COPY   AFILE`
`.LABEL ANOP`
	No, AFILE is never opened, read from, or processed in any way.	Yes, AFILE is scanned during lookahead processing	Yes, AFILE is scanned during lookahead processing

COPY instruction operand can, in open code, be specified as a variable symbol.	No	No	Yes
ISEQ instruction extension Sequence checking of any column on input records is allowed.	No	Yes	Yes
Macro names Inline macro names may contain the underscore character (_).	No	Yes	Yes
Continuation lines	Up to 2	Up to 9	Up to 9
Continuation lines and double-byte data	No	Yes	Yes
Symbol name length up to 63 characters	No, limited to 8	Yes	Yes
Levels within expressions Any number of terms or levels of parenthesis is an expression is allowed.	No	Yes	Yes
Underscores in symbols You can specify the underscore character (_) in ordinary symbols and variable symbols.	No	Yes	Yes
Underscore character accepted in any position in symbol name	No	No	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Underscore character accepted in external symbols	No	No	Yes
Underscore character accepted in name field of OPSYN instruction	No	No	Yes
Maximum number of external symbols	511	65 535	65 535
DBCS language support Pure double-byte data, and double-byte data mixed with single-byte data is permitted.	No	Yes	Yes
Location counter value printed for EQU, USING, ORG (in ADDR2 field)	3 bytes	4 bytes (up to 3 leading zeroes suppressed).	4 bytes (up to 3 leading zeroes suppressed).
Self-defining term			
Maximum value	2^{24}-1	2^{31}-1	2^{31}-1
Number of digits			
Binary:	24	31	31
Decimal:	8	10	10
Hexadecimal:	6	8	8
Characters:	3	4	4
Relocatable and absolute expressions			
value carried:	Truncated to 24 bits	Truncated to 31 bits	Truncated to 31 bits
Number of operators:	15	Not limited	Not limited
Levels of parenthesis:	5	Not limited	Not limited
All control sections initiated by a CSECT start at location 0 in listing and object module.	Yes	No	No
Copy files read once	Copy files read when statement is found	Copy files read when macro is edited (only once)	Copy files read when macro is edited (only once)
Operand greater than 255 characters when SUBLIST	Error diagnostic with message and return code of 8	Error diagnostic with message and return code of 12	Error diagnostic with message and return code of 12
Remarks generated because of generated blanks in operand field	No	Yes	Yes
Blank lines treated as equivalent to SPACE 1.	No	No	Yes
Literals usable as relocatable terms in expressions	No	No	Yes
Literals usable in RX format instructions in which index register is used	No	No	Yes
Mixed case input	No	No	Yes
2-byte relocatable address constants	No	No	Yes
Multi-level PUSH supported e.g. PUSH USING,USING	No	Yes. Not documented.	No

Comparison of Assembler Options

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
*PROCESS statements	No	No	Selected assembler options can be specified in the source program on *PROCESS statements.
ADATA	No	No	Yes
ALIGN	Yes	Yes	Yes
ASA	No	No	Yes
BATCH	No	Yes	Yes
COMPAT	No	No	Yes
DBCS	No	Yes	Yes
DECK	Yes	Yes	Yes
DISK	No	Yes (CMS only)	Yes (CMS only)
DXREF	No	No	Yes
EDECK	Yes	No	No
ESD	Yes	Yes	Yes
EXIT	No	No	Yes
FLAG	No	Yes	FLAG(ALIGN), FLAG(CONT), FLAG(RECORD) and FLAG(SUBSTR) can be specified.
FOLD	No	No	Yes
LANGUAGE	No	No	Yes. Applies to messages and listing headings.
LIBMAC	No	No	Yes
LINECOUNT	Yes	Yes	Yes
LINK	Yes	No	No, see OBJECT option
LIST	Yes	Yes	LIST(121), LIST(133), and LIST(MAX) can be specified.
MCALL	Yes	No	No; PCONTROL(MCALL) can be specified.
MXREF	No	No	MXREF(SOURCE), MXREF(XREF), and MXREF(FULL) can be specified.

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
NOSEG	No	No	Yes (CMS only). See also SEG.
NUM	No	Yes (CMS only)	No
OBJECT	Yes	Yes	Yes
OPTABLE	No	No	Yes
PCONTROL	No	No	Yes
PESTOP	No	No	Yes
PRINT	No	Yes (CMS only)	Yes (CMS only)
PROFILE	No	No	Yes
RA2	No	No	Yes
RLD	Yes	Yes	Yes
SEG	No	No	Yes (CMS only). See also NOSEG.
SIZE	No	No	Yes
STMT	No	Yes (CMS Only)	No
SYSPARM	Yes	Yes	Yes
SXREF	Yes	Same as XREF(SHORT)	Same as XREF(SHORT)
TERM	No	Yes	TERM(WIDE) and TERM(NARROW) can be specified.
TEST	No	Yes	Yes
TRANSLATE	No	No	Yes
USING	No	No	Yes
XOBJECT	No	No	Yes
XREF	Same as XREF(LONG)	XREF(SHORT) or XREF(LONG)	XREF(SHORT), XREF(FULL), and XREF(UNREFS) can be specified.

Comparison of Assembler Listing

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Mixed case listing headings	No	No	Headings can be in mixed case English or uppercase English. See LANGUAGE assembler option.

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
National Language Support	No	No	Diagnostic messages in English, German, Japanese, and Spanish.
Option summary	At end of listing in Diagnostic and Statistics section.	At end of listing in Diagnostic and Statistics section.	At start of listing.
External symbol dictionary	Yes	Yes	Yes
Dummy section dictionary	Yes	No	See DSECT Cross Reference
Source and object program	Yes	Yes	Yes
Page-break handling	Limited logic	Limited logic	Improved page-break handling in conjunction with the EJECT, SPACE, and TITLE assembler instructions, to prevent unnecessary blank pages.
Optional 133-character wide format with extended addresses	No	No	Yes. Required for XOBJECT.
Control section headings	No	No	Show current control section type in fixed heading line for COM section, DSECT, and RSECT.
Heading date includes century	No	No	Yes
Active USING Summary	No	No	Yes
PRINT instruction with MCALL option	No	No	Yes
PRINT instruction with MSOURCE option	No	No	Yes
PRINT instruction with NOGEN option shows object code for first instruction generated, or the first 8 bytes of data generated, in the *object code* column.	No	No	Yes
PRINT, PUSH and POP instructions with NOPRINT option	No	No	Yes
Relocation dictionary	Yes	Yes	Yes
Ordinary symbol and literal cross reference	Yes	Yes	Yes
Cross reference includes modification and branch flags, USING and DROP flags, EXecute instruction flag, and relocatability-type column.	No	No	Yes
Unreferenced symbols defined in CSECTs	No	No	Yes
Macro and copy code source summary	No	No	Yes
Macro and copy code cross reference	No	No	Yes
DSECT cross reference	No	No	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
USING map	No	No	Yes
Diagnostic cross reference and assembler summary	Diagnostic and Statistics section including error diagnostic messages	Yes	Yes
Flagged statements with input dataset information	No	No	Yes, if FLAG(RECORD) assembler option specified
Print line with current PTF level of assembler	No	No	Yes
Print line showing operating system, jobname, stepname and procedure stepname of assembly job	No	No	Yes
Print lines showing file names (data set names), member and volume serial numbers of each of the input and output data sets	No	No	Yes
Print lines showing statistics for I/O exits	No	No	Yes
Print line showing the amount of storage in the buffer pool and the amount of storage required for an in-storage assembly	No	No	Yes
Record counts show the number of Work file reads and writes	No	No	Yes
Print line showing the return code of the assembly	No	No	Yes
Print lines showing assembly start and stop time, and processor time	No	No	Yes
Terminal output	No	Yes	Yes
Multiple consecutive blanks compressed to a single blank	No	No	Yes, when TERM(NARROW) specified.
One Line summary	No	No	Yes

Comparison of Diagnostic Features

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Error messages for conflicting assembler options When conflicting assembler options are specified, such as OBJECT with NOOBJECT, the assembler issues warning messages.	No	No	Yes
Diagnostic information message The FLAG(RECORD) assembler option causes message ASMA435I to be printed after the last diagnostic message for each statement in error. The message shows the statement relative record number and where the statement in error was read from.	No	No	Yes
Statement continuation errors The FLAG(CONT) assembler option instructs the assembler to issue diagnostic messages ASMA430W through ASMA433W when it suspects a continuation error in a macro call instruction.	No	No	Yes
Suppress alignment error messages The FLAG(ALIGN) assembler option instructs the assembler to issue diagnostic messages ASMA033W when an alignment error is detected. This message may be suppressed by specifying the FLAG(NOALIGN) assembler option.	No	No	Yes
Error messages Error messages are printed in the listing and a summary at the end lists a total of the errors and a table of their line numbers.	No	Yes	Yes
Diagnostic messages in macro assembly More descriptive diagnostic error messages are printed in macro generated text.	No	Yes	Yes
Sequence field in macro-generated text The sequence field (columns 73 through 80) of the generated statements contains the level of the macro call, a hyphen, and the first five characters of the macro-definition name.	No	Yes	Yes
Format of macro-generated text Wherever possible, a generated statement is printed in the same format as the corresponding macro definition (model) statement.	No	Yes	Yes

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Error messages for a library macro definition Format errors within a particular library macro definition are listed directly following the first call of that macro.	No	Yes	Yes
Error messages for source program macro definition Macro definitions contained in the source program are printed in the listing, provided the applicable PRINT options are in effect.	No	Yes	Yes
Error messages in macro-generated text Diagnostic messages in generated text generally include a description of the error, the recovery action, model statement number at which the error occurred, and a SET symbol name, parameter number, or a value associated with the error.	No	Yes	Yes
Macro Trace Facility (MHELP)	No	Yes	Yes

Other Assembler Differences

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
Object module			
DXD, CXD and Q-type constants produced	No	Yes	Yes
Named COMMON	No	Yes	Yes
Unnamed CSECTS (private code)	No	Yes	Yes
SYM records produced	No	Yes	Yes
Extended object format module generation	Not applicable	No	Yes. See XOBJECT assembler option.
Diagnostics			
Diagnostic messages issued	At end of assembly	At line where error occurred where possible.	At line where error occurred where possible.
Diagnostic dump	No	Produced at time of failure	Produced at time of failure
Error diagnostics messages in mixed case	No	No	Yes
Resources			
Work file	3 Work Files	1 Work File	1 Work File
Associated data file	No	No	Yes
QSAM Input/output	Not applicable	No	Yes
Input/Output user exits	No	No	Yes.

Element	DOS/VSE Assembler	Assembler H Version 2	High level Assembler
System-Determined Blocksize	Not applicable	No	Yes; supported in MVS/ESA only.
31-bit addressing	No	No	Yes; does not include I/O buffers.
Minimum virtual storage requirements	200K	200K	500K
Printer control characters	American National Standard	Machine	American National Standard or machine depending on ASA option

Appendix B. Cross-System Portability Considerations

This section describes the issues you must consider when you use High Level Assembler to assemble a program under one operating system and execute the resulting program under another operating system.

Using Extended Architecture Instructions

High Level Assembler supports assembly of programs using Extended Architecture instructions, Enterprise System Architecture instructions, and Vector instructions under all operating systems supported by High Level Assembler.

A generated object program using Extended Architecture (370-XA) instructions can only run on a 370-XA mode processor under an operating system that provides the necessary architecture support for the 370-XA instructions used.

Similarly, a generated object program using Enterprise Systems Architecture/370 (ESA/370) or Enterprise Systems Architecture/390 (ESA/390) instructions can only run on an applicable processor under an operating system that provides the necessary architecture support for the ESA/370 or ESA/390 instructions used.

Using System Macros

Many system macros have the same name under different systems but generate different object code and have different parameters. For example, the OPEN, CLOSE, GET, and PUT macros have the same name on MVS and VSE but generate different object code.

Where ever the assembler program uses system macros, the system macros for the target system must be used when the program is assembled.

For example, when the assembler program is to be run under VSE, the VSE system macros must be used, even if the program is assembled under CMS.

Ensure that the macros used during assembly are for the correct release of the operating system upon which the assembler program is to run.

Migrating Object Programs

The object module produced by High Level Assembler is portable across all the supported operating systems. Therefore, an assembler program may be assembled on any of the supported operating systems and run on any of the supported operating systems. For example, an assembler program may be assembled under CMS and run under VSE.

The object module is portable across the supported operating systems with the following restrictions.

- Where ever the assembler program uses system macros, the system macros for the target system must be used.
- The object module must be link-edited using the target system linkage editor.

- The assembler instructions included in the assembler program must be supported by the system linkage editor.

  The VSE linkage editor, prior to VSE/ESA Version 2 Release 1, does not support dummy external DSECTS. Therefore, to link-edit the assembler program under earlier VSE operating systems, the assembler program must not include any DXD or CXD statements or Q-type address constants.

- The TEST assembler option should only be used if the object module is to be link-edited under MVS. The TEST option cannot be specified with the XOBJECT assembler option, which produces the extended object format module.

- An extended object format module cannot be ported to a VSE or CMS environment.

The AMODE and RMODE assembler instructions indicate to the linkage editor the addressing mode and residency mode for the section. The addressing mode and residency mode are ignored by the linkage editor on systems that do not support 31-bit addressing.

The AMODE and RMODE assembler instructions have an effect on the addressing mode and residency mode only if the object module produced is link-edited using an XA or ESA linkage editor and run on a system that supports 31-bit addressing.

Appendix C. Object Deck Output

The object module is produced by High Level Assembler when the OBJECT or DECK assembler option is specified.

The object module consists of 80 byte records with 5 record types. The record types are:

ESD External symbol dictionary records describe the external symbols used in the program.

TXT Text records describe object code generated.

RLD Relocation dictionary records provide the information required to relocate address constants within the object module.

END End records terminate the object module and optionally provide the entry point.

SYM Symbol table records provide symbol information for TESTRAN or TSO TEST.

The assembler can also produce records via the PUNCH and REPRO assembler statements, whose contents and format are entirely determined by the program.

The following sections describe the format of each record type.

ESD Record Format

Columns	Contents
1	X'02'
2-4	ESD
5-10	Blank
11-12	Variable field count—number of bytes of information in variable field (columns 17-64)
13-14	Blank
15-16	ESDID of first SD, XD, CM, PC, ER, or WX in variable field
17-64	Variable field. One-to-three 16-byte items of the following format:

17-24, 33-40, 49-56	8 bytes—Name
25, 41, 57	1 byte —ESD type code; the hexadecimal value is: 00 SD 01 LD 02 ER 04 PC 05 CM 06 XD(PR) 0A WX
26-28, 42-44, 58-60	3 bytes—Address
29, 45, 61	1 byte —Alignment if XD —Blank if LD, ER, or WX —AMODE/RMODE flags if SD, PC, or CM. Figure 69 describes the AMODE and RMODE flag values.

Figure 69. AMODE/RMODE Flags

Bits	Value	Description
4	1	RSECT
5	0	RMODE 24
	1	RMODE ANY
6-7	00	AMODE 24
	01	AMODE 24
	10	AMODE 31
	11	AMODE ANY

30-32, 46-48, 62-64	3 bytes—Length, LDID, or blank
65-72	Blank
73-80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a nonblank name field. This name can be 1-to-8 characters. If the name is fewer than 8 characters or if there is no name, the remaining columns contain a record sequence number.

TEXT (TXT) Record Format

Columns	Contents
1	X'02'
2-4	TXT
5	Blank
6-8	Relative address of first instruction on record
9-10	Blank
11-12	Byte count—number of bytes in information field (columns 17-72)
13-14	Blank
15-16	ESDID
17-72	56-byte information field
73-80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1-to-8 characters. If the name is fewer than 8 characters or if there is no name, the remaining columns contain a record sequence number.

RLD Record Format

Columns	Contents
1	X'02'
2-4	RLD
5-10	Blank
11-12	Data field count—number of bytes of information in data field (columns 17-72)
13-16	Blank
17-72	Data field: **17-18** Relocation ESDID **19-20** Position ESDID **21** Flag byte **22-24** Absolute address to be relocated **25-72** Remaining RLD entries
73-80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1-to-8 characters or if there is no name, the remaining columns contain a record sequence number.

If the rightmost bit of the flag byte is set, the following RLD entry has the same relocation ESDID and position ESDID, and this information is not repeated; if the rightmost bit of the flag byte is not set, the next RLD entry has a different relocation ESDID or position ESDID, and both ESDIDs are recorded.

For example, if the RLD entries 1, 2, and 3 of the program listing contain the following information:

Entry	Position ESDID	Relocation ESDID	Flag	Address
1	02	04	0C	000100
2	02	04	0C	000104
3	03	01	0C	000800

then columns 17-72 of the RLD record would be:

```
        |          Entry 1          |   Entry 2   |          Entry 3          |
Column: |17 18 19 20 21 22 23 24|25 26 27 28|29 30 31 32 33 34 35 36|37──────►72
        |00|04|00|02|0D|00|01|00|0C|00|01|04|00|01|00|03|0C|00|08|00|
        | ESD Ids    ▲  Address |▲  Address | ESD Ids    ▲  Address | Blanks
        |            |          ||          |            |          |
        |           Flag        |Flag       |           Flag        |
        |           (Set)       |(not       |           (not        |
        |                       |set)       |           set)        |
```

END Record Format

Columns	Contents
1	X'02'
2-4	END
5	Blank
6-8	Entry address from operand of END record in source deck (blank if no operand)
9-14	Blank
15-16	ESDID of entry point (blank if no operand)
17-32	Blank
33	Number of IDR items that follow (EBCDIC 1 or EBCDIC 2)
34-52	Translator identification, version and release level (such as 0101), and date of the assembly (yyddd)
53-71	When present, they are the same format as columns 34-52
72	Blank
73-80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1-to-8 characters. If the name is fewer than 8 characters or if there is no name, the remaining columns contain a record sequence number.

TESTRAN (SYM) Record Format

If you request it, the assembler writes out symbolic information for TESTRAN concerning the assembled program ahead of all other object module records. The format of the record images for TESTRAN output follows:

Columns	Contents
1	X'02'
2-4	SYM
5-10	Blank
11-12	Variable field—number of bytes of text in variable field (columns 17-72)
13-16	Blank
17-72	Variable field (see below)
73-80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1-to-8 characters. If the name is fewer than 8 characters or if there is no name, the remaining columns contain a record sequence number.

The variable field (columns 17-72) contains up to 56 bytes of TESTRAN text. The items comprising the text are packed together; consequently, only the last record may contain less than 56 bytes of text in the variable field. The formats of a text record and an individual text item are shown in Figure 71 on page 216. The contents of the fields within an individual entry are as follows:

1. Organization (1 byte). The possible values are shown in Figure 70.

Figure 70 (Page 1 of 2). Organization Value Byte

Bits	Value	Description
0	0	Non-data type
	1	Data type
1-3 If non-data type	000	Space
	001	Control section
	010	Dummy control section
	011	Common
	100	Instruction
	101	CCW, CCW0, CCW1
1 If data type	0	No multiplicity
	1	Multiplicity (indicates presence of M Field)
2 If data type	0	Independent (not a packed or zoned decimal constant)
	1	Cluster (packed or zoned decimal constant)

Figure 70 (Page 2 of 2). Organization Value Byte

Bits	Value	Description
3 If data type	0	No scaling
	1	Scaling (indicates presence of S field)
4	0	Name present
	1	Name not present
5-7		Length of name minus 1

2. Address (3 bytes)—displacement from base of control section
3. Symbol Name (0-8 bytes)—symbolic name of particular item. If the entry is nondata type and space, an extra byte is present that contains the number of bytes that have been skipped.
4. Data Type (1 byte)—contents in hexadecimal

   00 = character
   04 = hexadecimal or pure DBCS (G-type)
   08 = binary
   10 = fixed point, full
   14 = fixed point, half
   18 = floating point, short
   1C = floating point, long
   20 = A-type or Q-type data
   24 = Y-type data
   28 = S-type data
   2C = V-type data
   30 = packed decimal
   34 = zoned decimal
   38 = floating point, extended

5. Length (2 bytes for character, hexadecimal, decimal, or binary items; 1 byte for other types)—length of data item minus 1
6. Multiplicity–M field (3 bytes)—equals 1 if not present
7. Scale–signed integer–S field (2 bytes)—present only for F-, H-, E-, D-, P-, and Z-type data, and only if scale is nonzero.

1	2 4	5 10	11 12	13 16	17 72	73 80
X'02'	SYM	Blank	No. of bytes of text	Blank	TESTRAN Text - packed entries	Deck Id and Seq. Number
1	3	6	2	4	56	8

TESTRAN Text

Entry (Complete or end portion)	N Complete entries N >= 1	Entry (Complete or head portion)

Variable size entries

Entry

Org.	Address	Symbol name	Data Type	Length	Mult. Factor	Scale
1	3	0-8	1	1-2	3	2

Figure 71. TESTRAN SYM Record Format

Appendix D. Associated Data File Output

When you specify the ADATA assembler option, a file containing associated data is produced by the assembler. When you specify the ADATA suboption of the XOBJECT assembler option, ADATA records are written to the object data set as text records. You can specify both ADATA and XOBJECT(ADATA) to produce ADATA records in both the associated data file and the object data set. Information about the assembled program can be extracted from either data set, and used by debugging tools or cross reference tools. You don't need to extract this information from the assembler listing.

The associated data records are subject to change in future releases of High Level Assembler without prior notice. Any utility which processes associated data files should not process any files with architecture levels beyond those the utility has been designed and tested to process.

The ASMADATA macro maps the records in the associated data file, and the extended object format data set. The syntax and parameter keywords for this macro are shown on page 218.

ASMADATA

```
     ┌─,────────────────────────────┐
                        ┌─NOGEN─┐
►►──▼┬─PRINT=──┴─GEN───┴────┬─┴───►◄
     ├─AID=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─COMPUNIT=─┬─NO──┬────┤
     │           └─YES─┘    │
     ├─DCDS=──┬─NO──┬───────┤
     │        └─YES─┘       │
     ├─DCDSX=──┬─NO──┬──────┤
     │         └─YES─┘      │
     ├─ESD=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─JID=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─MACH=──┬─NO──┬───────┤
     │        └─YES─┘       │
     ├─MXREF=──┬─NO──┬──────┤
     │         └─YES─┘      │
     ├─MXREFX=──┬─NO──┬─────┤
     │          └─YES─┘     │
     ├─OPT=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─OUTPUT=──┬─NO──┬─────┤
     │          └─YES─┘     │
     ├─RLD=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─SOURCE=──┬─NO──┬─────┤
     │          └─YES─┘     │
     ├─SRCERR=──┬─NO──┬─────┤
     │          └─YES─┘     │
     ├─STATS=──┬─NO──┬──────┤
     │         └─YES─┘      │
     ├─SYM=──┬─NO──┬────────┤
     │       └─YES─┘        │
     ├─USER=──┬─NO──┬───────┤
     │        └─YES─┘       │
     ├─USING=──┬─NO──┬──────┤
     │         └─YES─┘      │
     └─XREF=──┬─NO──┬───────┘
              └─YES─┘
```

Default
PRINT=NOGEN,*keyword*=NO

NOGEN
Do not print the generated DSECTs in the listing

GEN
Print the generated DSECTs in the listing

NO
Do not generate a DSECT for this record

YES
Generate a DSECT for this record

keywords

AID	ADATA Identification DSECT (Type X'0001')

COMPUNIT ADATA Compilation Unit Start/End DSECT (Type X'0002')

DCDS DC/DS DSECT (Type X'0034')

DCDSX DC Extension DSECT(Type X'0035')

ESD External Symbol Dictionary (ESD) DSECT (Type X'0020')

JID Job Identification DSECT (Type X'0000')

MACH Machine Instruction DSECT (Type X'0036')

MXREF Macro and Copy Code Source Summary DSECT (Type X'0060')

MXREFX Macro and Copy Code Cross Reference DSECT (Type X'0062')

OPT Options DSECT (Type X'0010')

OUTPUT Output File DSECT (Type X'000A')

RLD Relocation Dictionary (RLD) DSECT (Type X'0040')

SOURCE Source Analysis DSECT (Type X'0030')

SRCERR Source Error DSECT (Type X'0032')

STATS Statistics DSECT (Type X'0090')

SYM Symbol DSECT (Type X'0042')

USER User Data Record DSECT (Type X'0070')

USING Using Map DSECT (Type X'0080')

XREF Symbol Cross Reference DSECT (Type X'0044')

Record Types

The file contains records classified into different record types. Each type of record provides information about the assembler language program being assembled. Each record consists of two parts:

- An 8-byte header section, which has the same structure for all record types
- A variable-length data section, which varies by record type.

The header section contains, among other items, the record code which identifies the type of record.

The record types, and their contents, written to the associated data file are:

Job Identification X'0000'
Provides information about the assembly job, the host system environment, and the names of the primary input data sets.

ADATA Identification X'0001'
Provides a precise time stamp, and a description of the character set used for character data in the file.

The time stamp is represented as Universal Time (UT) with the low-order bit representing 1 microsecond.

ADATA Compilation Unit Start/End X'0002'
Indicates where the associated data records for each assembly unit begin and end. The START record is written to the associated data file at the beginning of each assembly. The END record is written to the associated data file at the end of each assembly. The last record written to the The END record contains a count of the total number of records written to the associated data file.

When there are multiple assembler programs in the input file, there is a START and END record for each program assembled.

Output File X'000A'
Provides information about all the assembler output files used for the assembly.

Options X'0010'
Describes the assembler options used for the assembly.

External Symbol Dictionary X'0020'
Describes all the control sections, including DSECTs, defined in the program.

Source Analysis X'0030'
Describes a single source line.

There is one source analysis record in the file for each source record which would appear in the listing as if PRINT ON,GEN was active. This includes those source records generated by macro instructions, or included by COPY instructions. A source analysis record is also produced for TITLE statements. The FOLD assembler option does not cause the source in the source analysis record to be converted to uppercase.

The source analysis records appear in the sequence they would appear in the listing. Conditional assembly statements might cause the source statements to be skipped or the sequence of the records to be altered.

Source Error X'0032'
Describes errors in source program statements.

All Source Error records follow the source analysis record to which they apply.

DC/DS X'0034'
Describes the constant or storage defined by a source program statement that contains a DC, DS, CXD, DXD, CCW, CCW0, or CCW1 instruction.

If a source program statement contains one of the above then a DC/DS record is written following the source analysis record.

If there is an error in the DC, DS, CXD, DXD, CCW, CCW0, or CCW1 instruction, the DC/DS record is not produced.

If the DC statement has a duplication factor greater than 1, and at least one of the operand values has a reference to the current location counter (*****), then a DC extension record (`X'0035'`) is generated.

DC Extension X'0035'
This record describes the object text generated by a DC statement when the DC statement has repeating fields. This record is only created if the

DC statement has a duplication factor greater than 1 and at least one of the operand values has a reference to the current location counter (*).

Machine Instruction X'0036'
Describes the object code generated for a source program statement.

If a source program statement causes machine instructions to be generated, then a machine instruction record is written following the Source record. If there is an error in the machine instruction, the machine instruction record follows the source error record.

Relocation Dictionary X'0040'
Describes the relocation dictionary information that is contained in the object module RLD records.

Symbol X'0042'
Describes a single symbol defined in the program.

There is one Symbol record for each symbol defined in the program, including literals.

Symbol and Literal Cross Reference X'0044'
Describes the references to a single symbol.

All Symbol and Literal Cross Reference records follow the Symbol record to which they apply.

Macro and Copy Code Source Summary X'0060'
Describes the source of each macro and copy code member retrieved by the program.

Macro and Copy Code Cross Reference X'0062'
Describes the references to a single macro, or member copied by the COPY assembler instruction.

User Data X'0070'
Describes the data written by the ADATA assembler instruction.

Using Map X'0080'
Describes all USING, DROP, PUSH USING, and POP USING statements in the program.

Statistics X'0090'
Describes the statistics about the assembly.

Figure 72 on page 222 shows part of the listing of an assembler program. If this assembler program were assembled with the ADATA option, the records produced in the associated data file would be in the sequence shown below.

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                      HLASM R2.0  1995/03/26 08.00

000000                               1 CSECTNAM CSECT
000000 90EC D00C            0000C     2          STM   14,12,12(13)
                  R:F 00000           3          USING CSECTNAM,15
000004 0000 0000            00000     4          A     2,FIELD3
   ASMA044E *** ERROR *** UNDEFINED SYMBOL - FIELD3
000008 98EC C00C            0000C     5          LM    14,12,12(12)
00000C 07FE                           6          BR    14
                                      7          DROP  15
                                      8          COPY  ADATA
00000E                                9 FIELD1   DS    CL8
000016                               10 FIELD2   DS    CL8
                                     11          END
```

Figure 72. Sample Assembler Program for Associated Data Output

Type	Description
X'0001'	ADATA Identification record
X'0002'	ADATA Compilation Unit START record
X'0000'	Job Identification record
X'000A'	Output File record
X'0010'	Options record
X'0020'	External Symbol Dictionary record for CSECTNAM
X'0030'	Source record for statement 1 `CSECTNAM CSECT`
X'0030'	Source record for statement 2 `STM   14,12,12(13)`
X'0036'	Machine Instruction record for STM instruction
X'0030'	Source record for statement 3 `USING CSECTNAM,15`
X'0030'	Source record for statement 4 `A     2,FIELD3`
X'0032'	Source Error record for message ASMA044E
X'0036'	Machine Instruction record for A instruction
X'0030'	Source record for statement 5 `LM    14,12,12(12)`
X'0036'	Machine Instruction record for LM instruction
X'0030'	Source record for statement 6 `BR    14`
X'0036'	Machine Instruction record for BR instruction
X'0030'	Source record for statement 7 `DROP  15`
X'0030'	Source record for statement 8 `COPY  ADATA`
X'0030'	Source record for statement 9 (From COPY member ADATA) `FIELD1   DS    CL8`
X'0034'	DC/DS record for FIELD1
X'0030'	Source record for statement 10 (From COPY member ADATA) `FIELD2   DS    CL8`
X'0034'	DC/DS record for FIELD2
X'0030'	Source record for statement 11 `END`
X'0042'	Symbol record for CSECTNAM
X'0044'	Symbol and Literal Cross-Reference record for CSECTNAM
X'0042'	Symbol record for FIELD1
X'0044'	Symbol and Literal Cross-Reference record for FIELD1

X'0042' Symbol record for FIELD2
X'0044' Symbol and Literal Cross-Reference record for FIELD2
X'0042' Symbol record for FIELD3
X'0044' Symbol and Literal Cross-Reference record for FIELD3
X'0060' Macro and Copy Code Source Summary record for COPY ADATA
X'0062' Macro and Copy Code Cross Reference record for COPY ADATA
X'0080' USING Map record for USING on statement 3
X'0080' USING Map record for DROP on statement 7
X'0090' Assembly Statistics record
X'0002' ADATA Compilation Unit END record
The count value in this record is 38.

Macro-only Assemblies

The associated data file can also be useful for assemblies that have macro processing only (SYSGENs for example). The printing of the generated assembler source is not printed in the listing, but the information is available in the associated data file. Figure 73 shows part of the listing of an assembler program that only includes a macro instruction. The statements generated by the macro instruction (statements 9 through 11) are not printed on the listing. If this program were assembled with the ADATA option, the records produced in the associated data file would be in the sequence shown below.

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT        HLASM R2.0  1995/03/26 11.00

                                      1          print nogen
                                      2          macro
                                      3 &NAME    testhla &job
                                      4          punch '//&job JOB'
                                      5          punch '//STEP1   EXEC PGM=ABC'
                                      6          punch '//DDNAME1 DD   DSN=DSN.&job.,DISP=SHR'
                                      7          mend
                                      8          TESTHLA TESTJOB
                                     12          END
```

Figure 73. Sample Assembler Program for Macro only Assembly

Type	Description
X'0001'	ADATA Identification record
X'0002'	ADATA Compilation Unit START record
X'0000'	Job Identification record
X'000A'	Output File record
X'0010'	Options record
X'0030'	Source record for statement 1 `         print nogen`
X'0030'	Source record for statement 2 `         macro`
X'0030'	Source record for statement 3 `&NAME    testhla &job`
X'0030'	Source record for statement 4 `         punch '//&job JOB'`
X'0030'	Source record for statement 5 `         punch '//STEP1   EXEC PGM=ABC'`
X'0030'	Source record for statement 6 `         punch '//DDNAME1 DD DSN=DSN.&job.,DISP=SHR'`
X'0030'	Source record for statement 7 `         mend`

X'0030' Source record for statement 8

```
TESTHLA TESTJOB
```

X'0030' Source record for statement 9

```
punch '//TESTJOB JOB'
```

X'0030' Source record for statement 10

```
punch '//STEP1   EXEC PGM=ABC'
```

X'0030' Source record for statement 11

```
punch '//DDNAME1 DD DSN=DSN.TESTJOB,DISP=SHR'
```

X'0030' Source record for statement 12

```
END
```

X'0060' Macro and Copy Code Source Summary record for macro TESTHLA
X'0062' Macro and Copy Code Cross Reference record for macro TESTHLA
X'0090' Assembly Statistics record
X'0002' ADATA Compilation Unit END record
The count value in this record is 21.

ADATA Record Layouts

The formats of the records written to the associated data file are shown in the sections that follow.

In the fields described in each of the record types, a notation based on the assembler language data type is used:

C indicates EBCDIC data
H indicates 2-byte binary integer data
F indicates 4-byte binary integer data
A indicates 4-byte binary integer address and offset data
X indicates hexadecimal (bit) data

No boundary alignments are implied by any data type, and you can change the implied lengths by using a length indicator (L*n*). All integer data is in *System/370* format; that is bit 0 is always the most significant bit, bit *n* is the least significant bit, and the byte ordering in the records is from most significant to the least significant. The bits within a byte are numbered from left to right starting from 0.

Common Header Section

Each ADATA record contains an 8-byte common header section.

All ADATA records at the same architecture level have the same header section, which describes the producing language, the record type, the record architecture level (or version), a continued-record indicator, and, starting at level 2, an edition number.

High Level Assembler Release 2 produces architecture level 2 header records. This level is described in the following sections.

Figure 74 (Page 1 of 2). ADATA Record - Common Header Section

Field	Size	Description	
Language code	FL1	16	Assembler

Figure 74 (Page 2 of 2). ADATA Record - Common Header Section

Field	Size	Description
Record type	XL2	The record type, which can be one of the following: X'0000' Job Identification record X'0001' ADATA Identification record X'0002' Compilation Unit Start/End record X'000A' Output File Information record X'0010' Options record X'0020' External Symbol Dictionary record X'0030' Source Analysis record X'0032' Source Error record X'0034' DC/DS record X'0035' DC/DS Extension record X'0036' Machine Instruction record X'0040' Relocation Dictionary record X'0042' Symbol record X'0044' Symbol and Literal Cross-Reference record X'0060' Macro and Copy Code Source Summary record X'0062' Macro and Copy Code Cross Reference record X'0070' User Data record X'0080' USING Map record X'0090' Assembly Statistics record
Associated Data Architecture level	FL1	2
Flag	XL1	X'00' This record is not continued X'01' This record is continued on the next record All other values are reserved.
Edition Number	FL1	0
Reserved	CL2	

Note:

1. The mapping of the 8-byte header does not include the area used for the variable-length, record-descriptor word required by the access method.
2. The BATCH option, when used in conjunction with the ADATA option, produces a group of records for each assembly. Each group of records is delimited by the ADATA Compilation Start/End records.
3. All undefined and unused values are reserved.

Job Identification Record - X'0000'

Field	Size	Description
Date	CL8	The date of the assembly in the format YYYYMMDD
Time	CL4	The time of the assembly in the format HHMM
Product Number	CL8	The product number of the assembler that produced the associated data file
Product version	CL8	The version number of the assembler that produced the associated data file, in the form V.R.M and padded to the right with blanks. For example, `C'1.2.0   '`.
PTF level	CL8	The PTF level number of the assembler that produced the associated data file
System ID	CL24	The system identification of the system on which the assembly was run

Field	Size	Description
Jobname	CL8	The jobname of the assembly job
Stepname	CL8	The MVS stepname of the assembly step
Procstep	CL8	The MVS procedure step name of the assembly procedure step
Number of input files	HL2	The number of input files in this record. The following group of seven fields will occur 'n' times depending on the value in this field.
...Input file number	HL2	The assigned sequence number of the file
...Input file name length	HL2	The length of the following input file name
...Volume serial number length	HL2	The length of the volume serial number
...Member name length	HL2	The length of the member name
...Input file name	CL(n)	The name of the input file for the assembly
...Volume serial number	CL(n)	The volume serial number of the (first) volume on which the input file resides
...Member name	CL(n)	Where applicable, the name of the member in the input file

Note:

1. Where the number of input files would exceed the record size for the associated data file, the record is continued on the next record. The current number of input files (for that record) are stored in the record and the record written to the associated data file. The next record contains the rest of the input files. The count of the number of input files is a count for the current record.
2. If a SOURCE user exit has been specified for the assembly, and the SOURCE user exit has opened the input file, the input file details are those returned by the user exit.

ADATA Identification Record - X'0001'

Field	Size	Description
Time (binary)	XL8	Universal Time (UT) with the low-order bit representing 1 microsecond. This time may be used as a time-zone-independent time stamp.
CCSID	XL2	Coded Character Set IDentifier for any character data within the file

ADATA Compilation Unit Start/End Record - X'0002'

Field	Size	Description
Indicator	HL2	Start/End Indicator **X'0000'** Start of a group of compilation-unit-related ADATA records **X'0001'** End of a group of compilation-unit-related ADATA records All other values are reserved.
Reserved	CL2	

Field	Size	Description
Record Count	FL4	On an ADATA Compilation Unit End record, a count of all the ADATA records for this compilation unit. (On an ADATA Compilation Unit Start record, this field should be zero, unless the producing translator has foreknowledge of the exact number of records to be written, in which case it must be identical to the count in the Compilation Unit End record. Otherwise, it may be ignored by any consumer of the ADATA stream.) In High Level Assembler, the record count in the ADATA Compilation Unit Start record is always zero.

System 370/390 Output File Information Record - X'000A'

The Output File Information record provides data about the files produced by the translator.

This architecture level provides for five such output files:

1. The object data set produced when you specify the OBJECT or XOBJECT option
2. The object data set produced when you specify the DECK option
3. The listing file produced when you specify the LIST option
4. The terminal messages file produced when you specify the TERM option
5. The SYSADATA file produced when you specify the ADATA option

Field	Size	Description
Number of primary object-file (OBJECT) output files	HL2	The number of primary object files in this record. The groups of seven primary output-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Number of secondary object-file (PUNCH) output files	HL2	The number of secondary (punch) object files in this record. The groups of seven secondary output-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Number of listing (PRINT) output files	HL2	The number of listing (print) files in this record. The groups of seven listing-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Number of terminal (TERM) output files	HL2	The number of terminal output files in this record. The groups of seven terminal-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Number of SYSADATA (ADATA) output files	HL2	The number of ADATA output files in this record. The groups of seven associated data (ADATA) output-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
	XL10	Reserved
		Start of primary output-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Object-file primary output file number	HL2	The assigned sequence number of the file

Field	Size	Description
...Object file (primary) name length	HL2	The length of the following output file name for the primary object-file
...Volume serial number length	HL2	The length of the volume serial number for the primary object-file
...Member name length	HL2	The length of the member name for the primary object-file. If no member name is applicable, this field will contain binary zeros.
...Output (primary) file name	CL(n)	The name of the primary object output file for the compilation
...Volume serial number	CL(n)	The volume serial number of the volume on which the primary object output file resides
...Member name	CL(n)	Where applicable, the name of the member in the primary object output file.
		End of primary output-file information group.
		Start of secondary output-file information groups, one group per file.
...Object-file secondary output file number	HL2	The assigned sequence number of the secondary object-output file
...Output file name length	HL2	The length of the following secondary object-output file name
...Volume serial number length	HL2	The length of the volume serial number for the secondary object-output file
...Member name length	HL2	The length of the member name for the secondary object-output file. If no member name is applicable, this field contains binary zeros.
...Output (secondary) file name	CL(n)	The name of the secondary object output file for the compilation.
...Volume serial number	CL(n)	The volume serial number of the volume on which the secondary object output file resides.
...Member name	CL(n)	Where applicable, the name of the member in the output file.
		End of secondary output-file information group.
		Start of listing-file information groups, one group per file.
...Listing-file output file number	HL2	The assigned sequence number of the listing file
...Listing file name length	HL2	The length of the following listing file name
...Volume serial number length	HL2	The length of the volume serial number for the listing file
...Member name length	HL2	The length of the member name for the listing file. If no member name is applicable, this field contains binary zeros.
...Listing file name	CL(n)	The name of the listing output file for the compilation.
...Volume serial number	CL(n)	The volume serial number of the volume on which the listing file resides.
...Member name	CL(n)	Where applicable, the name of the member for the listing file.
		End of listing-file information group.

Field	Size	Description
		Start of terminal-file information groups, one group per file.
...Terminal file output file number	HL2	The assigned sequence number of the terminal file
...Terminal file name length	HL2	The length of the following terminal file name
...Volume serial number length	HL2	The length of the volume serial number for the terminal file
...Member name length	HL2	The length of the member name for the terminal file. If no member name is applicable, this field contains binary zeros.
...Terminal file name	CL(n)	The name of the terminal output file for the compilation.
...Volume serial number	CL(n)	The volume serial number of the volume on which the terminal file resides.
...Member name	CL(n)	Where applicable, the name of the member for the terminal file.
		End of terminal-file information group.
		Start of SYSADATA-file information groups, one group per file.
...ADATA file output file number	HL2	The assigned sequence number of the SYSADATA file
...ADATA file name length	HL2	The length of the SYSADATA file name
...Volume serial number length	HL2	The length of the volume serial number for the SYSADATA file
...Member name length	HL2	The length of the member name for the SYSADATA file. If no member name is applicable, this field contains binary zeros.
...ADATA file name	CL(n)	The name of the SYSADATA output file for the compilation.
...Volume serial number	CL(n)	The volume serial number of the volume on which the SYSADATA file resides.
...Member name	CL(n)	Where applicable, the name of the member for the SYSADATA file.
		End of SYSADATA-file information group.

Note:

If the number of output data sets causes the record to exceed the ADATA record size, the record is continued on the next record. The number of output files in the record is stored in the record, and the record is written to the ADATA file. The next record contains the rest of the output files. The count of the number of output files in the current record.

Options Record - X'0010'

This record indicates which assembler options were used for the assembly, and the values passed as suboptions. For example, if the PROFILE option was specified, bit 0 in option byte 8 would be 1, and the PROFILE_NAME field would contain the profile member name.

Field	Size	Description
Option Byte 1	XL1	1... .... Bit 1 = ALIGN, Bit 0 = NOALIGN .1.. .... Bit 1 = ASA, Bit 0 = NOASA ..1. .... Bit 1 = BATCH, Bit 0 = NOBATCH ...1 .... Bit 1 = COMPAT, Bit 0 = NOCOMPAT .... 1... Bit 1 = COMPAT(CASE), Bit 0 = not COMPAT(CASE) .... .1.. Bit 1 = COMPAT(SYSLIST), Bit 0 = not COMPAT(SYSLIST) .... ..1. Bit 1 = DBCS, Bit 0 = NODBCS ....1 Bit 1 = DECK, Bit 0 = NODECK
Option Byte 2	XL1	1... .... Bit 1 = DXREF, Bit 0 = NODXREF .1.. .... Bit 1 = ESD, Bit 0 = NOESD ..1. .... Bit 1 = FOLD, Bit 0 = NOFOLD ...1 .... Bit 1 = LIBMAC, Bit 0 = NOLIBMAC .... 1... Bit 1 = LIST, Bit 0 = NOLIST .... .1.. Bit 1 = ADATA, Bit 0 = NOADATA .... ..1. Bit 1 = MXREF or MXREF(FULL), Bit 0 = NOMXREF ....1 Bit 1 = OBJECT, Bit 0 = NOOBJECT
Option Byte 3	XL1	1... .... Bit 1 = PCONTROL, Bit 0 = NOPCONTROL .1.. .... Bit 1 = PCONTROL(ON), Bit 0 = not PCONTROL(ON) ..1. .... Bit 1 = PCONTROL(DATA), Bit 0 = not PCONTROL(DATA) ...1 .... Bit 1 = PCONTROL(GEN), Bit 0 = not PCONTROL(GEN) .... 1... Bit 1 = PCONTROL(UHEAD), Bit 0 = not PCONTROL(UHEAD) .... .1.. Bit 1 = PCONTROL(MSOURCE), Bit 0 = not PCONTROL(MSOURCE) .... ..1. Bit 1 = PCONTROL(MCALL), Bit 0 = not PCONTROL(MCALL) ....1 Bit 1 = COMPAT(MACROCASE), Bit 0 = not COMPAT(MACROCASE)
Option Byte 4	XL1	1... .... Bit 1 = RENT, Bit 0 = NORENT .1.. .... Bit 1 = RLD, Bit 0 = NORLD ..1. .... Bit 1 = TERM, Bit 0 = NOTERM ...1 .... Bit 1 = TEST, Bit 0 = NOTEST .... 1... Bit 1 = XREF, Bit 0 = NOXREF .... .1.. Bit 1 = XREF(FULL), Bit 0 = Not XREF(FULL) .... ..1. Reserved ....1 Bit 1 = XREF(SHORT), Bit 0 = not XREF(SHORT)
Option Byte 5	XL1	1... .... Bit 1 = EXIT, Bit 0 = NOEXIT .1.. .... Bit 1 = INEXIT, Bit 0 = NOINEXIT ..1. .... Bit 1 = LIBEXIT, Bit 0 = NOLIBEXIT ...1 .... Bit 1 = OBJEXIT, Bit 0 = NOOBJEXIT .... 1... Bit 1 = PRTEXIT, Bit 0 = NOPRTEXIT .... .1.. Bit 1 = ADEXIT, Bit 0 = NOADEXIT .... ..1. Bit 1 = TRMEXIT, Bit 0 = NOTRMEXIT ....1 Reserved

Field	Size	Description
Option Byte 6	XL1	1... .... Bit 1 = USING(WARN(m)), Bit 0 = USING(NOWARN) .1.. .... Bit 1 = USING(LIMIT(nnnn)), Bit 0 = USING(NOLIMIT) ..1. .... Bit 1 = USING(MAP), Bit 0 = USING(NOMAP) ...1 .... Bit 1 = FLAG(ALIGN), Bit 0 = FLAG(NOALIGN) .... 1... Bit 1 = FLAG(CONT), Bit 0 = FLAG(NOCONT) .... .1.. Bit 1 = FLAG(RECORD), Bit 0 = FLAG(NORECORD) .... ..1. Bit 1 = XOBJECT, Bit 0 = not XOBJECT ....1 Bit 1 = XOBJECT(ADATA), Bit 0 = XOBJECT(NOADATA)
Option Byte 7	XL1	1... .... Bit 1 = PESTOP, Bit 0 = NOPESTOP .1.. .... Bit 1 = RA2, Bit 0 = NORA2 ..1. .... Bit 1 = FLAG(SUBSTR), Bit 0 = FLAG(NOSUBSTR) ...1 .... Bit 1 = TRANSLATE(xx), Bit 0 = NOTRANSLATE .... 00.. Reserved .... 01.. LIST(121) .... 10.. LIST(133) .... 11.. LIST(MAX) .... ..00 Reserved .... ..01 MXREF(FULL) .... ..10 MXREF(SOURCE) .... ..11 MXREF(XREF)
Option Byte 8	XL1	1... .... Bit 1 = PROFILE, Bit 0 = NOPROFILE .1.. .... Bit 1 = PCONTROL(OFF), Bit 0 = not PCONTROL(OFF) ..1. .... Bit 1 = PCONTROL(NODATA), Bit 0 = not PCONTROL(NODATA) ...1 .... Bit 1 = PCONTROL(NOGEN), Bit 0 = not PCONTROL(NOGEN) .... 1... Bit 1 = PCONTROL(NOUHEAD), Bit 0 = not PCONTROL(NOUHEAD) .... .1.. Bit 1 = PCONTROL(NOMSOURCE), Bit 0 = not PCONTROL(NOMSOURCE) .... ..1. Bit 1 = PCONTROL(NOMCALL), Bit 0 = not PCONTROL(NOMCALL) ....1 Bit 1 = XREF(UNREFS), Bit 0 = not XREF(UNREFS)
Warn_Value	FL1	Value from USING(WARN(m))
Flag_Value	FL1	Value from Flag(n)
Reserved	CL2	Reserved
TRANS_SUFFIX	CL2	Value from TRANSLATE(xx). Blank if not provided.
PROFILE_NAME	CL8	Value from PROFILE(xxxxxxxx). Blank if not provided.
Limit_Value	HL2	Value from USING(LIMIT(nnnn))
LANGUAGE	CL3	Language option in effect for the assembly
OPTABLE	CL3	OPTABLE option in effect for the assembly
LINECOUNT	HL2	Linecount option in effect for the assembly
INEXIT_PROG_LEN	HL2	Length of INEXIT program name
LIBEXIT_PROG_LEN	HL2	Length of LIBEXIT program name
OBJEXIT_PROG_LEN	HL2	Length of OBJEXIT program name
PRTEXIT_PROG_LEN	HL2	Length of PRTEXIT program name
ADEXIT_PROG_LEN	HL2	Length of ADEXIT program name
TRMEXIT_PROG_LEN	HL2	Length of TRMEXIT program name
INEXIT_STR_LEN	HL2	Length of string supplied to exit
LIBEXIT_STR_LEN	HL2	Length of string supplied to exit
OBJEXIT_STR_LEN	HL2	Length of string supplied to exit

Field	Size	Description
PRTEXIT_STR_LEN	HL2	Length of string supplied to exit
ADEXIT_STR_LEN	HL2	Length of string supplied to exit
TRMEXIT_STR_LEN	HL2	Length of string supplied to exit
SYSPARM length	HL2	Length of the SYSPARM string supplied
PARMS length	HL2	Length of the PARM string supplied
Reserved	CL8	Reserved for future use
INEXIT_PROG	CL(n)	Input exit name
LIBEXIT_PROG	CL(n)	Library exit name
OBJEXIT_PROG	CL(n)	Object exit name
PRTEXIT_PROG	CL(n)	Print exit name
ADEXIT_PROG	CL(n)	ADATA exit name
TRMEXIT_PROG	CL(n)	Term exit name
INEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
LIBEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
OBJEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
PRTEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
ADEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
TRMEXIT_STR	CL(n)	Field to contain the string to be passed to the exit program
SYSPARM string	CL(n)	Field to contain the SYSPARM string that is being used for the assembly
PARM string	CL(n)	Field to contain the PARM string that is being used for the assembly

External Symbol Dictionary Record - X'0020'

Field	Size	Description
Section Type	FL1	X'00' Control Section (CSECT) SD X'01' Entry Point LD X'02' External Reference ER X'04' Private Code PC X'05' Common Section CM X'06' Dummy External DSECT XD X'0A' Weak External Reference WX X'FF' Dummy Section (DSECT) (no type designator)
Flags	XL1	— Alignment if XD — Zero if LD, ER, or WX — RSECT/AMODE/RMODE flags if SD, PC, or CM Bits 0-3: Reserved Bit 4: 1 = RSECT Bit 5: 0 = RMODE is 24 1 = RMODE is ANY Bits 6-7: 00 = AMODE is 24 01 = AMODE is 24 10 = AMODE is 31 11 = AMODE is ANY

Field	Size	Description
Reserved	HL2	Reserved for future use
ESDID	FL4	External Symbol Dictionary ID (ESDID) or zero
Section Address	AL4	The section address For SD- and LD-type entries it contains the address of the symbol. For PC- and CM-type entries, it indicates the beginning address of the control section. For XD-type entries, it indicates the number of bytes for alignment less one.
Section Length	FL4	The length of the section
LD ID	FL4	For LD-type entries, the ESDID of the CSECT in which the entry point was defined
Reserved	CL8	Reserved for future use
External Name length	HL2	Number of characters in the external name (zero if private code, unnamed common or unnamed DSECT)
Alias Name length	HL2	Number of characters in the Alias name (zero if no alias)
External name	CL(n)	The external name
Alias Section name	CL(n)	The alias name for the section

Source Analysis Record - X'0030'

Field	Size	Description
Statement number	FL4	The statement number of the source record.
ESDID	FL4	The ESDID for the source record.
Input record number	FL4	The input source record number within the current input file. This field is always present except when the source line is macro generated. (That is, the Input record origin value is X'02'.) This field contains the value returned by the exit if the source record is provided by an exit.
Parent record number	FL4	The parent source record number. If the source record was included by a COPY statement or generated by a macro instruction, the Parent input number is the record number of the COPY statement or macro instruction. This field contains the value returned by the input or library exits if the source record is provided by either of these exits.
Input assigned file number	HL2	The input file's assigned sequence number. (Refer to the input file *n* in the Job identification record if the Input record origin is X'01', or the Library Record - X'0060' with Concatenation number *n* otherwise). This field is set to zero if an exit provides the source record.
Parent assigned file number	HL2	The parent file's assigned sequence number. (Refer to the Input file *n* in the Job identification record if the Parent record origin is X'01', or the Library Record - X'0060' with Concatenation number *n* otherwise). This field is set to zero if an exit provides the source record.
Location Counter	FL4	The current location counter for the source record.

Field	Size	Description
Input record origin	XL1	`X'01'` Source line from primary input `X'02'` Source line from Macro generation. `X'03'` Source line from copy code member. `X'04'` Source line from libmac copy code.
Parent record origin	XL1	`X'01'` Source line from primary input `X'02'` Source line from Macro generation. `X'03'` Source line from copy code member. `X'04'` Source line from libmac copy code.
Reserved	XL3	Reserved for future use
Source record type (within source record origin)	XL1	`X'01'` Comment line that is not within a macro definition. `X'02'` Machine instruction that is not within a macro definition. `X'03'` Assembler instruction that is not within a macro definition. This includes conditional assembly instructions such as AIF and SETC. `X'04'` Macro call instruction. `X'05'` Macro definition. All statements between (and including) the MACRO prototype statement and the corresponding MEND statement. This includes nested macro definitions. This field is set to zero for ICTL and EXITCTL assembler instructions.
Assember operation code	XL1	The assembler operation code for assembler instructions. (See note 3 on page 235). This field is only valid if the `'Source Record Type'` is set to `X'03'`.
Flags	XL1	Flag byte for address fields. `X'80'` Address 1 present `X'40'` Address 2 present
Address 1	AL4	The address 1 field from the assembly
Address 2	AL4	The address 2 field from the assembly
Offset of name entry in statement field	HL2	Zero if name entry not present or if the name begins at the beginning of the record. (see notes 1 and 2 on page 235)
Length of name entry	HL2	Zero if name entry not present (see note 2 on page 235)
Offset of operation entry in statement field	HL2	Zero if operation entry not present (see note 2 on page 235)
Length of operation entry	HL2	Zero if operation entry not present (see note 2 on page 235)
Offset of operand entry in statement field	HL2	Zero if operand entry not present (see note 2 on page 235)
Length of operand entry	HL2	Zero if operand entry not present (see note 2 on page 235)
Offset of remarks entry in statement field	HL2	Zero if remarks entry not present (see note 2 on page 235)
Length of remarks entry	HL2	Zero if remarks entry not present (see note 2 on page 235)
Offset of continuation indicator field	HL2	Zero if no continuation indicator present (see note 2 on page 235)
Reserved	CL4	Reserved for future use
Length of input macro or copy member name	HL2	Zero if the input record line does not come from a macro or a copy member

Field	Size	Description
Length of parent macro or copy member name	HL2	Zero if the parent record line does not come from a macro or a copy member
Length of source record	HL2	The length of the actual source record following
Reserved	CL8	Reserved for future use
Input Macro or copy member name	CL(n)	The macro or copy member name if the input record originated from a macro or copy member
Parent macro or copy member name	CL(n)	The macro or copy member name if the parent record originated from a macro or copy member
Source record	CL(n)	

Notes:

1. The offset and length fields are provided to allow the different fields to be retrieved from the source without being dependent on the format of the source record. The offsets are from the start of the source record.
2. The length and offset fields for the name entry, operation entry, remarks entry, and continuation indicator are zero for the following statements:
   - Macro definition statements with a Source Record Type of X'04'.
   - Macro definition statements with a Source Record Type of X'05'.
   - EXITCTL assembler statements
   - ICTL assembler statements
3. The assembler operation code field can contain the operation code values shown in Figure 75. There are no operation codes assigned in the Associated Data Source records for the assembler ICTL and EXITCTL instructions.

Figure 75. Assembler Operation Code Values

Operation Code	Assembler Instruction	Operation Code	Assembler Instruction	Operation Code	Assembler Instruction
X'00'	GBLA	X'17'	REPRO	X'2E'	OPSYN
X'01'	GBLB	X'18'	TITLE	X'2F'	PUSH
X'02'	GBLC	X'19'	ENTRY	X'30'	POP
X'03'	LCLA	X'1A'	EXTRN	X'33'	Literal
X'04'	LCLB	X'1B'	START	X'37'	MHELP
X'05'	LCLC	X'1C'	CSECT	X'38'	AREAD
X'06'	SETA	X'1D'	DSECT	X'3B'	WXTRN
X'07'	SETB	X'1E'	COM	X'3D'	AMODE
X'08'	SETC	X'1F'	EQU	X'3E'	RMODE
X'09'	AIF	X'20'	ORG	X'3F'	RSECT
X'0A'	AGO	X'21'	END	X'40'	CCW0
X'0B'	ANOP	X'22'	LTORG	X'41'	CCW1
X'0C'	COPY	X'23'	USING	X'43'	ASPACE
X'0D'	MACRO	X'24'	DROP	X'44'	AEJECT
X'0E'	MNOTE	X'25'	ACTR	X'45'	ALIAS
X'0F'	MEXIT	X'26'	DC	X'46'	CEJECT
X'10'	MEND	X'27'	DS	X'47'	ADATA
X'12'	ISEQ	X'28'	CCW	X'48'	SETAF
X'13'	PRINT	X'29'	CNOP	X'49'	SETCF
X'14'	SPACE	X'2A'	LOCTR	X'4A'	CATTR
X'15'	EJECT	X'2B'	DXD		
X'16'	PUNCH	X'2C'	CXD		

Source Error Record - X'0032'

Field	Size	Description
Statement number	FL4	The statement number of the statement in error
Error Identifier	CL16	The error message identifier
Error Severity	HL2	The severity of the error
Error message length	HL2	The length of the error message text
Reserved	CL8	Reserved for future use
Error Message	CL(n)	The error message text

Note:

1. This record also includes MNOTEs generated by the assembler.
2. The language of the error diagnostic messages is determined by the LANGUAGE assembler option.

DC/DS Record - X'0034'

Field	Size	Description
ESDID	FL4	The ESDID for the source record.
Number of operands	HL2	The number of operands defined by the source record.

Field	Size	Description
Type Flag	XL1	1... .... Bit 1 = Define Constant (DC, CXD, CCW, CCW0, or CCW1), Bit 0 = Define Storage (DS or DXD) .1.. .... If 'Define Constant' bit is set, bit 1 indicates the operand is a CXD. If 'Define Constant' bit is *not* set, bit 1 indicates the operand is a DXD. ..1. .... If 'Define Constant' bit is set, bit 1 indicates the operand is a CCW, CCW0 or CCW1. ...1 .... Bit 1 indicates this record is associated with an object text record (X'003A'). The object text record is created when a DC statement has a duplication factor greater than 1, and at least one of the operand values has a reference to the current location counter (*****). .... 1... Reserved .... .1.. Reserved .... ..1. Reserved ....1 Reserved
Reserved	CL5	Reserved for future use
Statement Number	FL4	The statement number of the source line that generated this text, if known. Otherwise it contains zeros.
...Location Counter	FL4	The location counter for this operand. This field repeats within the group for the number of operands on the source record.
...Duplication Factor	FL4	The duplication factor for the operand. This field repeats within the group for the number of operands on the source record.
...Bit Offset	XL1	The offset within byte (0-7) for B-type operands. This field repeats within the group for the number of operands on the source record.
...Type Attribute	CL1	As per symbol records. That is, the value that the assembler Type Attribute reference (T') returns. This field repeats within the group for the number of operands on the source record.
...Number of values	HL2	The number of nominal values. This field repeats within the group for the number of operands on the source record.
...Reserved	CL8	Reserved for future use. This field repeats within the group for the number of operands on the source record.
......Byte length	HL2	The number of bytes in the nominal value. This field repeats within the group for the number of nominal values in the operand.
......Bit length	HL2	The number of bits if the operand specifies a bit length that is not a multiple of 8. This field repeats within the group for the number of nominal values in the operand.
......Value	XL(n)	If this record describes a DC, CXD, CCW, CCW0 or CCW1, then the value contains the nominal value. (A DC with a zero duplication factor is treated the same as a DS and this field is not present). If this record describes a DS or DXD, this field is not present. This field repeats within the group for the number of nominal values in the operand. If a byte length is specified (or implied), the value contains the number of bytes specified. The value field is aligned according to the operand type. For example, hexadecimal values are left aligned and packed values are right aligned. If a bit length is specified, the length of the value is the number of bytes required to contain the required bits. For example, if the bit length was 10, the value is 2 bytes in length. The value is in the left most 10 bits. Alignment within the specified number of bits is according to the operand type. For example, hexadecimal values are left aligned and packed values are right aligned.

Field	Size	Description

Note:

1. Only one of the two fields for byte/bit lengths contains a non zero value. This means that there is a byte length or a bit length but not both.
2. No description of any padding is produced. Any padding because of alignment can be calculated by comparing the location counter of the current operand with the sum of the location counter and length of the previous operand.

   The length of the previous operand would need to be calculated using the duplication factor, number of nominal values and the length of each nominal value.
3. High Level Assembler the DC/DS Extension record X'0035' when the duplication factor is greater than 1 and at least one of the operand values has a reference to the current location counter ('*').

The following examples show the format of a DC/DS record for various DC statements.

1. EXAMPLE1 DC 3F'5,6',H'7'

```
ESDID                     F'1'
Number of operands        H'2'
Type flag                 B'1000000'
Statement Number          F'5'
Reserved                  C'     '
   Location counter       X'00000000'
   Bit offset             B'00000000'
   Type attribute         C'F'
   Duplication factor     F'3'
   Number of values       H'2'
   Reserved               CL8
      Byte length         H'4'
      Bit length          H'0'
      Value               XL4'00000005'
      Byte length         H'4'
      Bit length          H'0'
      Value               XL4'00000006'

   Location counter       F'24'
   Bit offset             B'00000000'
   Type attribute         C'H'
   Duplication factor     F'1'
   Number of values       H'1'
   Reserved               CL8
      Byte length         H'2'
      Bit length          H'0'
      Value               XL2'0007'
```

2. EXAMPLE2 DC   P'5,927'

```
ESDID                       F'1'
Number of operands          H'1'
Type flag                   B'10000000'
Statement Number            F'6'
Reserved                    C'      '
   Location counter         X'00000000'
   Bit offset               B'00000000'
   Type attribute           C'P'
   Duplication factor       F'1'
   Number of values         H'2'
   Reserved                 CL8
      Byte length           H'1'
      Bit length            H'0'
      Value                 XL1'5C'

      Byte length           H'2'
      Bit length            H'0'
      Value                 XL2'927C'
```

3. EXAMPLE3 DC   B'101',2B'10111'

```
ESDID                       F'1'
Number of operands          H'2'
Type flag                   B'10000000'
Statement Number            F'7'
Reserved                    C'      '
   Location counter         X'00000000'
   Bit offset               B'00000000'
   Type attribute           C'B'
   Duplication factor       F'1'
   Number of values         H'1'
   Reserved                 CL8
      Byte length           H'1'
      Bit length            H'0'
      Value                 XL1'05'            (binary value 101)

   Location counter         X'00000001'
   Bit offset               B'00000000'
   Type attribute           C'B'
   Duplication factor       F'2'
   Number of values         H'1'
   Reserved                 CL8
      Byte length           H'1'
      Bit length            H'0'
      Value                 XL1'17'            (binary value 10111)
```

4. EXAMPLE4 DC BL.3'101',BL.5'10111,11001'

```
ESDID                      F'1'
Number of operands         H'2'
Type flag                  B'10000000'
Statement Number           F'8'
Reserved                   C'    '
   Location counter        X'00000000'
   Bit offset              B'00000000'
   Type attribute          C'B'
   Duplication factor      F'1'
   Number of values        H'1'
   Reserved                CL8
      Byte length          H'0'
      Bit length           H'3'
      Value                XL1'A0'          (binary value 10100000)

   Location counter        X'00000000'
   Bit offset              B'00000011'
   Type attribute          C'B'
   Duplication factor      F'1'
   Number of values        H'2'
   Reserved                CL8
      Byte length          H'0'
      Bit length           H'5'
      Value                XL1'B8'          (binary value 10111000)

      Byte length          H'0'
      Bit length           H'5'
      Value                XL1'C8'          (binary value 11001000)
```

5. EXAMPLE5 DC 5Y(*-2)

This example shows a DC statement that requires a DC extension record (X'0034') to contain the repeating fields.

```
ESDID                      F'1'
Number of operands         H'1'
Type flag                  B'10010000'
Statement Number           F'9'
Reserved                   C'    '
   Location counter        X'00000000'
   Duplication factor      F'5'
   Bit offset              B'00000000'
   Type attribute          C'Y'
   Number of values        H'1'
   Reserved                C'       '
      Byte length          H'2'
      Bit length           H'0'
      Value                X'FFFE'          (remainder on DC extension)
```

The object text for the remainder of the DC statement:

```
ESDID                      F'1'
Location counter           X'00000002'
Statement Number           F'9'
Language Dependent Field   F'0'
Reserved                   F'0'
Object text length         H'8'
Object text                X'0000000200040006'
```

6. EXAMPLE6 DC 5Y(*-2),5Y(*-1)

   This example shows a DC statement that requires a DC extension record (X'0034') to contain the repeating fields.

   The object code generated, and shown in the assembler listing:

```
                                      2          Print Data
 000000 FFFE000000020004              3          DC    5y(*-2),5y(*-1)
 000008 00060009000B000D
 000010 000F0011
```

   The ADATA records produced:

```
ESDID                      F'1'
Number of operands         H'2'
Type flag                  B'10010000'
Statement Number           F'3'
Reserved                   C'    '
   Location counter        X'00000000'
   Duplication factor      F'5'
   Bit offset              B'00000000'
   Type attribute          C'Y'
   Number of values        H'1'
   Reserved                C'       '
      Byte length          H'2'
      Bit length           H'0'
      Value                X'FFFE'           (remainder on DC extension)

   Location counter        X'0000000A'
   Duplication factor      F'5'
   Bit offset              B'00000000'
   Type attribute          C'Y'
   Number of values        H'1'
   Reserved                C'       '
      Byte length          H'2'
      Bit length           H'0'
      Value                X'0009'           (remainder on DC extension)
```

   The object text for the remainder of the first operand of the DC statement:

```
ESDID                      F'1'
Location counter           X'00000002'
Statement Number           F'3'
Language Dependent Field   F'0'
Reserved                   F'0'
Object text length         H'8'
Object text                X'0000000200040006'
```

   The object text for the remainder of the second operand of the DC statement:

```
ESDID                      F'1'
Location counter           X'0000000C'
Statement Number           F'3'
Language Dependent Field   F'0'
Reserved                   F'0'
Object text length         H'8'
Object text                X'000B000D000F0011'
```

DC/DS Extension Record - X'0035'

Field	Size	Description
ESDID	FL4	The ESDID for the record.
Location Counter	FL4	Address (offset) of the text within the module
Statement number	FL4	The statement number of the source line that generated this text, if known. Zero otherwise.
Reserved	FL8	Reserved for future use
Length of Object text	HL2	The length of the following object text
Object text	XL(n)	The actual object text

Machine Instruction Record - X'0036'

Field	Size	Description
ESDID	FL4	The ESDID for the machine instruction record
Location Counter	FL4	The location counter for this instruction
Reserved	CL8	Reserved for future use
Length of Instruction	HL2	The length of the machine instruction
Value of Instruction	XL(n)	The actual value of the machine instruction

Relocation Dictionary Record - X'0040'

Field	Size	Description
POS.ID	FL4	The external symbol dictionary ID number assigned to the ESD entry for the control section in which the address constant is used as an operand.
REL.ID	FL4	The external symbol dictionary ID number assigned to the ESD entry for the control section in which the referenced symbol is defined.
Address	AL4	The assembled address of the field where the address constant is stored.

Field	Size	Description
Flags	XL1	The 2 digit hexadecimal number represented by the characters in this field is interpreted as follows. First Digit: • 0 indicates that the entry describes an A-type or Y-type constant • 1 indicates that the entry describes a V-type address constant • 2 indicates that the entry describes a Q-type address constant • 3 indicates that the entry describes a CXD entry Second Digit: The first three bits of this digit indicate the length of the constant and whether the base should be added or subtracted. • Bits 0 and 1 – 00 = 1 byte – 01 = 2 bytes – 10 = 3 bytes – 11 = 4 bytes • Bit 2 – 0 = + – 1 = – • Bit 3 – Always 0

Symbol Record - X'0042'

Field	Size	Description
ESDID	FL4	ESDID of the section in which the symbol is defined. This is zero for an undefined symbol type.
Statement Number	FL4	The number of the statement in which the symbol is defined. This is zero for an undefined symbol type.
Symbol Type	XL1	X'00' Undefined name X'01' CSECT / RSECT name X'02' DSECT name X'03' Common section name X'04' Dummy External DSECT name (DXD) X'05' V-type constant name X'06' Qualifier X'07' EXTRN/WXTRN name X'08' LOCTR name X'09' Duplicate name X'0A' Literal name X'0B' *-in-literal name X'0C' EQU name 1 X'0D' Ordinary label X'0E' Unresolvable EQU, DC or DS symbol

Field	Size	Description
Type Attribute	CL1	The Type Attribute can be any of the following values that the assembler Type Attribute reference (T') returns. The type attributes for DC and DS statements include: A A-type address constant, implied length, aligned (also CXD instruction label) B Binary constant C Character constant D Long Floating-point constant, implicit length, aligned E Short Floating-point constant, implicit length, aligned F Fullword Fixed-point constant, implicit length, aligned G Fixed-point constant, explicit length H Halfword Fixed-point constant, implicit length, aligned K Floating-point constant, explicit length L Extended Floating Point constant, implicit length, aligned P Packed Decimal constant Q Q-type address constant, implicit length, aligned R A-, S-, Q-, V- or Y-type address constant, explicit length S S-type address constant, implicit length, aligned V V-type address constant, implicit length, aligned X Hexadecimal constant Y Y-type address constant, implicit length, aligned Z Zoned decimal constant @ Graphic constant The type attributes for data represented by ordinary symbols include: I Machine instruction J Identified as a Control section name M Macro instruction T Identified as an external symbol by EXTRN instruction W CCW, CCW0 or CCW1 instruction $ X'5B' Identified as an external symbol by WXTRN instruction The attribute used if none of the above can be assigned: U Undefined
Duplication Factor	FL4	Number of times the first operand field named by the symbol occurs. This is zero for an undefined symbol type.
Length attribute	HL2	Length in bytes, either specified or by default.
Integer attribute	HL2	Number of positions occupied by the integer portion of fixed-point and decimal constants in their object code form. This is zero for an undefined symbol type.
Scale attribute	HL2	Number of positions occupied by the fractional portion of fixed-point and decimal constants in their object code form. This is zero for an undefined symbol type.
Location Counter	FL4	Contains the offset from the start of the DSECT, the non-relocated address of the instruction belonging to this symbol in a CSECT (this is not always the offset from the start of the CSECT) or the value of the equate. For an undefined symbol, it is zero.
Symbol Flags	XL1	1... .... Bit 1 = 1, the symbol is a relocatable, Bit 0 = the symbol is absolute. This bit is zero for an undefined symbol type. .1.. .... Reserved ..1. .... Reserved ...1 .... Reserved .... 1... Reserved .... .1.. Reserved .... ..1. Reserved ....1 Reserved

Field	Size	Description
Reserved	CL7	Reserved for future use
Symbol name length	HL2	Number of characters in the symbol name.
Symbol name	CL(n)	The symbol name. Variable length.

Note:

For record type 'EQU' specified at **1**, where the 'EQU' is for a relocatable value, the ESDID of the 'EQU' is provided. Where the 'EQU' is non-relocatable, the ESDID of the section in control will be provided. The symbol flags can be checked to determine whether the 'EQU' is relocatable or absolute.

Symbol Cross Reference Record - X'0044'

Field	Size	Description
Symbol length	HL2	The length of the symbol.
Statement Definition	FL4	The statement number where the symbol is defined or declared.
Number of references	HL2	The number of references to the symbol.
Relocatability Type	CL1	C' ' Simple relocatable symbol C'A' Absolute symbol C'C' Complex relocatable symbol
Reserved	CL7	Reserved for future use.
Symbol name	CL(n)	The symbol name. Variable length.
...Reference Flag	CL1	C' ' No branch or modification C'M' Modification reference flag C'B' Branch reference flag C'U' USING reference flag C'D' DROP reference flag C'X' Execute Instruction reference flag
...Statement number	FL4	The statement number on which the symbol is referenced.

Note:

1. The reference flag field and the statement number field occur as many times as the number of references field dictates. That is, if there is a value of ten in the number of references field, then there are ten occurrences of the reference flag and statement number pair.
2. Where the number of references would exceed the record size for the ADATA file then the record is continued on the next record. The continuation flag is set in the common header section of the record.

Library Record - X'0060'

Field	Size	Description
Number of Macros / Copy code members	HL2	Count of the number of macros and copy code members described in this record. For example, if ten macros and source copy code members are retrieved from a data set, the count field contains 10 and there are ten occurrences of the length field and the field containing the either the macro or source copy code names.
Data set Name length	HL2	The length of the data set (file) name.

Field	Size	Description
Data set Volume length	HL2	The length of the data set (file) volume.
Concatenation number	XL2	The library concatenation number.
DDNAME length	HL2	The length of the ddname.
Reserved	CL4	Reserved for future use.
Data set Name	CL(n)	The name of the data set (file) from which the macro or copy member was retrieved, or 'PRIMARY INPUT' for an in stream macro. Under VSE, this field contains the library and sublibrary name.
Data set Volume	CL(n)	The volume identification of the volume where the data set (file) resides.
DDNAME	CL(n)	The ddname of the library.
...Macro name length	HL2	The length of the macro name following.
...Macro name	CL(n)	The name of the macro or source copy code that has been used. If the source is 'PRIMARY INPUT' then this field contains the macro name from the source program.

Note:

If a LIBRARY user exit has been specified for the assembly, and the LIBRARY user exit has opened the Library data set, the record contains the library names returned by the user exit.

Library Member and Macro Cross Reference Record - X'0062'

Field	Size	Description
Concatenation Number	FL4	The concatenation number of the library or primary input file
Statement Definition	FL4	The statement number is: 0 — When the member or macro is retrieved from a library >0 — When the macro is defined in the primary input file. It represents the statement number where the macro is defined.
Concatenation Type	CL1	C'L' — Concatenation number refers to a library C'P' — Concatenation number refers to the primary input
Statement Definition Flag	CL1	C'X' — The macro is read from the library and imbedded in the primary source, using the LIBMAC option. C' ' — The flag is usually blank except in special cases, as described above
Number of references	FL4	The number of references to the member or macro
Reserved 1	CL8	Reserved for future use.
Member or Macro name	CL8	The name of the member or macro.
Parent Macro Name	CL8	The name of the macro that called this macro or issued the COPY instruction. This field contains PRIMARY when the member or macro is called directly from the primary input file.
....Number of references in this block by the caller	FL4	The number of references to the member or macro
...Reference Statement Number	FL4	The statement number on which the member is copied or included, or the statement number on which the macro is called

Field	Size	Description
...Reference Flag	CL1	C' ' Blank means the reference is caused by a macro call C'C' Reference is caused by a COPY instruction
...Reserved 2	XL1	Reserved for future use

Note:

1. The Calling Macro Name field immediately follows Member or Macro name field
2. The Reference Statement Number, the Reference Flag and the Reserved 2 fields occur as many times as the Number of References field dictates. For example, if there is a value of ten in the Number of References field, there are ten occurrences of the Reference Statement Number, the Reference Flag and the Reserved 2 fields.
3. Where the number of references would exceed the record size for the ADATA file then the record is continued on the next record. The continuation flag is set in the common header section of the record.

User-supplied Information Record - X'0070'

Field	Size	Description
User Field 1	XL4	User-specified binary data
User Field 2	XL4	User-specified binary data
User Field 3	XL4	User-specified binary data
User Field 4	XL4	User-specified binary data
User data length	HL2	Length of following field
User data	CL(n)	User-specified character data

USING Map Record - X'0080'

Field	Size	Description
Record type	XL1	X'00' USING record X'20' POP record X'40' PUSH record X'80' DROP record
Using Flag	XL1	USING type (ORDINARY, LABELED, DEPENDENT, LABELED DEPENDENT) X'00' Ordinary USING X'10' Labeled USING X'20' Dependent USING X'30' Labeled Dependent USING
Location ESDID	XL2	The value of the ESDID of the current section when the USING, DROP, PUSH USING, or POP USING was issued.
Statement number	FL4	The statement number of the USING, DROP, PUSH USING, or POP USING.
Location Counter	FL4	The value of the location counter when the USING, DROP, PUSH USING, or POP USING was issued.
USING value	FL4	The value of the USING statements first-operand expression. This is zero for PUSH, POP and DROP.

Field	Size	Description
Last statement	FL4	The last statement number for which this base-register was used in converting a symbolic address into its base-displacement form. This is zero for PUSH, POP and DROP.
Using ESDID	XL4	For ordinary and labeled USING instructions, this field indicates the ESDID of the section specified on first operand of the USING statement. For dependent USING instructions, this field indicates the ESDID of the section specified on the corresponding ordinary USING instruction that is used to resolve the address. This is zero for PUSH, POP and DROP.
Register	XL1	The register used in the USING. This is zero for PUSH and POP. Where a DROP with no operand, or a DROP ALL is specified this field contains X'FF'.
Displacement	XL2	The maximum displacement for this USING register. This is zero for PUSH, POP and DROP.
Reserved	CL7	Reserved for future use.
Label length	HL2	The length of the label and USING text field following. This is zero for PUSH and POP. This length field is rounded up to a doubleword boundary. Hence if the text was 13 bytes in length the length would be set at 16 and the text blank padded on the right.
Label	CL(n)	The source text for the LABEL and USING from the source using record.

Statistics Record - X'0090'

Field	Size	Description
Buffer pool allocation	FL4	The number of Kilobytes (KB) of storage allocated to the buffer pool
Required In-storage	FL4	The number of Kilobytes (KB) of storage required to make the assembly be an in-storage assembly.
Primary input records	FL4	The number of primary input records read for the assembly
Library records	FL4	The number of library records read for the assembly
Work file reads	FL4	The number of work file reads for the assembly
Print records written	FL4	The number of print records written for the assembly
Object records written	FL4	The number of object records written for the assembly
Work file writes	FL4	The number of work file writes for the assembly
Adata file writes	FL4	The number of Adata file writes for the assembly
Adata calls	FL4	The number of calls to the ADATA exit This field is zero if no exit is present.
Adata added records	FL4	The number of records added by the ADATA exit This field is always zero.
Adata deleted records	FL4	The number of records deleted by the ADATA exit This field is always zero.
Adata diagnostic messages	FL4	The number of diagnostic messages returned by the ADATA exit This field is zero if no exit is present.
Library calls	FL4	The number of calls to the LIBRARY exit This field is zero if no exit is present.

Field	Size	Description
Library added records	FL4	The number of records added by the LIBRARY exit This field is zero if no exit is present.
Library deleted records	FL4	The number of records deleted by the LIBRARY exit This field is zero if no exit is present.
Library diagnostic messages	FL4	The number of diagnostic messages returned by the LIBRARY exit This field is zero if no exit is present.
Listing calls	FL4	The number of calls to the LISTING exit This field is zero if no exit is present.
Listing added records	FL4	The number of records added by the LISTING exit This field is zero if no exit is present.
Listing deleted records	FL4	The number of records deleted by the LISTING exit This field is zero if no exit is present.
Listing diagnostic messages	FL4	The number of diagnostic messages returned by the LISTING exit This field is zero if no exit is present.
Object calls	FL4	The number of calls to the OBJECT exit This field is zero if no exit is present.
Object added records	FL4	The number of records added by the OBJECT exit This field is zero if no exit is present.
Object deleted records	FL4	The number of records deleted by the OBJECT exit This field is zero if no exit is present.
Object diagnostic messages	FL4	The number of diagnostic messages returned by the OBJECT exit This field is zero if no exit is present.
Source calls	FL4	The number of calls to the SOURCE exit This field is zero if no exit is present.
Source added records	FL4	The number of records added by the SOURCE exit This field is zero if no exit is present.
Source deleted records	FL4	The number of records deleted by the SOURCE exit This field is zero if no exit is present.
Source diagnostic messages	FL4	The number of diagnostic messages returned by the SOURCE exit This field is zero if no exit is present.
Punch calls	FL4	The number of calls to the PUNCH exit This field is zero if no exit is present.
Punch added records	FL4	The number of records added by the PUNCH exit This field is zero if no exit is present.
Punch deleted records	FL4	The number of records deleted by the PUNCH exit This field is zero if no exit is present.
Punch diagnostic messages	FL4	The number of diagnostic messages returned by the PUNCH exit This field is zero if no exit is present.

Field	Size	Description
Term calls	FL4	The number of calls to the TERM exit This field is zero if no exit is present.
Term added records	FL4	The number of records added by the TERM exit This field is zero if no exit is present.
Term deleted records	FL4	The number of records deleted by the TERM exit This field is zero if no exit is present.
Term diagnostic messages	FL4	The number of diagnostic messages returned by the TERM exit This field is zero if no exit is present.
Assembly start time	FL4	The local time when the assembly commenced. This time is recorded after data set allocation, storage allocation, invocation parameter processing, and other initialization processing. Stored in packed format as *hhmmssth* *hh* The hour *mm* The minute *ss* The second *t* Tenths of a second *h* Hundredths of a second
Assembly stop time	FL4	The local time when the assembly completed Stored in packed format as *hhmmssth* *hh* The hour *mm* The minute *ss* The second *t* Tenths of a second *h* Hundredths of a second
Processor time	FL4	The number of processor seconds utilized by this assembly The low order bit represents 1 microsecond.

Appendix E. Sample Program

The sample program included with High Level Assembler is described in this appendix. This program demonstrates some basic assembler language, macro, and conditional assembly features, most of which are unique to High Level Assembler. The highlighted characters in the descriptions below refer to corresponding characters in the listing that precedes the descriptions.

```
                                       HIGH LEVEL ASSEMBLER OPTION SUMMARY                        PAGE    1
                                                                               HLASM R2.0  1995/03/26 08.00
OVERRIDING PARAMETERS-  batch,mxref(source),sysparm(SAMPLE PROGRAM)
NO PROCESS STATEMENTS


OPTIONS FOR THIS ASSEMBLY

 NOADATA
   ALIGN
 NOASA
   BATCH
 NOCOMPAT
 NODBCS
 NODECK
   DXREF
   ESD
 NOEXIT
   FLAG(0,NOALIGN,NOCONT,NORECORD,NOSUBSTR)
 NOFOLD
   LANGUAGE(UE)
 NOLIBMAC
   LINECOUNT(60)
   LIST(121)
   MXREF(SOURCE)
   OBJECT
   OPTABLE(UNI)
 NOPCONTROL
 NOPESTOP
 NOPROFILE
 NORA2
 NORENT
   RLD
   SIZE(MAX)
   SYSPARM(SAMPLE PROGRAM)
 NOTERM
 NOTEST
 NOTRANSLATE
 NOUSING
 NOXOBJECT
   XREF(FULL)

NO OVERRIDING DD NAMES
```

```
BIGNAME                                     EXTERNAL SYMBOL DICTIONARY                     PAGE    2

SYMBOL   TYPE   ID    ADDRESS   LENGTH     LD ID  FLAGS ALIAS-OF              HLASM R2.0  1995/03/26 08.00

A        SD 00000001 00000000 000000DE            00
PD2      CM 00000002 00000000 00000814            00 A

BIGNAME  Sample program.  1ST TITLE statement has no name, 2ND one does                               PAGE    3
  ACTIVE USINGS: NONE

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                             HLASM R2.0  1995/03/26 08.00

000000                               2 a       csect                                                   00002000
                 R:8  00000          3         using *,8                                               00003000
000000 1BFF                          4         sr    15,15      Set return code to zero                00004000
000002 07FE                          5         br    14          and return.                           00005000

                                     7 **********************************************************************  00007000
                                     8 *              PUSH  and POP  statements                          *  00008000
                                     9 * Push down the PRINT statement, replace it, retreive original      *  00009000
                                    10 **********************************************************************  00010000

                          B         12         push  print     Save Default setting '  PRINT ON,NODATA,GEN'   00012000
                                    13         print nogen,data                                        00013000
000004 0A23                         14         wto   mf=(E,(1))                  Expansion not shown       00014000
000006 01230ABC0102030A   C         16  dc x'123,ABC',(reallylongsymbol-transylvania)b'1,10,11,1010,1011,1100' 00015000
00000E 0B0C0102030A0B0C
000016 0102030A0B0C0102
00001E 030A0B0C
                                    17         pop   print                 Restore default PRINT setting   00016000
                                    18         wto   mf=(E,(1))                  Expansion shown           00017000
000022 0A23                         19+        SVC   35                     ISSUE SVC 35              @L2C 01-WTO
000024 01230ABC0102030A             20  dc x'123,ABC',(reallylongsymbol-transylvania)b'1,10,11,1010,1011,1100' 00018000

                                    22 **********************************************************************  00020000
                                    23 *                      LOCTR  instruction                          *  00021000
                                    24 * LOCTR allows 'REMOTE' assembly of constant                        *  00022000
                                    25 **********************************************************************  00023000

000040 5850 80AC           000AC    27         l     5,constant                                        00025000
0000AC                    D         28 deecees  loctr                                                  00026000
0000AC 00000005                     29 constant dc    f'5'      Constant coded here, assembled behind LOCTR A  00027000
000044                              30 a       loctr                Return to 1st LOCTR in CSECT A     00028000

                                    32 **********************************************************************  00030000
                                    33 *  3 operand EQUATE with forward reference in 1ST operand          *  00031000
                                    34 **********************************************************************  00032000

000044 1812                         36 a5      lr    1,2            L'A5 = 2, T'A5 = I                  00034000
                                    37         print data                                              00035000
000046 0000
000048 413243F6A8885A30             38 a7 dc l'3.14159265358979323846264338327950288841972' L'A7 = 16,T'A7 = L  00036000
000050 338D313198A2E037
                                    39 &type   setc  t'a7                                              00037000
                          E         40 a8      equ   b5,l'a5,c'&type'                                  00038000
                           000B0      +a8      equ   b5,l'a5,c'L'                                      00038000
```

A The external symbol dictionary shows a named common statement. The named common section is defined in statement 173.

B Statement 12: Save the current status of the PRINT statement.

Statement 13: Modify the print options to DATA and NOGEN.

Statement 14: Macro call; note that the expansion (statement 15) is not printed.

Statement 16: All 28 bytes of data are displayed to the two-operand DC.

Statement 17: Restore earlier status of PRINT.

Statements 19 and 20: The generated output of the macro WTO is shown and only the first 8 bytes of the data are displayed.

C Statements 16 and 20: Multiple constants are allowed in hexadecimal and binary DC operands, and neither symbol in the duplication factor has been defined yet. Definition occurs in statements 115 and 116.

D Statements 28, 30, 151, and 164 show use of the LOCTR assembler instruction. This feature allows you to break down control sections into 'subcontrol' sections. It can be used in CSECT, RSECT, DSECT, and COM. LOCTR has many of the features of a control section; for example, all of the first LOCTR in a section is assigned space, then the second, and so on. The name of the control section automatically names the first LOCTR section. Thus LOCTR A is begun, or continued, at statements 2, 30, and 170. The location counter value shown each time is the continued value of the LOCTR. On the other hand, various LOCTR sections within a control section have common addressing as far as USING statements are concerned, subject to the computed displacement falling within 0 through 4095. In the sample, CONSTANT is in LOCTR DEECEES but the instruction referring to it (statement 27) has no addressing problems.

E Three-operand EQU. Here, we assign: (a) the value of B5 (not yet defined) to A8, (b) the length attribute of A5 to A8, and (c) the type attribute of A7 to A8. If no operand is present in an EQU statement, the type attribute is U and the length attribute is that of the first term in the operand expression. Symbols present in the label and operand field must be previously defined. You cannot express the type attribute of A7 directly in the EQU statement. The EQU statement at 40 could have been written

```
a8        equ       b5,2,c'L'

a8        equ       b5,x'2',x'D3'
```

```
BIGNAME  Sample program.  1ST TITLE statement has no name, 2ND one does                      PAGE    4
  ACTIVE USINGS: a,R8

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                               HLASM R2.0  1995/03/26 08.00

                                     42 ***********************************************************************  00040000
                                     43 * Implicit declaration of locals &A, &C -- Use of SETC dup factor to *  00041000
                                     44 *   produce SETC string longer than 8, MNOTE in open code           *  00042000
                                     45 ***********************************************************************  00043000

                              F      47 &la8     seta  l'a8                                                      00045000
                                     48 &ta8     setc  t'a8                                                      00046000
                                     49          mnote *,'Length of A8 = &LA8, Type of A8 = &TA8'                00047000
                              G     +*,Length of A8 = 2, Type of A8 = L                                          00047000

                                     51 &a       seta  2                                                         00049000
                              H      52 &c       setc  (&a+3)'STRING,'                                           00050000
                                     53          mnote *,'&&C has value = &c'                                    00051000
                                       +*,&C has value = STRING,STRING,STRING,STRING,STRING,                     00051000

                                     55 ***********************************************************************  00053000
                              I      56 * Examples of 4 byte self-defined terms, unary + and -              *  00054000
                                     57 ***********************************************************************  00055000

000058 7FFFFFFFC1C2C3C4              59          dc    a(2147483647,C'ABCD',X'ffffffff')                         00057000
000060 FFFFFFFF
000064 181D                          60          lr    -1+2,16+-3                                                00058000

                         FFFFFFE8    62 X        equ   4*-6                                                      00060000
```

F Set symbols &LA8 and &TA8 have not been previously declared in LCL or GBL statements. Therefore, they default to local variable symbols as follows: &LA8 is an LCLA SET symbol because it appears in the name field of a SETA; &TA8 is an LCLC SET symbol because it is first used in a SETC.

G MNOTEs can appear in open code. As such, they have all properties of MNOTEs inside macros, including substitution.

H A SETC expression can have a duplication factor. The SETA expression must be enclosed in parentheses and immediately precede the character string, the substring notation, or the type attribute reference.

I Statements 59 through 62 show 4-byte self-defining values and unary + and –. The value of X appears later in a literal address constant (see statement 252).

```
BIGNAME  Insert Programmer Macro in Source Stream now                                         PAGE    5
  ACTIVE USINGS: a,R8

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                             HLASM R2.0  1995/03/26 08.00

                                     64 *********************************************************************  00062000
                                     65 *  Mixed keywords and positional parameters, extended AGO and AIF    *  00063000
                                     66 *     statements, declaration and use of subscripted SET symbols,    *  00064000
                                     67 *       Use of created SET symbols, extended SET statements          *  00065000
                                     68 *********************************************************************  00066000

                               J     70            macro                                                         00068000
                                     71            demo  &p1,&key1=A,&p2,&key2=1,&p3,&key3=3,&p4                  00069000
                               K     72 &loc(1)  setc  '2','3'        &LOC is dimensioned LCLC by default        00070000
                                     73            gblc  &xa(5),&xb(20),&xc(1)                                   00071000
                                     74            aif   ('&system_id'(1,3) eq 'VSE').vse                         00072000
                               L     75            &p1   &syslist(4),&syslist(5),&syslist(6),mf=E                 00073000
                                     76            ago   .notvse                                                 00074000
                                     77 .vse     anop                 Use VSE WRITE macro parameters             00075000
                                     78            &p1   &syslist(4),SQ,&syslist(6)                              00076000
                                     79 .notvse  anop                                                            00077000
                                     80 &n       seta  1                                                         00078000
                               M     81            ago   (&key2).mnote1,.mnote2,.mnote3                          00079000
                                     82 &n       seta  2                                                         00080000
                                     83            mnote *,'&&KEY2 not 1,2, or 3---Use &&KEY3 in place of it'    00081000
                               N     84            aif   (&key3 eq 1).mnote1,                                   X00082000
                                                         (&key3 eq 2).mnote2,(&key3 eq 3).mnote3                 00083000
                                     85            mnote *,'Both &&KEY2 and &&KEY3 fail to qualify'              00084000
                                     86            ago   .common                                                 00085000
                                     87 .mnote1  mnote *,'&&KEY&LOC(&N) = 1'                                     00086000
                                     88            ago   .common                                                 00087000
                                     89 .mnote2  mnote *,'&&KEY&LOC(&N) = 2'                                     00088000
                                     90            ago   .common                                                 00089000
                                     91 .mnote3  mnote *,'&&KEY&LOC(&N) = 3'                                     00090000
                                     92 .common  l     5,8(,10)       Note that opcodes, operands & comments     00091000
                                     93 &xb(2) sr 9,10                  on MODEL statements                      00092000
                               O     94 &(x&key1)(2) lm 12,13,=a(a5,x)    are kept in place unless displaced    00093000
                                     95 &p2 st 7,&p3                          as a result of substitution        00094000
                                     96            mend                                                          00095000

                                     98 *****          DEMO  MACRO  instruction (call)                           00097000

                               P    100            gblc  &xa(1),&xb(2),&xc(3)                                    00099000
                                    101 &xa(1)   setc  'A','MISSISSIPPI'                                         00100000
                                    102 &xb(1)   setc  'B','SUSQUEHANNA'                                         00101000
                                    103 &xc(1)   setc  'C','TRANSYLVANIA'                                        00102000
                               Q    104            demo  key3=2,write,reallylongsymbol,                         M00103000
                                                         a8+8*(b5-constant-7)(3),key1=C,(6),SF,                 N00104000
                                                         (8),key2=7                                              00105000
000066 1816                         105+         LR    1,6                         LOAD DECB ADDRESS        03-IHBRD
000068 9220 1005      00005         106+         MVI   5(1),X'20'              SET TYPE FIELD               03-IHBRD
00006C 5081 0008            00008   107+         ST    8,8(1,0)                    STORE DCB ADDRESS        03-IHBRD
000070 58F1 0008            00008   108+         L     15,8(1,0)           LOAD DCB ADDRESS                 03-IHBRD
000074 58F0 F030            00030   109+         L     15,48(0,15)                 LOAD RDWR ROUTINE ADDR   03-IHBRD
000078 05EF                         110+         BALR  14,15                       LINK TO RDWR ROUTINE     03-IHBRD
                                    111+*,&KEY2 not 1,2, or 3---Use &KEY3 in place of it                  01-00083
                                    112+*,&KEY3 = 2                                                        01-00089
00007A 5850 A008            00008   113+         l     5,8(,10)       Note that opcodes, operands & comments   01-00092
00007E 1B9A                     R   114+SUSQUEHANNA sr 9,10            on MODEL statements                   01-00093
000080 98CD 8090            00090   115+TRANSYLVANIA lm 12,13,=a(a5,x)    are kept in place unless displaced   01-00094
000084 5073 8098            00098   116+reallylongsymbol st 7,a8+8*(b5-constant-7)(3)                      X01-00095
                                       +                                      as a result of substitution
```

J The macro DEMO is defined after the start of the assembly. Macros can be defined at any point and, having been defined, expanded, or both, can be redefined. The parameters on the prototype are a mixture of keywords and positional operands. &SYSLIST may be used. The positional parameters are identified and numbered 1, 2, 3 from left to right; keywords are skipped over.

K Statement 72 shows the extended SET feature (as well as implicit declaration of &LOC(1) as an LCLC). Both &LOC(1) and &LOC(2) are assigned values. One SETA, SETB, or SETC statement can then do the work of many.

L Statement 75 is a model statement with a symbolic parameter in its operation field. This statement is edited as if it is a macro call; at this time, each operand is denoted as positional or keyword. At macro call time, you cannot reverse this decision. Even though it's treated as a macro, it is still expanded as a machine or assembler operation.

M Statement 81 shows the computed AGO statement. Control passes to .MNOTE1 if &KEY2 is 1, to .MNOTE2 if &KEY2 is 2, to .MNOTE3 if &KEY2 is 3, or otherwise it falls through to the model statement at 82.

N Statement 84 shows the extended AIF facility. This statement is written in the alternative format. The logical expressions are examined from left to right. Control passes to the sequence symbol corresponding to the first true expression encountered, or else falls through to the next model statement.

O Statement 94 contains a subscripted created SET symbol in the name field. The created SET symbol has the form &(*e*), where *e* is an expression made up of character strings, variable symbols, or both. When the symbol is encountered during macro generation, the assembler evaluates the expression *e*. The operation code DEMO is used as a macro instruction in statement 104, and &KEY1 is given the value C. The *e* in this case is X&KEY1, which results in the value XC. Thus the name field in statement 94, &(x&key1)(2), becomes &XC(2). Statement 103 assigns the value C to &XC(1), and the value `TRANSYLVANIA` to &XC(2). The model statement (94) is generated at statement 115; the name field contains `TRANSYLVANIA`. The sequence field of statement 115, shows that this statement is a level 01 expansion of a macro, and the corresponding model statement is statement number 94.

You can use created SET symbols wherever regular SET symbols are used in declarations, name fields, or operands of SET statements, in model statements, etc. Likewise, they are subject to all the restrictions of regular SET symbols.

P In statements 100 and 101, &XA is declared as a subscripted global SETC variable with a subscript of 1 and, in the next statement, which is an extended SET statement, we store the value `MISSISSIPPI` into &XA(2). The assembler allows up to 2,147,483,647 array values in a subscripted global SETC symbol.

Q Statement 104 is the macro instruction DEMO. &P1 has the value WRITE. Therefore, the model statement at statement 75 becomes an inner macro instruction, WRITE, producing the code at statements 105-110. The sequence field of these statements contains 03-IHBRD, indicating that they are generated by a level 03 macro (DEMO is 01, WRITE is 02) named IHBRDWRS. It is an inner macro called by WRITE.

R Statements 115 and 116 contain some ordinary symbols longer than 8 characters. The limit for ordinary symbols, operation codes (for programmer and library macros and operation codes defined through OPSYN), variable symbols, and sequence symbols is 63 characters (including the & and . in the latter two instances, respectively).

```
BIGNAME  Insert Programmer Macro in Source Stream now                                                  PAGE    7
  ACTIVE USINGS: a,R8

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1995/03/26 08.00

                                    118 ***********************************************************************  00107000
                                    119 *  Copy 'NOTE' macro in from maclib, rename it 'MARK', call it under *  00108000
                                    120 *    its ALIAS -- in expansion of MARK, notice reference back to     *  00109000
                                    121 *      definition statements in 'columns' 76-80 of expansion         *  00110000
                                    122 ***********************************************************************  00111000
                             S      124          copy  note                                                    00113000
                                    125          MACRO                                                          00010000
                                    126 &NAME    NOTE  &DCB,&DUMMY=,&TYPE=REL                                   00020000
                                    127 .* $MAC(NOTE):                                                          00030000
                                    128 .*  5665-XA2                                                            00040000
                                    129 .*  CONTAINS RESTRICTED MATERIALS OF IBM                                00050000
                                    130 .*  (C) COPYRIGHT IBM CORP. 1984                                        00060000
                                    131 .*  LICENSED MATERIALS - PROPERTY OF IBM                                00070000
                                    132 .*  REFER TO COPYRIGHT INSTRUCTIONS                                     00080000
                                    133 .*  FORM NUMBER G120-2083.                                              00090000
                                    134 .* STATUS =  MVS/XA* DFP RELEASE 1.2                           @H1   00100000
                                    135 .*                                                                      00110000
                                    136 .* CHANGE ACTIVITY =                                                    00120000
                                    137 .*                                                                      00130000
                                    138 .* $H1=3480,JDP1111,,STLPKH:  3480 SUPPORT                         *  00140000
                                    139 .*                                                                      00150000
                                    140          AIF   ('&DCB' EQ '').ERR                                       00160000
                                    141 &NAME    IHBINNRA &DCB                                                  00170000
                                    142          AIF   ('&TYPE' NE 'REL').NOTREL                          @H1A 00180000
                                    143          L     15,84(0,1)                  LOAD NOTE  RTN ADDRESS 00190000
                                    144          BALR  14,15                       LINK TO NOTE  ROUTINE  00200000
                                    145          MEXIT                                                          00210000
                                    146 .NOTREL  AIF   ('&TYPE' NE 'ABS').ERR1                            @H1A 00220000
                                    147          SLR   0,0                INDICATES NOTE MACRO       @H1A 00230000
                                    148          LA    15,32              ROUTER CODE                @H1A 00240000
                                    149          SVC   109                SUPERVISOR CALL            @H1A 00250000
                                    150          MEXIT                                                     @H1A 00260000
                                    151 .ERR1    MNOTE 8,'INVALID PARAMETER FOR TYPE'                 @H1A      00270000
                                    152          MEXIT                                                     @H1A 00280000
                                    153 .ERR     IHBERMAC 6                                                     00290000
                                    154          MEND                                                           00300000
                             T      157 mark     opsyn note      Comments of generated statements occupy same   0011600
                                    158          mark  (6)         'COLUMNS' as those in MODEL statements      00117000
000088 1816                         159+         LR    1,6                         LOAD PARAMETER REG 1   02-IHBIN
00008A 58F0 1054          00054     160+         L     15,84(0,1)                  LOAD NOTE  RTN ADDRESS 01-00143
00008E 05EF                         161+         BALR  14,15                       LINK TO NOTE  ROUTINE  01-00144
                                    163 ***********************************************************************  00119000
0000B0                              164 deecees  loctr        Switch to alternate location counter             00120000
0000B0 0B0000B000000050             165 b5       ccw   X'0b',b5,0,80                                            00121000
                                    167 ***********************************************************************  00123000
                                    168 *  Display of &SYSTIME, &SYSDATE, &SYSPARM and &SYSLOC               *  00124000
                                    169 ***********************************************************************  00125000
                                    171          print nodata                                                   00127000
                             U      172   dc  c'TIME = &systime, DATE = &sysdate, PARM = &sysparm'            0012800

BIGNAME  Insert Programmer Macro in Source Stream now                                                  PAGE    8
  ACTIVE USINGS: a,R8

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1995/03/26 08.00

0000B8 E3C9D4C5407E40F1                +  dc  c'TIME = 08.00, DATE = 02/28/95, PARM = '                    00128000
                                    174          macro                                                          00130000
                                    175          locate                                                         00131000
                                    176 &sysect  csect      Display of current control section                  00132000
                                    177 &sysloc  loctr          and location counter                            00133000
                                    178          mend                                                           00134000
                                    180          locate                                                         00136000
0000DE                       V      181+a        csect      Display of current control section                  01-0017
0000DE                              182+deecees  loctr          and location counter                            01-00177
000090                              183 a        loctr                                                          00137000
```

S Library macros can be inserted into the source stream as programmer macros by use of a COPY statement. The result (statements 126 to 141) is essentially a programmer macro definition. When a library macro is brought in and expanded by use of a macro instruction, the assembler (1) looks the macro up by its member-name and (2) verifies that this same name is used in the operation field of the prototype statement. Therefore, for example, DCB has to be cataloged as DCB. However, as COPY code, the member name bears no relationship to any of the statements in the member. Thus, several variations of a given macro could be stored as a library under separate names, then copied in at various places in a single assembly as needed. (High Level Assembler allows you to define and redefine a macro any number of times).

T In statement 157, MARK is made a synonym for NOTE. To identify NOTE as a macro, it has to be used as either a system macro call (that is, from a macro library) or a programmer macro definition before its use in the operand field of an OPSYN statement. The COPY code at statements 126 through 157 is a programmer macro definition. The macro instruction at statement 158 is MARK. We can use MARK and NOTE interchangeably. If required, we could remove NOTE as a macro definition in the following way:

```
MARK      OPSYN          NOTE
NOTE      OPSYN
```

We could then refer to the macro only as MARK.

U Statement 172 demonstrates &SYSTIME, &SYSDATE and &SYSPARM. The values for the first two are the same as in the heading line. The value for &SYSPARM is the value passed in the PARM field of the EXEC statement or the default value assigned to &SYSPARM when High Level Assembler is installed.

V System variable symbols &SYSLOC and &SYSECT are displayed. The sequence field indicates that the model statements are statements 176 and 177.

```
BIGNAME  Ordinary, Labeled and Dependent USING Instructions                                         PAGE    9
  ACTIVE USINGS: a,R8

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                 HLASM R2.0  1995/03/26 08.00

                                    185 ********************************************************************  00139000
000000                      W       186 pd2      com                Named COMMON thrown in for good measure   0014000
000000                              187          ds    500f                                                   00141000
0007D0 1867                         188          lr    6,7                                                    00142000
                                    190 ********************************************************************  00144000
                                    191 *  Use of ordinary, labeled and dependent USING Instructions        *  00145000
                                    192 ********************************************************************  00146000
                            X
                R:C  007D2          194          using *,12                                                   00148000
0007D2 4110 C022           007F4    195          la    1,area1                                                00149000
0007D6 4120 C032           00804    196          la    2,area2                                                00150000
                R:1  00000          197          using first,1               Ordinary USING                   00151000
                R:2  00000          198 lab      using first,2               Labeled USING                    00152000
              2 008  00000 00008    199          using second,first2         Dependent USING                  00153000
              2 008  00000 00008    200 labdep   using third,lab.first2      Labeled dependent USING          00154000
0007DA D207 1000 8098 00000 00098   201          mvc   first1,=cl8'1st'      Uses ordinary USING              00155000
0007E0 D207 2000 8098 00000 00098   202          mvc   lab.first1,=cl8'1st'  Uses labeled USING               00156000
0007E6 D203 1008 80A0 00000 000A0   203          mvc   second1,=cl4'2nd'     Uses dependent USING             00157000
0007EC D201 2008 80A4 00000 000A4   204          mvc   labdep.third1,=cl2'3d' Uses labeled dependent USING    00158000
0007F4                              205 area1    ds    0f                    First data area                  00159000
0007F4                              206 area1a   ds    cl8                                                    00160000
0007FC                              207 area1b   ds    cl8                                                    00161000
000804                              208 area2    ds    0f                    Second data area                 00162000
000804                              209 area2a   ds    cl8                                                    00163000
00080C                              210 area2b   ds    cl8                                                    00164000
000000                              211 first    dsect                       First dsect                      00165000
000000                              212 first1   ds    cl8                                                    00166000
000008                              213 first2   ds    cl8                                                    00167000
000000                              214 second   dsect                       Second dsect                     00168000
000000                              215 second1  ds    cl4                                                    00169000
000004                              216 second2  ds    cl4                                                    00170000
000000                              217 third    dsect                       Third dsect                      00171000
000000                              218 third1   ds    cl2                                                    00172000
i000002                             219 third2   ds    cl2                                                    00173000
```

W Illustration of named COMMON. You can establish addressability for a named COMMON section with:

```
        USING          section-name,register
```

You can address data in a blank COMMON section by labeling a statement *after* the COMMON statement.

X In statement 197, an ordinary USING is established for AREA1 using the DSECT FIRST. When the fields within DSECT FIRST are referenced, register 1 is used to resolve the address as in statement 201.

In statement 198, a labeled USING is established for AREA2 using the DSECT FIRST. Register 2 is used to resolve the address for fields within AREA2 when referred to using the label as in statement 202.

In statement 199, a dependent USING is established for the field FIRST2 using the DSECT SECOND. The corresponding ordinary USING for field FIRST2 is the USING on statement 197. It uses register 1 to resolve the address. The statement on line 203 specifies a field within DSECT SECOND and the assembler uses register 1 to resolve the address.

In statement 200, a labeled dependent USING is established for the field FIRST2 using the DSECT THIRD. The USING specifies the labeled USING LAB to resolve the address for field FIRST2. In statement 204, the labeled dependent USING is specified and register 2 is used to resolve the address of the field THIRD1.

```
 BIGNAME  Predefined Absolute Symbols in SETA and SETC expressions                              PAGE   10
   ACTIVE USINGS: first,R1  second,R1+X'8'  a,R8  pd2+X'7D2',R12  lab.first,R2  labdep.third,R2+X'8'

 D-LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1995/03/26 08.00

                                     221 **********************************************************************  00175000
                                     222 *  Use of predefined absolute symbols in SETA and SETC expressions   *  00176000
                                     223 **********************************************************************  00177000
                              00064  225 hundred  equ   100                                                      00179000
                                     226 &dividnd seta  20                                                       00180000
                              Y      227 % seta  &dividnd*100/40         Predefined symbol in SETA       0018100
                              00032  228 fifty    equ   50                                                       00182000
                              00081  229 chara    equ   c'a'                                                     00183000
                              Z      230 &longwd  setc  (hundred)chara          Predefined symbol in SETC        0018400
                                     231          dc    c'&longwd'                                               00185000
 000004 8181818181818181               +          dc    c'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaX00185000
 00000C 8181818181818181               +                aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
                                     232 &twowds  setc  (fifty)chara.' '.(hundred/2)'B'                          00186000
                                     233          dc    c'&twowds'                                               00187000
 000068 8181818181818181               +          dc    c'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa BBBX00187000
i000070 8181818181818181               +                BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB'
```

Y In statement 227, the SETA statement specifies a predefined absolute symbol (DIVIDND) and well as other arithmetic terms.

Z In statements 230 and 231, the SETC statement specifies a predefined absolute symbol (FIFTY) as the duplication factor.

```
BIGNAME  Symbol Attribute Enhancements                                                            PAGE   11
  ACTIVE USINGS: first,R1  second,R1+X'8'  a,R8  pd2+X'7D2',R12  lab.first,R2  labdep.third,R2+X'8'

D-LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1995/03/26 08.00

                                    235 **********************************************************************  00189000
                                    236 *  Symbol Attribute enhancements                                     *  00190000
                                    237 **********************************************************************  00191000
0000CD C1C2C3                       239 SYMBOL1  DC    C'ABC'                                                    00193000
0000D0 12345C                       240 SYMBOL2  DC    P'123.45'                                                 00194000
                                    241 &VAR1    SETC  'SYMBOL1'                                                 00195000
                                    242 &VAR2    SETC  'SYMBOL2'                                                 00196000
0000D3 00
0000D4 4110 80A6           000A6    243          LA    1,=C'ABC'                                                 00197000
0000D8 4110 80A9           000A9    244          LA    1,=P'123.45'                                              00198000
                            1       246 &TYPE    SETC  T'=C'ABC'                                                 0020000
                                    247          DC    CL1'&TYPE'                                                00201000
0000DC C3                             +          DC    CL1'C'                                                    00201000
0000DD C3                           248          DC    AL1(T'SYMBOL1)                                            00202000
                                    249          DC    AL1(T'&VAR1)                                              00203000
0000DE C3                             +          DC    AL1(T'SYMBOL1)                                            00203000
0000DF C3                           250          DC    AL1(T'=C'ABC')                                            00204000
                            2       251 &LEN     SETA  L'=C'ABC'                                                 0020500
                                    252          DC    AL1(&LEN)                                                 00206000
0000E0 03                             +          DC    AL1(3)                                                    00206000
0000E1 03                           253          DC    AL1(L'SYMBOL1)                                            00207000
                                    254          DC    AL1(L'&VAR1)                                              00208000
0000E2 03                             +          DC    AL1(L'SYMBOL1)                                            00208000
0000E3 03                           255          DC    AL1(L'=C'ABC')                                            00209000
                            3       256 &INT     SETA  I'=P'123.45'                                              0021000
                                    257          DC    AL1(&INT)                                                 00211000
0000E4 03                             +          DC    AL1(3)                                                    00211000
0000E5 03                           258          DC    AL1(I'SYMBOL2)                                            00212000
                                    259          DC    AL1(I'&VAR2)                                              00213000
0000E6 03                             +          DC    AL1(I'SYMBOL2)                                            00213000
0000E7 03                           260          DC    AL1(I'=P'123.45')                                         00214000
                            4       261 &SCALE   SETA  S'=P'123.45'                                              0021500
                                    262          DC    AL1(&SCALE)                                               00216000
0000E8 02                             +          DC    AL1(2)                                                    00216000
0000E9 02                           263          DC    AL1(S'SYMBOL2)                                            00217000
                                    264          DC    AL1(S'&VAR2)                                              00218000
0000EA 02                             +          DC    AL1(S'SYMBOL2)                                            00218000
0000EB 02                           265          DC    AL1(S'=P'123.45')                                         00219000
                                    266          end                                                             00220000
000090 00000044FFFFFFE8     5       267                =a(a5,x)
000098 F1A2A34040404040             268                =cl8'1st'
0000A0 F2958440                     269                =cl4'2nd'
0000A4 F384                         270                =cl2'3d'
0000A6 C1C2C3                       271                =C'ABC'
0000A9 12345C                       272                =P'123.45'
```

1 The Type attribute (T') is allowed for ordinary symbols, SETC symbols, and literals, in both conditional assembly instructions and machine or assembler instructions. It is allowed in both open code and macro definitions.

2 The Length attribute (L') is allowed for ordinary symbols, SETC symbols, and literals, in both conditional assembly instructions and machine or assembler instructions. It is allowed in both open code and macro definitions.

3 The Integer attribute (I') is allowed for ordinary symbols, SETC symbols, and literals, in both conditional assembly instructions and machine or assembler instructions. It is allowed in both open code and macro definitions.

4 The Scale attribute (S') is allowed for ordinary symbols, SETC symbols, and literals, in both conditional assembly instructions and machine or assembler instructions. It is allowed in both open code and macro definitions.

5 If there are literals outstanding when the END statement is encountered, they are assigned to the LOCTR now in effect for the first control section in the assembly. This may or may not put the literals at the end of the first control section. In this sample assembly, the first control section, A, has two LOCTRs, A and DEECEES. Because A is active (at statement 183), the literals are assembled there. You control placement of literal pools by means of the LTORG statement. Note that X‘FFFFFFE8’ is used for the contents of A(X), statement 265. The symbol X was assigned the value (4*-6) by an EQU in statement 62.

```
BIGNAME                                    RELOCATION DICTIONARY                                  PAGE   12

   POS.ID       REL.ID     FLAGS   ADDRESS                                          HLASM R2.0  1995/03/26 08.00

  00000001     00000001     0C     00000090
  00000001     00000001     08     000000B1

 BIGNAME                        ORDINARY SYMBOL AND LITERAL CROSS REFERENCE                        PAGE   13

 SYMBOL      LEN     VALUE     ID    R TYPE    DEFN  REFERENCES                          HLASM R2.0  1995/03/26 08.00

 a        00000001 00000000 00000001     J        2  30, 181, 183
 area1    00000004 000007F4 00000002     F      205  195
 area1a   00000008 000007F4 00000002     C      206
 area1b   00000008 000007FC 00000002     C      207
 area2    00000004 00000804 00000002     F      208  196
 area2a   00000008 00000804 00000002     C      209
 area2b   00000008 0000080C 00000002     C      210
 a5       00000002 00000044 00000001     I       36  40, 267
 a7       00000016 00000048 00000001     L       38
 a8       00000002 000000B0 00000001 C   L       40  116M
 b5       00000008 000000B0 00000001     W      165  40, 116M, 165
 chara    00000001 00000081 FFFFFFFD     U      229
 constant 00000004 000000AC 00000001     F       29  27, 116M
 deecees  00000001 000000AC 00000001     J       28  164, 182
 fifty    00000001 00000032 FFFFFFFD     U      228
 first    00000001 00000000 FFFFFFFF     J      211  197U, 198U
 first1   00000008 00000000 FFFFFFFF     C      212  201M, 202M
 first2   00000008 00000008 FFFFFFFF     C      213  199U, 200
 hundred  00000001 00000064 FFFFFFFD     U      225
 lab                        00000002 A   U      198  200U, 202
 labdep                     00000002 A   U      200  204
 pd2      00000001 00000000 00000002     J      186
 reallylongsymbol
          00000004 00000084 00000001     I      116  16, 20
 second   00000001 00000000 FFFFFFFE     J      214  199U
 second1  00000004 00000000 FFFFFFFE     C      215  203M
 second2  00000004 00000004 FFFFFFFE     C      216
 SUSQUEHANNA
          00000002 0000007E 00000001     I      114
 SYMBOL1  00000003 000000CD FFFFFFFD     C      239  248, 249, 253, 254
 SYMBOL2  00000003 000000D0 FFFFFFFD     P      240  258, 259, 263, 264
 third    00000001 00000000 FFFFFFFD     J      217  200U
 third1   00000002 00000000 FFFFFFFD     C      218  204M
 third2   00000002 00000002 FFFFFFFD     C      219
 TRANSYLVANIA
          00000004 00000080 00000001     I      115  16, 20
 X        00000001 FFFFFFE8 00000001 A   U       62  267
 =a(a5,x) 00000004 00000090 00000001     A      267  115
 =C'ABC'  00000003 000000A6 00000001     C      271  243, 250, 255
 =cl2'3d' 00000002 000000A4 00000001     C      270  204
 =cl4'2nd'
          00000004 000000A0 00000001     C      269  203
 =cl8'1st'
          00000008 00000098 00000001     C      268  201, 202
 =P'123.45'
          00000003 000000A9 00000001     P      272  244, 260, 265
```

```
 BIGNAME                                   DSECT CROSS REFERENCE                                   PAGE   15

 DSECT     LENGTH      ID       DEFN                                                 HLASM R2.0  1995/03/26 08.00

 first    00000010  FFFFFFFF     211
 second   00000008  FFFFFFFE     214
 third    000000EC  FFFFFFFD     217
```

```
 BIGNAME                          DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                             PAGE   16
                                                                                                    HLASM R2.0  1995/03/26 08.00

      NO STATEMENTS FLAGGED IN THIS ASSEMBLY

 HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 2.0

 SYSTEM: CMS 7                     JOBNAME: (NOJOB)     STEPNAME: (NOSTEP)   PROCSTEP: (NOPROC)

DATASETS ALLOCATED FOR THIS ASSEMBLY
 CON DDNAME   DATASET NAME                              VOLUME(S)           MEMBER(S)
  P1 SYSIN    ASMASAMP ASSEMBLE F1                      CJM191
  L1 SYSLIB   CJMMAC   MACLIB   A1                      CJM191
  L2          OSMACRO  MACLIB   S2                      MNT190
     SYSLIN   ASMASAMP TEXT     A1                      CJM191
     SYSPRINT ASMASAMP LISTING  A1                      CJM191

    4028K ALLOCATED TO BUFFER POOL,          200K WOULD BE REQUIRED FOR THIS TO BE AN IN-STORAGE ASSEMBLY
     220 PRIMARY INPUT RECORDS READ         2215 LIBRARY RECORDS READ          0 WORK FILE READS
     416 PRIMARY PRINT RECORDS WRITTEN        11 PUNCH RECORDS WRITTEN         0 WORK FILE WRITES
       0 ADATA RECORDS WRITTEN
 ASSEMBLY START TIME: 08:00:02 STOP TIME: 08:00:10 PROCESSOR TIME: 00.00.00.0011
 RETURN CODE 000
```

Appendix F. MHELP Sample Macro Trace and Dump

The macro trace and dump (MHELP) facility is a useful means of debugging macro definitions. MHELP can be used anywhere in the source program or in macro definitions. MHELP is processed during macro generation. It is completely dynamic; you can branch around the MHELP statements by using AIF or AGO statements. Therefore, you can control its use by symbolic parameters and SET symbols. MHELP options remain in effect until superseded by another MHELP statement.

Figure 76 on page 265 shows a sample program that uses five functions of MHELP. The macro dumps and traces in the listing are highlighted, for example **1A** . Most dumps refer to statement numbers. When you call a library macro, the macro name is used instead of the statement number in the identification-sequence field. To get the statement numbers, you should use the LIBMAC assembler option or the COPY statement to copy the library definition into the source program before the macro call.

MHELP 1, Macro Call Trace: Item **1A** shows an outer macro call, **1B** an inner one. In each case, the amount of information given is short. This trace is given after successful entry into the macro; no dump is given if error conditions prevent an entry.

MHELP 2, Macro Branch Trace: This trace provides a one-line trace for each AGO and true AIF branch within a programmer macro. In any such branch, the "branched from" statement number, the "branched to" statement number, and the macro name are included. Note, in example **2A** , the "branched to" statement number indicated is not that of the ANOP statement bearing the target sequence symbol but that of the statement following it. The branch trace facility is suspended when library macros are expanded and MHELP 2 is in effect. To obtain a macro branch trace for such a macro, use the LIBMAC assembler option or insert a COPY "macro-name" statement in the source deck at some point before the MHELP 2 statement of interest.

MHELP 4, Macro AIF Dump: Items **4A** , **4B** , **4C** , **4D** , and **4E** , are examples of these dumps. Each dump includes a complete set of unsubscripted SET symbols with values. This list covers all unsubscripted variable symbols that appear in the same field of a SET statement in the macro definition. Values of elements of dimensioned SET symbols are not displayed.

MHELP 8, Macro Exit Dump: Items **8A** , and **8B** are examples of these dumps. This option provides a dump of the same group of SET symbols as are included in the macro AIF dump when an MEXIT or MEND is encountered.

Local and global variable symbols are not displayed at any point unless they appear in the current macro explicitly as SET symbols.

MHELP 16, Macro Entry Dump: This option provides a dump of the values of system variable symbols and symbolic parameters at the time the macro is called. The following numbering system is used:

Number	Item
000	&SYSNDX
001	&SYSECT
002	&SYSLOC
003	&SYSTIME
004	&SYSDATE
005	&SYSASM
006	&SYSVER
007	&SYSDATC
008	&SYSJOB
009	&SYSSTEP
010	&SYSSTYP
011	&SYSSTMT
012	&SYSNEST
013	&SYSSEQF
014	&SYSOPT_DBCS
015	&SYSOPT_OPTABLE
016	&SYSOPT_RENT
017	&SYSTEM_ID
018	&SYSIN_DSN
019	&SYSIN_MEMBER
020	&SYSIN_VOLUME
021	&SYSLIB_DSN
022	&SYSLIB_MEMBER
023	&SYSLIB_VOLUME
024	&SYSPRINT_DSN
025	&SYSPRINT_MEMBER
026	&SYSPRINT_VOLUME
027	&SYSTERM_DSN
028	&SYSTERM_MEMBER
029	&SYSTERM_VOLUME
030	&SYSPUNCH_DSN
031	&SYSPUNCH_MEMBER
032	&SYSPUNCH_VOLUME
033	&SYSLIN_DSN
034	&SYSLIN_MEMBER
035	&SYSLIN_VOLUME
036	&SYSADATA_DSN
037	&SYSADATA_MEMBER
038	&SYSADATA_VOLUME
039	&SYSPARM
040	Name Field on Macro Instruction

If there are *numkw* keyword parameters, they follow in order of appearance on the prototype statement:

Number	Item
041	1st keyword value
042	2nd keyword value

to

040+*numkw* *numkw*-th keyword value

If there are *numpp* positional parameters, they follow in order of appearance in the macro instruction:

Number	**Item**
041+*numkw*	1st positional parameter values
042+*numkw*	2nd positional parameter values

to

040+*numkw*+*numpp* *numpp*-th positional parameter values

For example, item **16A** has one keyword parameter (&OFFSET) and one positional parameter. The value of the keyword parameter appears opposite //0041, the positional parameter, opposite //0042. In both the prototype (statement 4) and the macro instruction (statement 55), the positional parameter appears in the first operand field, the keyword in the second. A length appears between the NUM and VALUE fields. A length of NUL indicates the corresponding item is empty.

Item **16B** shows an inner call containing zero keywords and two positional parameters.

MHELP 64, Macro Hex Dump: This option, when used in conjunction with the Macro AIF dump, the Macro Exit dump or the Macro Entry dump, dumps the parameter and SETC symbol values in EBCDIC and hexadecimal formats.

The hexadecimal dump precedes the EBCDIC dump, and dumps the full value of the symbol. System parameters are not dumped in hexadecimal.

MHELP 128, MHELP Suppression: This option suppresses all the MHELP options that are active at the time.

MHELP Control on &SYSNDX: The maximum value of the &SYSNDX system variable can be controlled by the MHELP instruction. The limit is set by specifying a number in the operand of the MHELP instruction, that is not one of the MHELP codes defined above, and is in the following number ranges:

- 256 to 65535
- Most numbers in the range 65792 to 9999999. Details for this number range are descibed in the *Language Reference*.

When the &SYSNDX limit is reached, message `ASMA013S ACTR counter exceeded` is issued, and the assembler, in effect, ignores all further macro calls. Refer to the *Language Reference* for further information.

```
                                                                                                                PAGE    3
  ACTIVE USINGS: NONE

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    HLASM R2.0  1995/03/26 08.00

000000                                1          CSECT
                                      2 *        COPY LNSRCH
                                      3          MACRO
                                      4 &NAME    LNSRCH &ARG,&OFFSET=STNUMB-STCHAIN
                                      5          LCLC   &LABEL
                                      6 &LABEL   SETC   'A&SYSNDX'              GENERATE SYMBOL
                                      7          AIF    (T'&NAME EQ 'O').SKIP
                                      8 &LABEL   SETC   '&NAME'                 IF MACRO CALL HAS LABEL, USE IT
                                      9 .SKIP    ANOP                           INSTEAD OF GENERATED SYMBOL
                                     10 &LABEL   LA     0,&OFFSET               LOAD REG. 0
                                     11          SCHI   &ARG,0(1)               SEARCH
                                     12          BC     1,&LABEL                IF MAX REACHED, CONTINUE
                                     13          MEND
                                                                                                                PAGE    4
  ACTIVE USINGS: NONE

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    HLASM R2.0  1995/03/26 08.00

                                     15 *        COPY   SCHI
                                     16          MACRO
                                     17 &NM      SCHI   &COMP,&LIST
                                     18          LCLA   &CNT
                                     19          LCLC   &CMPADR
                                     20 &CNT     SETA   1
                                     21 &NM      STM    1,15,4(13)
                                     22 .TEST    ANOP
                                     23 &CMPADR  SETC   '&CMPADR'.'&COMP'(&CNT,1)
                                     24          AIF    ('&COMP'(&CNT,1) EQ '(').LPAR
                                     25 &CNT     SETA   &CNT+1
                                     26          AIF    (&CNT LT K'&COMP).TEST
                                     27 .NOLNTH  ANOP
                                     28          LA      3,&COMP     COMPARAND
                                     29          AGO     .CONTIN
                                     30 .LPAR    AIF     ('&COMP'(&CNT+1,1) EQ ',').FINISH
                                     31 &CNT     SETA    &CNT+1
                                     32          AIF     (&CNT LT K'&COMP).LPAR
                                     33          AGO     .NOLNTH
                                     34 .FINISH  ANOP
                                     35 &CMPADR  SETC    '&CMPADR'.'&COMP'(&CNT+2,K'&COMP-&CNT)
                                     36          LA      3,&CMPADR              COMPARAND SANS LENGTH
                                     37 .CONTIN  ANOP
                                     38          LA      1,&LIST                LIST HEADER
                                     39          MVC     &COMP,0(0)             DUMMY MOVE TO GET COMP LENGTH
                                     40          ORG     *-6                    CHANGE MVC TO MVI
                                     41          DC      X'92'                  MVI OPCODE
                                     42          ORG     *+1                    PRESERVE LENGTH AS IMMED OPND
                                     43          DC      X'D000'                RESULT IS MVI 0(13),L
                                     44          L       15,=V(SCHI)
                                     45          BALR    14,15
                                     46          LM      1,15,4(13)
                                     47          MEXIT
                                     48          MEND
```

Figure 76 (Part 1 of 8). Sample Program Using MHELP

```
                                                                                                                PAGE    5
  ACTIVE USINGS: NONE

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    HLASM R2.0  1995/03/26 08.00

000000                               50 TEST     CSECT
000000 05C0                          51          BALR  12,0
                  R:C 00002          52          USING *,12

                                     54          MHELP B'11111'
                                     55          LNSRCH LISTLINE,OFFSET=LISTLINE-LISTNEXT

                            1A          ++//MHELP CALL TO MACRO LNSRCH      DEPTH=001   SYSNDX=0000001   STMT=00055
```

Figure 76 (Part 2 of 8). Sample Program Using MHELP

```
                    16A   //MHELP ENTRY TO  LNSRCH   MODEL STMT=00000 DEPTH=001 SYSNDX=0000001 KWCNT=001
                          ////SYSTEM PARAMETERS:                                                         /
                          ////  0. SYSNDX            1. SYSECT            2. SYSLOC             /
                          ////  3. SYSTIME           4. SYSDATE           5. SYSASM             /
                          ////  6. SYSVER            7. SYSDATC           8. SYSJOB             /
                          ////  9. SYSSTEP          10. SYSSTYP          11. SYSSTMT            /
                          //// 12. SYSNEST          13. SYSSEQF          14. SYSOPT_DBCS        /
                          //// 15. SYSOPT_OPTABLE   16. SYSOPT_RENT      17. SYSTEM_ID          /
                          //// 18. SYSIN_DSN        19. SYSIN_MEMBER     20. SYSIN_VOL          /
                          //// 21. SYSLIB_DSN       22. SYSLIB_MEMBER    23. SYSYSLIB_VOLUME    /
                          //// 24. SYSPRINT_DSN     25. SYSPRINT_MEMBER  26. SYSPRINT_VOLUME    /
                          //// 27. SYSTERM_DSN      28. SYSTERM_MEMBER   29. SYSTERM_VOLUME     /
                          //// 30. SYSPUNCH_DSN     31. SYSPUNCH_MEMBER  32. SYSPUNCH_VOLUME    /
                          //// 33. SYSLIN_DSN       34. SYSLIN_MEMBER    35. SYSLIN_VOLUME      /
                          //// 36. SYSADATA_DSN     37. SYSADATA_MEMBER  38. SYSADATA_VOLUME    /
                          //// 39. SYSPARM          41. NAME                                    /
                          ////KEYWORD PARAMETERS; POSITIONAL PARAMETERS                                  /
                          //NUM  LNTH  VALUE (64 CHARS/LINE)
                          //0000  004  0001
                          //0001  004  TEST
                          //0002  004  TEST
                          //0003  005  08.00
                          //0004  008  02/28/95
                          //0005  020  HIGH LEVEL ASSEMBLER
                          //0006  005  1.2.0
                          //0007  008  19950228
                          //0008  007  CJMJOB
                          //0009  008  (NOSTEP)
                          //0010  005  CSECT
                          //0011  008  00000056
                          //0012  001  1
                          //0013  008
                          //0014  001  0
                          //0015  003  UNI
                          //0016  001  0
                          //0017  005  CMS 7
                          //0018  020  MHELP    ASSEMBLE A1
                          //0019  NUL
                          //0020  006  AT5191
                          //0021  020  MHELP    ASSEMBLE A1
                          //0022  NUL
                          //0023  006  AT5191
                          //0024  020  MHELP    LISTING  A1
                          //0025  NUL
                          //0026  006  AT5191
                          //0027  NUL
                          //0028  NUL
                          //0029  NUL
                          //0030  NUL
                          //0031  NUL
                          //0032  NUL
                          //0033  020  MHELP    TEXT     A1
                          //0034  NUL
                          //0035  006  AT5191
                          //0036  NUL
                          //0037  NUL
                          //0038  NUL
                          //0039  NUL
                          //0040  NUL
                          //0041  017  LISTLINE-LISTNEXT
                          //0042  008  LISTLINE


                    4A    //MHELP AIF IN    LNSRCH   MODEL STMT=00007 DEPTH=001 SYSNDX=0000001 KWCNT=001
                          ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                          //0001 LCLC       LABEL                                        LNTH= 005
                          //     VAL=A0001
```

Figure 76 (Part 3 of 8). Sample Program Using MHELP

```
                                                                                              PAGE    6
  ACTIVE USINGS: TEST+X'2',R12

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                             HLASM R2.0  1995/03/26 08.00



                    2A    ++//MHELP  BRANCH FROM STMT 00007 TO STMT 00010 IN MACRO LNSRCH

000002 4100 0002            00002    56+A0001    LA     0,LISTLINE-LISTNEXT    LOAD REG. 0                    01-00010

                    1B    ++//MHELP CALL TO MACRO SCHI        DEPTH=002   SYSNDX=0000002   STMT=00011
```

Figure 76 (Part 4 of 8). Sample Program Using MHELP

```
                     16B      //MHELP ENTRY TO  SCHI     MODEL STMT=00000 DEPTH=002 SYSNDX=0000002 KWCNT=000
                              ////SYSTEM PARAMETERS:                                                       /
                              ////  0. SYSNDX            1. SYSECT            2. SYSLOC             /
                              ////  3. SYSTIME           4. SYSDATE           5. SYSASM             /
                              ////  6. SYSVER            7. SYSDATC           8. SYSJOB             /
                              ////  9. SYSSTEP          10. SYSSTYP          11. SYSSTMT            /
                              //// 12. SYSNEST          13. SYSSEQF          14. SYSOPT_DBCS        /
                              //// 15. SYSOPT_OPTABLE   16. SYSOPT_RENT      17. SYSTEM_ID          /
                              //// 18. SYSIN_DSN        19. SYSIN_MEMBER     20. SYSIN_VOL          /
                              //// 21. SYSLIB_DSN       22. SYSLIB_MEMBER    23. SYSYSLIB_VOLUME    /
                              //// 24. SYSPRINT_DSN     25. SYSPRINT_MEMBER  26. SYSPRINT_VOLUME    /
                              //// 27. SYSTERM_DSN      28. SYSTERM_MEMBER   29. SYSTERM_VOLUME     /
                              //// 30. SYSPUNCH_DSN     31. SYSPUNCH_MEMBER  32. SYSPUNCH_VOLUME    /
                              //// 33. SYSLIN_DSN       34. SYSLIN_MEMBER    35. SYSLIN_VOLUME      /
                              //// 36. SYSADATA_DSN     37. SYSADATA_MEMBER  38. SYSADATA_VOLUME    /
                              //// 39. SYSPARM          40. NAME                                    /
                              ////KEYWORD PARAMETERS; POSITIONAL PARAMETERS                                /
                              //NUM  LNTH  VALUE (64 CHARS/LINE)
                              //0000  004  0002
                              //0001  004  TEST
                              //0002  004  TEST
                              //0003  005  08.00
                              //0004  008  02/28/94
                              //0005  020  HIGH LEVEL ASSEMBLER
                              //0006  005  1.2.0
                              //0007  008  19950228
                              //0008  007  CJMJOB
                              //0009  008  (NOSTEP)
                              //0010  005  CSECT
                              //0011  008  00000057
                              //0012  001  2
                              //0013  008
                              //0014  001  0
                              //0015  003  UNI
                              //0016  001  0
                              //0017  005  CMS 7
                              //0018  020  MHELP    ASSEMBLE A1
                              //0019  NUL
                              //0020  006  AT5191
                              //0021  020  MHELP    ASSEMBLE A1
                              //0022  NUL
                              //0023  006  AT5191
                              //0024  020  MHELP    LISTING  A1
                              //0025  NUL
                              //0026  006  AT5191
                              //0027  NUL
                              //0028  NUL
                              //0029  NUL
                              //0030  NUL
                              //0031  NUL
                              //0032  NUL
                              //0033  020  MHELP    TEXT     A1
                              //0034  NUL
                              //0035  006  AT5191
                              //0036  NUL
                              //0037  NUL
                              //0038  NUL
                              //0039  NUL
                              //0040  NUL
                              //0041  008  LISTLINE
                              //0042  004  0(1)

000006 901F D004           00004    57+        STM    1,15,4(13)                                      02-00021

                     4B       //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                              ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                              //0001 LCLA       CNT                                        VAL=  0000000001
                              //0002 LCLC       CMPADR                                     LNTH= 001
```

Figure 76 (Part 5 of 8). Sample Program Using MHELP

```
                                                                                    PAGE    7
ACTIVE USINGS: TEST+X'2',R12

LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                              HLASM R2.0  1995/03/26 08.00

                                        //     VAL=L


                      4C                //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000002
                                        //0002 LCLC       CMPADR                                   LNTH= 001
                                        //     VAL=L


                      2B                ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                      4D                //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000002
                                        //0002 LCLC       CMPADR                                   LNTH= 002
                                        //     VAL=LI


                      4E                //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000003
                                        //0002 LCLC       CMPADR                                   LNTH= 002
                                        //     VAL=LI


                      2C                ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000003
                                        //0002 LCLC       CMPADR                                   LNTH= 003
                                        //     VAL=LIS


                                        //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000004
                                        //0002 LCLC       CMPADR                                   LNTH= 003
                                        //     VAL=LIS


                                        ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                      VAL=  0000000004
                                        //0002 LCLC       CMPADR                                   LNTH= 004
                                        //     VAL=LIST


                                        //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
```

Figure 76 (Part 6 of 8). Sample Program Using MHELP

```
                                                                                     PAGE    8
ACTIVE USINGS: TEST+X'2',R12

LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                          HLASM R2.0  1995/03/26 08.00

                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000005
                                        //0002 LCLC       CMPADR                               LNTH= 004
                                        //     VAL=LIST


                                        ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000005
                                        //0002 LCLC       CMPADR                               LNTH= 005
                                        //     VAL=LISTL


                                        //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000006
                                        //0002 LCLC       CMPADR                               LNTH= 005
                                        //     VAL=LISTL


                                        ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000006
                                        //0002 LCLC       CMPADR                               LNTH= 006
                                        //     VAL=LISTLI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000007
                                        //0002 LCLC       CMPADR                               LNTH= 006
                                        //     VAL=LISTLI


                                        ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI


                                        //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000007
                                        //0002 LCLC       CMPADR                               LNTH= 007
                                        //     VAL=LISTLIN


                                        //MHELP AIF IN    SCHI     MODEL STMT=00026 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                  VAL=  0000000008
                                        //0002 LCLC       CMPADR                               LNTH= 007
                                        //     VAL=LISTLIN
```

Figure 76 (Part 7 of 8). Sample Program Using MHELP

```
                                                                                                       PAGE    9
  ACTIVE USINGS: TEST+X'2',R12

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  HLASM R2.0  1995/03/26 08.00


00000A 4130 C024           00026    58+         LA      3,LISTLINE  COMPARAND                               02-00028

                                       ++//MHELP  BRANCH FROM STMT 00029 TO STMT 00038 IN MACRO SCHI

00000E 4111 0000           00000    59+         LA      1,0(1)               LIST HEADER                      02-00038
000012 D202 C024 0000 00026 00000   60+         MVC     LISTLINE,0(0)        DUMMY MOVE TO GET COMP LENGTH    02-00039
000018                     00012    61+         ORG     *-6                  CHANGE MVC TO MVI                02-00040
000012 92                           62+         DC      X'92'                MVI OPCODE                       02-00041
000013                     00014    63+         ORG     *+1                  PRESERVE LENGTH AS IMMED OPND    02-00042
000014 D000                         64+         DC      X'D000'              RESULT IS MVI 0(13),L            02-00043
000016 58F0 C02E           00030    65+         L       15,=V(SCHI)                                           02-00044
00001A 05EF                         66+         BALR    14,15                                                 02-00045
00001C 981F D004           00004    67+         LM      1,15,4(13)                                            02-00046

                             8A         //MHELP EXIT FROM SCHI     MODEL STMT=00047 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLA       CNT                                       VAL=  0000000008
                                        //0002 LCLC       CMPADR                                    LNTH= 007
                                        //     VAL=LISTLIN

000020 4710 C000           00002    68+         BC     1,A0001               IF MAX REACHED, CONTINUE         01-00012

                             8B         //MHELP EXIT FROM LNSRCH   MODEL STMT=00013 DEPTH=001 SYSNDX=0000001 KWCNT=001
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0001 LCLC       LABEL                                     LNTH= 005
                                        //     VAL=A0001

000024                              69 LISTNEXT DS     H
000026                              70 LISTLINE DS     FL3'0'
000030                              71          LTORG
000030 00000000                     72                =V(SCHI)
000000                              73          END    TEST
```

Figure 76 (Part 8 of 8). Sample Program Using MHELP

Appendix G. High Level Assembler Messages

High Level Assembler produces the following types of messages:

- Assembly error-diagnostic messages
- Assembly abnormal-termination messages
- ASMAHL command-error messages

The following section describes the format and placement of messages issued by the assembler. "Assembly Error Diagnostic Messages" on page 273, "Abnormal Assembly Termination Messages" on page 306, and "ASMAHL Command Error Messages" on page 310 list and describe each message.

Message Code Format

Assembly error diagnostic messages and assembly abnormal termination messages have the following message code format:

ASMA*nnns*

nnn a three-character message number

s severity indicator

The severity indicators, and the corresponding severity codes are:

I - Informational
(Severity code = 0)

This error does not affect the running of the program; rather it is a coding inefficiency or other such condition that can be changed. The assembler has not detected any conditions affecting the correctness of the program.

N - Notice
(Severity code = 2)

This type of message brings your attention to a condition that you might wish to correct. The assembler has not detected any conditions affecting the correctness of the program, however, the output from the assembly might not be what you expect.

W - Warning
(Severity code = 4)

Although the statement in which the condition occurs is syntactically correct, it has the potential for causing an error when the program is run.

E - Error
(Severity code = 8)

The condition is definitely an error. However, the assembler has tried to correct the error, or has ignored the statement in error. The program probably will not run successfully.

S - Severe
(Severity code = 12)

The condition is a serious error. The assembler has either ignored the statement in error, or the machine instruction has been assembled to zero. It is not likely that the program will assemble as expected or that it will run.

C - Critical
(Severity code = 16)

The condition is a critical error. It is not likely that the program will run successfully.

U - Unrecoverable
(Severity code = 20)

The error condition is of such magnitude that the assembler could not continue.

ASMAHL command error messages have the following message code format:

ASMACMS*nnn*E

nnn a three-character message number

E simply indicates an error. In some cases the assembly will proceed after the message has been issued.

LANGUAGE Assembler Option: The text of ASMAHL command error messages is produced in the language specified on the LANGUAGE operand in the installation default options.

Message Descriptions

Each message entry for assembly error diagnostic messages and assembly abnormal termination messages has the following five sections:

- Message Number and Text
- Explanation of Message
- System Action
- Programmer Response
- Severity Code

Each message entry for ASMAHL command error messages has up to five of the following sections:

- Message Number and Text
- Explanation of Message
- Supplemental Information
- System Action
- Programmer Response

Message Number and Text: Only the message number and the major fixed portion of the message text are included in the message description. Any abbreviations in actual message text are described under the message explanation section. Unused message numbers account for the gaps in the message number sequence. No messages are defined for numbers, such as `ASMA006`, not included in this section.

Explanation of Message: There is more than one explanation for some messages, because different sections of the assembler can generate the same message. Several assembler termination messages have identical explanations.

Supplemental Information: For ASMAHL command error messages, the supplemental information describes the possible contents of the variables in the message text.

System Action: This section describes how the assembler handles statements with errors. Some actions include:

- A machine instruction assembles as all zeros
- An assembler instruction is usually ignored; it is printed but has no effect on the assembly. Many assembler instructions, however, are partially processed or processed with a default value. For some instructions, the operands preceding the operand in error or every operand except the operand in error is processed. For example, if one of several operands on a DROP statement is a symbol that cannot be evaluated to a register number, only that operand is ignored. All the correctly specified registers are correctly processed.
- For some assembler statements, especially macro prototype and conditional assembly statements, the operand or term in error is given a default value. Thus the statement assembles completely, but will probably cause incorrect results if the program is run.

For ASMAHL command error messages this section describes the command return code, and the status of the system after the error.

Programmer Response: Many errors have specific or probable causes. In such a case, the Programmer Response section gives specific steps for fixing the error. Most messages, however, have too many possible causes (from keying errors to wrong use of the statement) to list. The programmer response section for these error messages does not give specific directions. The cause of most such errors can be determined from the message text and the explanation.

Severity Code: The level of severity code indicates how critical the error might be. The severity codes and their meanings are described in “Message Code Format” on page 271. ASMAHL command error messages do not have a severity code, although each message issued by the ASMAHL command, and that causes the assembly to terminate has a return code higher than 20.

The severity code is used to determine the return code issued by the assembler when it returns control to the operating system. The IBM-supplied cataloged procedures (for MVS) include a COND parameter on the linkage edit and run steps. The COND parameter prevents the running of these steps if the return code from the assembler is greater than 8. Thus errors with `*** ERROR ***` in the message and a severity code of `S` prevent the assembled program from linkage editing or running. Errors with `*** ERROR ***` in the message and a severity code of `E` and errors with `** WARNING **` in the message do not prevent the assembled program from linkage editing or running.

Assembly Error Diagnostic Messages

High Level Assembler prints most error messages in the listing immediately following the statements in error. It also prints the total number of flagged statements and their statement numbers in the *Diagnostic Cross Reference and Assembler Summary* section of the assembler listing.

The messages do not follow the statement in error when:

- Errors are detected during editing of macro definitions read from a library. A message for such an error appears after the first call in the source program to that macro definition. You can, however, bring the macro definition into the source program with a COPY statement. The editing error messages will then be attached to the statements in error.
- Errors are detected by the lookahead function of the assembler. (For attribute references, lookahead processing scans statements after the one being assembled.). Messages for these errors appear after the statements in which they occur. The messages might also appear at the point at which lookahead was called.
- Errors are detected on conditional assembly statements during macro generation or MHELP testing. Such a message follows the most recently generated statement or MHELP output statement.

A typical error diagnostic messsage is:

```
ASMA057E ***ERROR***  Undefined operation
         code-xxxxx
```

The term ***ERROR*** is part of the message if the severity code is 8 or greater. The term **WARNING** is part of the message if the severity code is 0 to 4.

A copy of a segment of the statement in error, represented above by xxxxx, is inserted into many messages. Normally this segment begins at the bad character or term. For some errors, however, the segment might begin after the bad character or term. The segment might include part of the remarks field.

If a diagnostic message follows a statement generated by a macro definition, the following items might be appended to the error message:

- The number of the model statement in which the error occurred, or the first five characters of the macro name.
- The SET symbol, parameter number, or value string associated with the error.

References to macro parameters are by number (such as PARAM00044) instead of by name. The first forty one numbers are always assigned for the standard system parameters as follows:

```
PARAM00000 =  &SYSNDX
PARAM00001 =  &SYSECT
PARAM00002 =  &SYSLOC
PARAM00003 =  &SYSTIME
PARAM00004 =  &SYSDATE
PARAM00005 =  &SYSASM
PARAM00006 =  &SYSVER
PARAM00007 =  &SYSDATC
PARAM00008 =  &SYSJOB
PARAM00009 =  &SYSSTEP
PARAM00010 =  &SYSSTYP
PARAM00011 =  &SYSSTMT
PARAM00012 =  &SYSNEST
PARAM00013 =  &SYSSEQF
PARAM00014 =  &SYSOPT_DBCS
PARAM00015 =  &SYSOPT_OPTABLE
PARAM00016 =  &SYSOPT_RENT
PARAM00017 =  &SYSTEM_ID
PARAM00018 =  &SYSIN_DSN
PARAM00019 =  &SYSIN_MEMBER
PARAM00020 =  &SYSIN_VOLUME
PARAM00021 =  &SYSLIB_DSN
PARAM00022 =  &SYSLIB_MEMBER
PARAM00023 =  &SYSLIB_VOLUME
PARAM00024 =  &SYSPRINT_DSN
PARAM00025 =  &SYSPRINT_MEMBER
PARAM00026 =  &SYSPRINT_VOLUME
PARAM00027 =  &SYSTERM_DSN
PARAM00028 =  &SYSTERM_MEMBER
PARAM00029 =  &SYSTERM_VOLUME
PARAM00030 =  &SYSPUNCH_DSN
PARAM00031 =  &SYSPUNCH_MEMBER
PARAM00032 =  &SYSPUNCH_VOLUME
PARAM00033 =  &SYSLIN_DSN
PARAM00034 =  &SYSLIN_MEMBER
PARAM00035 =  &SYSLIN_VOLUME
PARAM00036 =  &SYSADATA_DSN
PARAM00037 =  &SYSADATA_MEMBER
PARAM00038 =  &SYSADATA_VOLUME
PARAM00039 =  &SYSPARM
PARAM00040 =  Name Field Parameter
```

After these, the keyword parameters are numbered in the order defined in the macro definition, followed by positional parameters. When there are no keyword parameters in the macro definition, PARAM00041 refers to the first positional parameter.

If a diagnostic message follows a conditional assembly statement in the source program, the following items are be appended to the error message:

- The word 'OPENC', meaning 'open code'
- The SET symbol or value string associated with the error

Several messages might be issued for a single statement or even for a single error within a statement. This happens because each statement is usually evaluated on more than one level (for example, term level, expression level, and operand level) or by more than one phase of the assembler. Each level or phase can diagnose errors; therefore, most or all of the errors in the statement are flagged. Occasionally, duplicate error messages might occur. This is a normal result of the error-detection process.

Message Not Known

The following message might appear in a listing:

```
ASMA000S ***ERROR*** Message not known - nnn
```

The statement preceding this message contains an error but the assembler routine that detected the error issued the number (*nnn*) of a nonexistent error message to the assembler's message generation routine. If you can correct the error, this statement will assemble correctly. However, this message indicates an error in the error detection process of the assembler. Save the output and the source deck from this assembly and report the problem to your IBM service representative.

Messages

ASMA001 Operation code not allowed to be generated - *xxxxxxxx*

Explanation: An attempt was made to produce a restricted operation code by variable symbol substitution. Restricted operation codes are:

```
ACTR    AGO     AGOB    AIF
AIFB    ANOP    AREAD   COPY
GBLA    GBLB    GBLC    ICTL
LCLA    LCLB    LCLC    MACRO
MEND    MEXIT   REPRO   SETA
SETAF   SETB    SETC    SETCF
```

System Action: The statement is ignored.

Programmer Response: If you want a variable operation code, use AIF to branch to the correct unrestricted statement.

Severity: 8

ASMA002 Generated statement too long; statement truncated

Explanation: The statement generated by a macro definition is more than 1728 characters long.

System Action: The statement is truncated; the leading 1728 characters are retained.

Programmer Response: Shorten the statement.

Severity: 12

ASMA003 Undeclared variable symbol; default=0, null, or type=U

Explanation: A variable symbol in the operand field of the statement has not been declared (defined) in the name field of a SET statement, in the operand field of an LCL or GBL statement, or in a macro prototype statement.

System Action: The variable symbol is given a default value as follows:

```
SETA = 0
SETB = 0
SETC = null (empty) string
```

The type attribute (T') of the variable is given a default value of U (undefined).

Programmer Response: Declare the variable *before* you use it as an operand.

Severity: 8

ASMA004 Duplicate SET symbol declaration; first is retained - *xxxxxxxx*

Explanation: A SET symbol has been declared (defined) more than once. A SET symbol is declared when it is used in the name field of a SET statement, in the operand field of an LCL or GBL statement, or in a macro prototype statement.

System Action: The value of the first declaration of the SET symbol is used.

Programmer Response: Eliminate the incorrect declarations.

Severity: 8

ASMA005 No storage for macro call; continue with open code

Explanation: An inner macro call could not be processed because no main storage was available.

System Action: The assembly continues with the next open code statement.

Programmer Response: Check whether the macro is recursive, and, if so, whether termination is provided for; correct the macro if necessary. If the macro is correct, allocate more main storage.

Severity: 12

ASMA007 Previously defined sequence symbol - *xxxxxxxx*

Explanation: The sequence symbol in the name field has been used in the name field of a previous statement.

System Action: The first definition of the sequence symbol is used; this definition is ignored.

Programmer Response: Remove or change one of the sequence symbols.

Severity: 12

ASMA008 Previously defined symbolic parameter

Explanation: The same variable symbol has been used to define two different symbolic parameters.

System Action: When the parameter name (the variable symbol) is used inside the macro definition, it refers to the *first* definition of the parameter in the prototype. However, if the second parameter defined by the variable symbol is a positional parameter, the count of positional operands still increases by one. The second parameter can then be referred to only through use of &SYSLIST.

Programmer Response: Change one of the parameter names to another variable symbol.

Severity: 12

ASMA009 System variable symbol illegally re-defined

Explanation: A system variable symbol has been used in the name field of a macro prototype statement. The system variable symbols are:

```
&SYSADATA_DSN     &SYSLIN_DSN       &SYSPUNCH_DSN
&SYSADATA_MEMBER  &SYSLIN_MEMBER    &SYSPUNCH_MEMBER
&SYSADATA_VOLUME  &SYSLIN_VOLUME    &SYSPUNCH_VOLUME
&SYSASM           &SYSLIST          &SYSSEQF
&SYSDATC          &SYSLOC           &SYSSTEP
&SYSDATE          &SYSNDX           &SYSSTMT
&SYSECT           &SYSNEST          &SYSSTYP
&SYSIN_DSN        &SYSOPT_DBCS      &SYSTEM_ID
&SYSIN_MEMBER     &SYSOPT_OPTABLE   &SYSTERM_DSN
&SYSIN_VOLUME     &SYSOPT_RENT      &SYSTERM_MEMBER
&SYSJOB           &SYSPARM          &SYSTERM_VOLUME
&SYSLIB_DSN       &SYSPRINT_DSN     &SYSTIME
&SYSLIB_MEMBER    &SYSPRINT_MEMBER  &SYSVER
&SYSLIB_VOLUME    &SYSPRINT_VOLUME
```

System Action: The name parameter is ignored. The name on a corresponding macro instruction is not generated.

Programmer Response: Change the parameter to one that is not a system variable symbol.

Severity: 12

ASMA010 Invalid use of symbol qualifier - *xxxxxxxx*

Explanation: One of the following has occurred:

- A symbol qualifier has been used to qualify a symbol in other than:
  - A machine instruction
  - The nominal value of an S-type address constant
  - The supporting address operand of a dependent USING statement
- A symbol qualifier is used to qualify a symbol that has an absolute value where a symbol that represents a relocatable address is required
- A symbol qualifier is used to qualify a symbol that is not within the range of the corresponding labeled USING statement
- A symbol qualifier is used to qualify an undefined symbol
- A symbol qualifier is used to qualify an incorrect symbol
- A period is used as the last character of a term, but the symbol preceding the period has not been defined in the name field of a labeled USING statement

A symbol qualifier can only be used in machine instructions, the nominal value of S-type address constants, or the second operand (supporting base address) of dependent USING instructions. A symbol qualifier can only be used to qualify symbols that are within the range of the corresponding labeled USING.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored. If there is a further error in the statement, a message that describes the error is issued.

Programmer Response: Correct the use of the symbol qualifier, or check the statement for the error indicated in the following message.

Severity: 8

ASMA011 Inconsistent global declarations; first is retained

Explanation: A global SET variable symbol has been defined in more than one macro definition or in a macro definition and in the source program, and the two definitions are inconsistent in type or dimension.

System Action: The first definition encountered is retained.

Programmer Response: Assign a new SET symbol or make the definitions compatible.

Severity: 8

ASMA012 Undefined sequence symbol - ***xxxxxxxx*****; macro aborted**

Explanation: A sequence symbol in the operand field is not defined; that is, it is not used in the name field of a model statement.

System Action: Exit from the macro definition.

Programmer Response: Define the sequence symbol or correct the reference to it.

Severity: 12

ASMA013 ACTR counter exceeded

Explanation: The conditional assembly loop counter (set by an ACTR statement) has been decremented to zero. The ACTR counter is decremented by one each time an AIF or AGO branch is processed successfully. The counter is halved for most errors encountered by the macro editor phase of the assembler.

System Action: Any macro expansion stops. If the ACTR statement is in the source program, the assembly stops.

Programmer Response: Check for an AIF/AGO loop or another type of error. (You can use the MHELP facility, described in Chapter 6, “Diagnosing Assembly Errors” on page 123 and Appendix F, “MHELP Sample Macro Trace and Dump” on page 262, to trace macro definition logic.) If there is no error, increase the initial count on the ACTR instruction.

Severity: 12

ASMA014 Irreducible qualified expression

Explanation: The statement cannot be resolved because two or more qualified symbols are used in a complex relocatable expression, or two or more qualified symbols with different symbol qualifiers are paired in an absolute expression.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored.

Programmer Response: Supply an absolute expression, or correct the qualified symbol in error.

Severity: 8

ASMA017 Undefined keyword parameter; default to positional, including keyword

Explanation: A keyword parameter in a macro call is not defined in the corresponding macro prototype statement.

This message is also generated by a valid positional parameter that contains an equal sign.

System Action: The keyword (including the equals sign and value) is used as a positional parameter.

Programmer Response: Define the keyword in the prototype statement, or enclose the valid positional parameter in parentheses, or single quotation marks, and adjust the macro coding appropriately.

Severity: 4

ASMA018 Duplicate keyword in macro call; last value is used

Explanation: A keyword operand occurs more than once in a macro call.

System Action: The latest value assigned to the keyword is used.

Programmer Response: Eliminate one of the keyword operands.

Severity: 12

ASMA020 Illegal GBL or LCL statement

Explanation: A global (GBL) or local (LCL) declaration statement does not have an operand.

System Action: The statement is ignored.

Programmer Response: Remove the statement or add an operand.

Severity: 8

ASMA021 Illegal SET statement

Explanation: The operand of a SETB statement is not 0, 1, or a SETB expression enclosed in parentheses.

System Action: The statement is ignored.

Programmer Response: Correct the operand or delete the statement.

Severity: 8

ASMA023 Symbolic parameter too long

Explanation: A symbolic parameter in this statement is too long. It must not exceed 63 characters, including the initial ampersand.

System Action: The symbolic parameter and any operand following it in this statement are ignored.

Programmer Response: Make sure all symbolic parameters consist of an ampersand followed by 1 to 62 alphanumeric characters, the first of which is alphabetic.

Severity: 8

ASMA024 Invalid variable symbol

Explanation: One of these errors has occurred:

- A symbolic parameter or a SET symbol is not an ampersand followed by 1 to 62 alphanumeric characters, the first being alphabetic.
- A created SET symbol definition is not a valid SET symbol expression enclosed in parentheses.

System Action: The statement is ignored.

Programmer Response: Supply a valid symbol or expression.

Severity: 8

ASMA025 Invalid macro prototype operand

Explanation: The format of the operand field of a macro prototype statement is not correct. For example, two parameters are not separated by a comma, or a parameter contains characters that are not permitted.

System Action: The operand field of the prototype is ignored.

Programmer Response: Supply a valid operand field.

Severity: 12

ASMA026 Macro call operand too long; 255 leading characters deleted

Explanation: An operand of a macro instruction is more than 255 characters long.

System Action: The leading 255 characters are deleted.

Programmer Response: Limit the operand to 255 characters, or limit it to two or more operands.

Severity: 12

ASMA027 Excessive number of operands

Explanation: One of the following has occurred:

- More than 32000 positional operands, keyword operands, or both have been explicitly defined in a macro prototype statement.
- There are more than 255 operands in a DC, DS, or DXD statement.

System Action: The excess parameters are ignored.

Programmer Response: For a DC, DS, or DXD statement, use more than one statement. For a macro prototype statement, delete the extra operands and use &SYSLIST to access the positional operands, or redesign the macro definition.

Severity: 12

ASMA028 Invalid displacement

Explanation: One of the following has occurred:

- The displacement field of an explicit address is not an absolute value within the range 0 through 4095.
- The displacement field of an S-type address constant is not an absolute value within the range 0 through 4095.

System Action: The statement or constant assembles as zero.

Programmer Response: Correct the displacement or supply a correct USING statement containing an absolute first operand before this statement.

Severity: 8

ASMA029 Incorrect register or mask specification

Explanation: The value specifying a register or a mask is not an absolute value within the range 0 through 15; an odd register is used where an even register is required; a register is used where none can be specified; or a register is not specified where one is required.

System Action: For machine instructions and S-type address constants, the statement or constant assembles as zero. For USING and DROP statements, the incorrect register operand is ignored.

Programmer Response: Specify a valid register.

Severity: 8

ASMA030 Invalid literal usage

Explanation: A literal is used in an assembler instruction, another literal, or a field of a machine instruction where it is not permitted.

System Action: An assembler instruction containing a literal is generally ignored and another message, relative to the operation code of the instruction, appears. A machine instruction assembles as zero.

Programmer Response: If applicable, replace the literal with the name of a DC statement.

Severity: 8

ASMA031 Invalid immediate field

Explanation: The value of an immediate operand of a machine instruction requires more than one byte of storage (exceeds 255) or the value of the immediate operand exceeds 9 on an SRP instruction.

System Action: The instruction assembles as zero.

Programmer Response: Use a valid immediate operand, or specify the immediate information in a DC statement or a literal and change the statement to a nonimmediate type.

Severity: 8

ASMA032 Relocatable value found when absolute value required

Explanation: One of the following has occurred:

- A relocatable or complex relocatable expression is used where an absolute expression is required.
- A DSECT-based expression is used as an operand for an address constant where an expression that resolves into a storage address is required.

System Action: A machine instruction assembles as zero. In a DC, DS, or DXD statement, the operand in error and the following operands are ignored.

Programmer Response: Supply an absolute expression or term, or for an address constant supply a valid storage address expression.

Severity: 8

ASMA033 Alignment error

Explanation: An address referenced by this statement might not be aligned to the correct boundary for this instruction; for example, the data referenced by a load instruction (L) might be on a halfword boundary, or the aligned address might depend upon an index register.

System Action: The instruction assembles as written.

Programmer Response: Correct the operand if it is in error. If you are using System/370 architecture that does not require alignment, or you want to suppress alignment checking for some other reason, you can specify the NOALIGN assembler option. If a particular statement is correct, you can suppress this message by writing the statement with an absolute displacement and an explicit base register, as in this example:

```
L   1,SYM-BASE(,2)
```

Severity: 4

ASMA034 Addressability error

Explanation: The address referenced by this statement does not fall within the range of a USING statement, or a base register is specified along with a relocatable displacement.

System Action: The instruction assembles as zero.

Programmer Response: Insert an applicable USING statement before this statement. Otherwise, check this statement for a misspelled symbol, an unintended term or symbol in an address expression, or a relocatable symbol used as a displacement.

Severity: 8

ASMA035 Invalid delimiter - *xxxxxxxx*

Explanation:

1. A required delimiter in a DC, DS, or DXD statement is missing or appears where none should be; the error might be any of these:
   - A quotation mark with an address constant.
   - A left parenthesis with a nonaddress constant.
   - A constant field not started with a quotation mark, left parenthesis, blank, or comma.
   - An empty constant field in a DC.
   - A missing comma or right parenthesis following an address constant.
   - A missing subfield right parenthesis in an S-type address constant.
   - A missing right parenthesis in a constant modifier expression.
2. A parameter in a macro prototype statement was not followed by a valid delimiter: comma, equal sign, or blank.
3. The DBCS option is on, and SO follows a variable symbol without an intervening period.

System Action: The operand or parameter in error and the following operands or parameters are ignored.

Programmer Response: Supply a valid delimiter.

Severity: 12

ASMA036 Reentrant check failed

Explanation: A machine instruction that might store data into a control section or common area when run has been detected. This message is generated only when reentrant checking is requested by the assembler option 'RENT' or within an RSECT.

System Action: The statement assembles as written.

Programmer Response: If you want reentrant code, correct the instruction. Otherwise, for a control section that has not been defined by an RSECT instruction, you can suppress reentrancy checking by specifying 'NORENT' as an assembler option. You cannot suppress reentrancy for a control section defined by an RSECT instruction.

Severity: 4

ASMA037 Illegal self-defining value - *xxxxxxxx*

Explanation: A decimal, binary (B), hexadecimal (X), or character (C) self-defining term contains characters that are not permitted or is in illegal format.

System Action: In the source program, the operand in error and the following operands are ignored. In a macro definition, the whole statement is ignored.

Programmer Response: Supply a valid self-defining term.

Severity: 8

ASMA038 Operand value falls outside of current section/LOCTR

Explanation: An ORG statement specifies a location outside the control section or the LOCTR in which the ORG is used. ORG cannot force a change to another section or LOCTR.

System Action: The statement is ignored.

Programmer Response: Change the ORG statement if it is wrong. Otherwise, insert a CSECT, DSECT, COM, or LOCTR statement to set the location counter to the correct section before the ORG statement is processed.

Severity: 12

ASMA039 Location counter error

Explanation: The maximum location counter value has been exceeded. When the OBJECT or DECK assembler option is specified the maximum location counter value is X'FFFFFF'. When the XOBJECT assembler option is specified the maximum location counter value is X'FFFFFFFF'.

System Action: The assembly continues, however, the resulting code will probably not run correctly.

Programmer Response: The probable cause is a high ORG statement value or a high START statement value. Correct the value or split up the control section.

Severity: 12

ASMA040 Missing operand

Explanation: The statement requires an operand, and none is present.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored.

Programmer Response: Supply the missing operand.

Severity: 12

ASMA041 Term expected; text is unclassifiable

Explanation: One of these errors has occurred:

- A term was expected, but the character encountered is not one that starts a term (letter, number, =, +, −, *).
- A letter and a quotation mark did not introduce a valid term; the letter is not L, C, G (DBCS option only), X, or B.

System Action: Another message accompanies an assembler statement. A machine instruction assembles as zero.

Programmer Response: Check for missing punctuation, a wrong letter on a self-defining term, a bad attribute request, a leading comma, or a dangling comma. Note that the length attribute is the only one accepted here. If a defined, scale, type, or integer attribute is needed, use a SETA statement and substitute the variable symbol where the attribute is needed.

Severity: 8

ASMA042 Length attribute of symbol is unavailable; default=1

Explanation: This statement has a length attribute reference to a symbol, and the length attribute of the symbol is unavailable for one of the following reasons:

- The symbols has not been previously defined.
- The type attribute of a symbol is U.

  A symbol defined by an EQU instruction has a type attribute of U, however, a reference to its length does not produce this message.

- The length cannot be determined due to lookahead processing. If a statement that defines a symbol, and references a length attribute, causes lookahead processing, the symbol might not be assigned a length attribute until after lookahead processing is complete. References to the same length attribute in subsequent conditional assembly statements, before lookahead processing completes, might cause this message to be produced.

System Action: The L' attribute defaults to 1.

Programmer Response: Ensure the symbol is defined. If you suspect the error might be caused because of lookahead processing, restructure your code so that the symbol is defined before it is referenced.

Severity: 8

ASMA043 Previously defined symbol - *xxxxxxxx*

Explanation: The symbol in a name field or in the operand field of an EXTRN or WXTRN statement was defined (used as a name or an EXTRN/WXTRN operand) in a previous statement.

System Action: The name or EXTRN/WXTRN operand of this statement is ignored. The following operands of an EXTRN or WXTRN are processed. The first occurrence of the symbol defines it.

Programmer Response: Correct a possible spelling error, or change the symbol.

Severity: 8

ASMA044 Undefined symbol - *xxxxxxxx*

Explanation: A symbol in the operand field has not been defined, that is, used in the name field of another statement, the operand field of an EXTRN or WXTRN, or, in the case of a literal, the operand of a previously processed machine instruction statement.

System Action: A machine instruction or an address constant assembles as zero. In a DC, DS, or DXD statement or in a duplication-factor or length-modifier expression, the operand in error and the following operands are ignored. In an EQU statement, zero is assigned as the value of the undefined symbol. Any other instruction is not processed.

Programmer Response: Define the symbol, or remove the references to it.

Severity: 8

ASMA045 Register or label not previously used - *xxxxxxxx*

Explanation: A register or label specified in a DROP statement has not been previously specified in a USING statement.

System Action: Registers or labels not active at the time are ignored.

Programmer Response: Remove the unreferenced registers or label from the DROP statement. You can drop all active base registers and labels at once by specifying DROP with a blank operand.

Severity: 4

ASMA046 Bit 7 of CCW flag byte must be zero

Explanation: Bit 7 of the flag byte of a channel command word specified by a CCW, CCW0, or CCW1 statement is not zero.

System Action: The CCW, CCW0, or CCW1 assembles as zero.

Programmer Response: Set bit 7 of the flag byte to zero to suppress this message during the next assembly.

Severity: 8

ASMA047 Severity code too large

Explanation: The severity code (first operand) of an MNOTE statement is not * or an unsigned decimal number from 0 to 255.

System Action: The statement is printed in standard format instead of MNOTE format. The MNOTE is given the severity code of this message.

Programmer Response: Choose a severity code of * or a number less than 255, or check for a generated severity code.

Severity: 8

ASMA048 ENTRY error

Explanation: One of the following errors was detected in the operand of an ENTRY statement:

- Duplicate symbol (previous ENTRY)
- Symbol defined in a DSECT or COM section
- Symbol defined by a DXD statement
- Undefined symbol
- Symbol defined by an absolute or complex relocatable EQU statement

System Action: The external symbol dictionary output is suppressed for the symbol.

Programmer Response: Define the ENTRY operand correctly.

Severity: 8

ASMA049 Illegal range on ISEQ

Explanation: If this message is accompanied by another, this one is advisory. If it appears by itself, it indicates one of the following errors:

- An operand value is less than 1 or greater than 80, or the second operand (rightmost column to be checked) is less than the first operand (extreme left column to be checked).
- More or fewer than two operands are present, or an operand is null (empty).
- An operand expression contains an undefined symbol.
- An operand expression is not absolute.

- The statement is too complex. For example, it might have forward references or cause an arithmetic overflow during evaluation.
- The statement is circularly defined.

System Action: Sequence checking stops.

Programmer Response: Supply valid ISEQ operands. Also, be sure that the records following this statement are in order; they have not been sequence checked.

Severity: 4

ASMA050 Illegal name field; name discarded - *xxxxxxxx*

Explanation: One of these errors has occurred:

- The name field of a macro prototype statement contains an incorrect symbolic parameter (variable symbol).
- The name field of a COPY statement in a macro definition contains an entry other than blank or a valid sequence symbol.

System Action: The incorrect name field is ignored.

Programmer Response: Correct the incorrect name field.

Severity: 8

ASMA051 Illegal statement outside a macro definition

Explanation: A MEND, MEXIT, ASPACE, or AREAD statement appears outside a macro definition.

System Action: The statement is ignored.

Programmer Response: Remove the statement or, if a macro definition is intended, insert a MACRO statement.

Severity: 8

ASMA052 Record out of sequence

Explanation: Input sequence checking, under control of the ISEQ assembler instruction, has determined that this statement is out of sequence. The sequence number of the statement is appended to the message.

System Action: The statement assembles normally. However, the sequence number of the next statement is checked relative to this statement.

Programmer Response: Put the statements in correct sequence. If you want a break in sequence, put in a new ISEQ statement and sequence number. ISEQ always resets the sequence number; the record following the ISEQ is not sequence checked.

Severity: 12

ASMA053 Blank sequence field

Explanation: Input sequence checking, controlled by the ISEQ assembler statement, has detected a statement with a blank sequence field. The sequence number of the last numbered statement is appended to the message.

System Action: The statement assembles normally. The sequence number of the next statement is checked relative to the last statement having a nonblank sequence field.

Programmer Response: Put the correct sequence number in the statement or discontinue sequence checking over the blank statements by means of an ISEQ statement with a blank operand.

Severity: 4

ASMA054 Illegal continuation record

Explanation: A statement has more than 10 records or end-of-input has been encountered when a continuation record was expected.

System Action: The records already read are processed as is. If the statement had more than 10 records, the next record is treated as the beginning of a new statement.

Programmer Response: In the first case, break the statement into two or more statements. In the second case, ensure that a continued statement does not span the end of a library member. Check for lost records or an extraneous continuation character.

Severity: 8

ASMA055 Recursive COPY

Explanation: A nested COPY statement (COPY within another COPY) attempted to copy a library member already being copied by a higher level COPY within the same nest.

System Action: This COPY statement is ignored.

Programmer Response: Correct the operand of this COPY if it is wrong, or rearrange the nest so that the same library member is not copied by COPY statements at two different levels.

Severity: 12

ASMA057 Undefined operation code - *xxxxxxxx*

Explanation: One of the following errors has occurred:

- The operation code of this statement is not a valid machine or assembler instruction or macro name.
- In an OPSYN statement, this operand symbol is undefined or illegal or, if no operand is present, the name field symbol is undefined.

System Action: The statement is ignored. Note that OPSYN does not search the macro library for an undefined operand.

Programmer Response: Correct the statement. In the case of an undefined macro instruction, the wrong data set might have been specified for the macro library. In the case of OPSYN, a previous OPSYN or macro definition might have failed to define the operation code.

Severity: 8

ASMA059 Illegal ICTL

Explanation: An ICTL statement has one of the following errors:

- The operation code was created by variable symbol substitution.
- It is not the first statement in the assembly.
- The value of one or more operands is incorrect.
- An operand is missing.
- A character is detected in the operand field that is not permitted.

System Action: The ICTL statement is ignored. Assembly continues with standard ICTL values.

Programmer Response: Correct or remove the ICTL. The begin column must be 1-40; the end column must be 41-80 and at least five greater than the begin column; and the continue column must be 2-40.

Severity: 16

ASMA060 COPY code not found

Explanation: (1) If this message is on a COPY statement and no text is printed with it, one of the following occurred:

- The library member was not found.
- The lookahead phase previously processed the COPY statement and did not find the library member, the copy was recursive, or the operand contains a variable symbol. Variable symbols can be used if the COPY statement is in open code.

(2) If this message is not on a COPY statement, but has a library member name printed with it, the lookahead phase of the assembler could not find the library member because the name is undefined or contains a variable symbol.

System Action: The COPY statement is ignored; the library member is not copied.

Programmer Response: Check that the correct macro library was assigned, or check for a possible misspelled library member name.

If COPY member is not defined in any macro library, and is not processed because of an AGO or AIF assembler instruction, add a dummy COPY member with the name to the macro library.

Severity: 12

ASMA061 Symbol not name of DSECT or DXD

Explanation: The operand of a Q-type address constant is not a symbol or the name of a DSECT or DXD statement.

System Action: The constant assembles as zero.

Programmer Response: Supply a valid operand.

Severity: 8

ASMA062 Illegal operand format

Explanation: One of the following errors has occurred:

- AMODE—the operand does not specify 24, 31, or ANY.
- DROP or USING—more than 16 registers are specified in the operand field.
- EXITCTL—more than five operands are specified, or the first operand is not a valid exit type, or the value of one of the expressions specified in the second and subsequent operands is outside the range -2^{31} to $+2^{31}-1$.
- MNOTE—the syntax of the severity code (first operand) is not correct, or the sum of the length of the operands including quotes and commas exceeds 1024 bytes.
- PRINT—an operand specifies an incorrect print option.
- PUSH or POP—an operand does not specify a PRINT or USING statement.
- RMODE—the operand does not specify 24 or ANY.
- TITLE—more than 100 bytes were specified.
- ADATA—more than five operands are specified, or the value of one of the expressions specified in one of the first four operands is outside the range -2^{31} to $+2^{31}-1$, or the fifth operand is not a valid character expression.

System Action: The first 16 registers in a DROP or USING statement are processed. The operand in error and the following operands of a PUSH, POP, or PRINT statement are ignored. The AMODE or RMODE instruction is ignored, and the name field (if any) does not appear in the cross-reference listing. The first 100 bytes of the operand of the TITLE instruction are used as the title.

Programmer Response: Supply a valid operand field.

Severity: 8

ASMA063 No ending apostrophe

Explanation: The quotation mark terminating an operand is missing, or the standard value of a keyword parameter of a macro prototype statement is missing.

System Action: The operand or standard value in error is ignored. If the error is in a macro definition model statement, the whole statement is ignored.

Programmer Response: Supply the missing quotation mark.

Severity: 8

ASMA064 Floating point characteristic out of range

Explanation: A converted floating-point constant is too large or too small for the processor. The allowable range is $7.2x10^{75.}$ to $5.3x10^{-77}$.

System Action: The constant assembles as zero.

Programmer Response: Check the characteristic (exponent), exponent modifier, scale modifier, and mantissa (fraction) for validity. Remember that a floating-point constant is rounded, not truncated, after conversion.

Severity: 12

ASMA065 Unknown type

Explanation: An unknown constant type has been used in a DC or DS statement or in a literal.

System Action: The operand in error and the following operands are ignored.

Programmer Response: Supply a valid constant. Look for an incorrect type code or incorrect syntax in the duplication factor.

Severity: 8

ASMA066 2-byte relocatable address constant

Explanation: This statement contains a relocatable Y-type address constant or a 2-byte relocatable A-type address constant. Addressing errors occur if the address constant is used to refer to a storage address equal to or greater than 64K (65,536).

System Action: The statement assembles as written.

Programmer Response:

Programmer Response: If the address constant is used to refer to a storage address less than 64K (65,536), the 2-byte relocatable address constant is valid. You can use the assembler option RA2 to suppress this message.

Severity: 4

ASMA067 Illegal duplication factor

Explanation: One of the following errors has occurred:

- A literal has a zero duplication factor.
- The duplication factor of a constant is greater than the maximum of $2^{24}-1$ bytes.
- A duplication factor expression of a constant is not correct.

System Action: The operand in error and the following operands of a DC, DS, or DXD statement are ignored. The statement containing the literal assembles as zero.

Programmer Response: Supply a valid duplication factor. If you want a zero duplication factor, write the literal as a DC statement.

Severity: 12

ASMA068 Length error

Explanation: One of the following errors has occurred:

- The length modifier of a constant is wrong.
- The C, X, B, Z, or P-type constant is too long.
- An operand is longer than $2^{24}-1$ bytes.
- A relocatable address constant has an illegal length.
- The length field in a machine instruction is not correct or out of the permissible range.

System Action: The operand in error and the following operands of the DC, DS, or DXD statement are ignored, except that an address constant with an illegal length is truncated. A machine instruction assembles as zero.

Programmer Response: Supply a valid length.

Severity: 12

ASMA070 Scale modifier error

Explanation: A scale modifier in a constant is used illegally, is out of range, or is relocatable, or there is an error in a scale modifier expression.

System Action: If the scale modifier is out of range, it defaults to zero. Otherwise, the operand in error and the following operands are ignored.

Programmer Response: Supply a valid scale modifier.

Severity: 8

ASMA071 Exponent modifier error

Explanation: The constant contains multiple internal exponents, the exponent modifier is out of range or relocatable, or the sum of the exponent modifier and the internal exponent is out of range.

System Action: If the constant contains multiple internal exponents, the operand in error and the following operands are ignored. Otherwise, the exponent modifier defaults to zero.

Programmer Response: Change the exponent modifier or the internal exponent.

Severity: 8

ASMA072 Data item too large

Explanation: The value of a Y-type address constant or H-type constant is larger than $2^{15}-1$ or smaller than -2^{15}, or the value of a F-type constant is larger than $2^{31}-1$ or smaller than -2^{31}.

System Action: The constant is truncated. The high-order bits are lost.

Programmer Response: Supply a smaller scale modifier, a longer constant, or a smaller value.

Severity: 8

ASMA073 Precision lost

Explanation: The scale modifier of a floating-point number was large enough to shift the whole fraction out of the converted constant.

System Action: The constant assembles with an exponent but with a zero mantissa (fraction).

Programmer Response: Change the scale modifier or use a longer constant. For example, use a D-type constant instead of an E-type constant.

Severity: 8

ASMA074 Illegal syntax in expression

Explanation: An expression has two terms or two operations in succession, or incorrect or missing characters or delimiters.

System Action: In a DC, DS, or DXD statement, the operand in error and the following operands are ignored. In a macro definition, the whole statement is ignored. A machine instruction assembles as zero.

Programmer Response: Check the expression for typing errors, or for missing or incorrect terms or characters.

Severity: 8

ASMA075 Arithmetic overflow

Explanation: The intermediate or final value of an expression is not within the range -2^{31} through $2^{31}-1$.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored.

Programmer Response: Change the expression.

Severity: 8

ASMA076 Statement complexity exceeded

Explanation: The complexity of this statement caused the assembler's expression evaluation work area to overflow.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored.

Programmer Response: Reduce the number of terms, levels of expressions, or references to complex relocatable EQU names.

Severity: 8

ASMA077 Circular definition

Explanation: The value of a symbol in an expression is dependent on itself, either directly or indirectly, via one or more EQU statements. In the following example:

```
A  EQU  B
B  EQU  C
C  EQU  A
```

A is circularly defined.

System Action: The value of the EQU statement defaults to the current value of the location counter. All other EQU statements involved in the circularity are defaulted in terms of this one.

Programmer Response: Supply a correct definition.

Severity: 8

ASMA079 Illegal PUSH-POP

Explanation: More POP assembler instructions than PUSH instructions have been encountered.

System Action: This POP instruction is ignored.

Programmer Response: Eliminate a POP statement, or add another PUSH statement.

Severity: 8

ASMA080 Statement is unresolvable

Explanation: A statement cannot be resolved, because it contains a complex relocatable expression or because the location counter has been circularly defined.

System Action: The statement is ignored.

Programmer Response: Untangle the forward references or check the complex relocatable EQU statements.

Severity: 8

ASMA081 Created SET symbol exceeds 63 characters

Explanation: A SET symbol created by variable symbol substitution is longer than 63 characters (including the ampersand as the first character).

System Action: If the symbol is in the operand field of a SET, AIF, or AGO statement, its value is set to zero or null, and the type attribute is set to undefined (U). If the symbol is in the operand field of a GBL, or LCL statement or the name field of a SET statement, processing of the macro stops.

Programmer Response: Shorten the symbol.

Severity: 8

ASMA082 Created SET symbol is null

Explanation: A SET symbol created by variable symbol substitution is null (empty string).

System Action: If the symbol is in the operand field of a SET, AIF, or AGO statement, its value is set to zero or null, and the type attribute is set to undefined (U). If the symbol is in the operand field of a GBL, or LCL statement or the name field of a SET statement, processing of the macro stops.

Programmer Response: Supply a valid symbol.

Severity: 8

ASMA083 Created SET symbol is not a valid symbol

Explanation: A SET symbol created by variable symbol substitution or concatenation does not consist of an ampersand followed by up to 62 alphanumeric characters, the first of which is alphabetic.

System Action: If the symbol is in the operand field of a SET, AIF, or AGO statement, its value is set to zero or null, and the type attribute is set to undefined (U). If the symbol is in the operand field of a GBL or LCL statement or the name field of a SET statement, processing of the macro stops.

Programmer Response: Supply a valid symbol.

Severity: 8

ASMA084 Generated name field exceeds 63 characters; discarded

Explanation: The name field on a generated statement is longer than 63 characters.

System Action: The name field is not generated. The rest of the statement assembles normally.

Programmer Response: Shorten the generated name to 63 characters or fewer.

Severity: 12

ASMA085 Generated operand field is null

Explanation: The operand field of a generated statement is null (empty).

System Action: The statement assembles as though no operand were specified.

Programmer Response: Provide a nonempty operand field. If you want the statement assembled with no operand, substitute a comma rather than leave the operand blank.

Severity: 0

ASMA086 Missing MEND generated

Explanation: A macro definition, appearing in the source program or being read from a library by a macro call or a COPY statement, ends before a MEND statement is encountered to end it.

System Action: A MEND statement is generated. The portion of the macro definition read in is processed.

Programmer Response: Insert the MEND statement if it was left out. Otherwise, check if all the macro definition is on the library.

Severity: 12

ASMA087 Generated operation code is null

Explanation: The operation code of a generated statement is null (blank).

System Action: The generated statement is printed but not assembled.

Programmer Response: Provide a valid operation code.

Severity: 12

ASMA088 Unbalanced parentheses in macro call operand

Explanation: Excess left or too few right parentheses occur in an operand (parameter) of a macro call statement.

System Action: The parameter corresponding to the operand in error is given a null (empty) value.

Programmer Response: Balance the parentheses.

Severity: 8

ASMA089 Arithmetic expression contains illegal delimiter or ends prematurely

Explanation: An arithmetic expression contains an incorrect character or an arithmetic subscript ends without enough right parentheses.

System Action: The statement is ignored.

Programmer Response: Supply a valid expression.

Severity: 8

ASMA090 Excess right parenthesis in macro call operand

Explanation: A right parenthesis without a corresponding left parenthesis was detected in an operand of a macro instruction.

System Action: The excess right parenthesis is ignored. The macro expansion might be incorrect.

Programmer Response: Insert the correct parenthesis.

Severity: 8

ASMA091 SETC or character relocatable operand over 255 characters; truncated to 255 characters

Explanation: The value of the operand of a SETC statement or the character relational operand of an AIF statement is longer than 255 characters. This might occur before substrings are evaluated.

System Action: The first 255 characters are used.

Programmer Response: Shorten the SETC expression value or the operand value.

Severity: 8

ASMA092 Substring expression 1 points past string end; default=null

Explanation: The first arithmetic expression of a SETC substring points beyond the end of the expression character string.

System Action: The substring is given a null value.

Programmer Response: Supply a valid expression.

Severity: 8

ASMA093 Substring expression 1 less than 1; default=null

Explanation: The first arithmetic expression of a SETC substring is less than one; that is, it points before the expression character string.

System Action: The substring expression defaults to null.

Programmer Response: Supply a valid expression.

Severity: 8

ASMA094 Substring goes past string end; default=remainder

Explanation: The second expression of a substring notation specifies a length that extends beyond the end of the string.

System Action: The result of the substring operation is a string that ends with the last character in the character string.

Programmer Response: Make sure the arithmetic expression used to specify the length does not specify characters beyond the end of the string. Either change the first or the second expression in the substring notation. You can use the assembler option FLAG(NOSUBSTR) to suppress this message.

Severity: 0

ASMA095 Substring expression 2 less than 0; default=null

Explanation: The second arithmetic expression of a SETC substring is less than or equal to zero.

System Action: No characters (a null string) from the substring character expression are used.

Programmer Response: Supply a valid expression.

Severity: 4

ASMA096 Unsubscripted SYSLIST; default=SYSLIST(1)

Explanation: The system variable symbol, &SYSLIST, is not subscripted. &SYSLIST(n) refers to the *n*th positional parameter in a macro instruction. N'&SYSLIST does not have to be subscripted.

System Action: The subscript defaults to one so that it refers to the first positional parameter.

Programmer Response: Supply the correct subscript.

Severity: 8

ASMA097 Invalid attribute reference to SETA or SETB symbol; default=U or 0

Explanation: A length (L'), scaling (S'), integer (I'), or defined (D') attribute refers to a SETA or SETB symbol.

System Action: The attributes are set to default values: L'=0, S'=0, I'=0 ,and D'=0.

Programmer Response: Change or remove the attribute reference.

Severity: 8

ASMA098 Attribute reference to invalid symbol; default=U or 0

Explanation: An attribute attempted to reference a symbol that is not correct or has a null value. (A valid symbol is 1 to 63 alphanumeric characters, the first of which is alphabetic.)

System Action: For a type (T') attribute, defaults to U. For all other attributes, defaults to 0.

Programmer Response: Supply a valid symbol.

Severity: 8

ASMA099 Wrong type of constant for S' or I' attribute reference; default=0

Explanation: An integer (I') or scaling (S') attribute references a symbol whose type is other than floating-point (E,D,L), decimal (P,Z), or fixed-point (H,F).

System Action: The integer or scaling attribute defaults to zero.

Programmer Response: Remove the integer or scaling attribute reference or change the constant type.

Severity: 4

ASMA100 Subscript less than 1; default to subscript = 1.

Explanation: The subscript of a subscripted SET symbol in the name field of a SET statement, the operand field of a GBL or LCL statement, or an &SYSLIST statement is less than 1.

System Action: The subscript defaults to 1.

Programmer Response: Supply the correct subscript.

Severity: 8

ASMA101 Subscript less than 1; default to value=0 or null

Explanation: The subscript of a SET symbol in the operand field is less than 1.

System Action: The value is set to zero or null.

Programmer Response: Supply a valid subscript.

Severity: 8

ASMA102 Arithmetic term is not self-defining term; default=0

Explanation: A SETC term or expression used as an arithmetic term is not a valid self-defining term.

System Action: The value of the SETC term or expression is set to zero.

Programmer Response: Make the SETC a self-defining term, such as C'A', X'1EC', B'1101', or 27. The C, X, or B and the quotation marks must be part of the SETC value.

Severity: 8

ASMA103 Multiplication overflow; default product=1

Explanation: A multiplication overflow occurred in a macro definition statement.

System Action: The value of the expression up to the point of overflow is set to one; evaluation continues.

Programmer Response: Change the expression so that overflow does not occur; break it into two or more operations, or regroup the terms by parentheses.

Severity: 8

ASMA105 Arithmetic expression too complex

Explanation: An arithmetic expression in a macro definition statement caused an overflow because it is too complex; that is, it has too many terms, levels, or both.

System Action: The assembly stops.

Programmer Response: Simplify the expression or break it into two or more expressions.

Severity: 20

ASMA106 Wrong target symbol type; value left unchanged

Explanation: The SET symbol in the name field has already been declared, and is a different type to the type of SETx instruction. For example, you might have previously declared a SET symbol as arithmetic (SETA), and you are attempting to use the SET symbol as the target of a SETC instruction.

System Action: The statement is ignored.

Programmer Response: Make the declaration agree with the SET statement type. If you want to store across SET symbol types, first store into a SET symbol of matching type, and then use another SETx instruction to store the value, represented by the matching SET symbol, into the non-matching SET symbol.

Severity: 8

ASMA107 Inconsistent dimension on target symbol; subscript ignored, or 1 used

Explanation: The SET symbol in the name field is dimensioned (subscripted), but was not declared in a GBL or LCL statement as dimensioned, or vice versa.

System Action: The subscript is ignored or a subscript of 1 is used, in accordance with the declaration.

Programmer Response: Make the declaration and the usage compatible. Note that you can declare a local SET symbol as dimensioned by using it, subscripted, in the name field of a SET statement.

Severity: 8

ASMA108 Inconsistent dimension on SET symbol reference; default = 0, null, or type = U

Explanation: A SET symbol in the operand field is dimensioned (subscripted), but was not declared in a GBL or LCL statement as dimensioned, or vice versa.

System Action: A value of zero or null is used for the subscript. If the type attribute of the SET symbol is requested, it is set to U.

Programmer Response: Make the declaration and the usage compatible. You can declare a SET symbol as dimensioned by using it, subscripted, in the name field of a SET statement.

Severity: 8

ASMA109 Multiple SET operands for undimensioned SET symbol; gets last operand

Explanation: Multiple operands were assigned to an undimensioned (unsubscripted) SET symbol.

System Action: The SET symbol is given the value of the last operand.

Programmer Response: Declare the SET symbol as dimensioned, or assign only one operand to it.

Severity: 8

ASMA110 Library macro first statement not 'MACRO' or comment

Explanation: A statement other than a comment statement preceded a MACRO statement in a macro definition read from a library.

System Action: The macro definition is not read from the library. A corresponding macro call cannot be processed.

Programmer Response: Ensure that the library macro definition begins with a MACRO statement preceded (optionally) by comment statements only.

Severity: 12

ASMA111 Invalid AIF or SETB operand field

Explanation: The operand of an AIF or SETB statement either does not begin with a left parenthesis or is missing altogether.

System Action: The statement is ignored.

Programmer Response: Supply a valid operand.

Severity: 12

ASMA112 Invalid sequence symbol - *xxxxxxxx*

Explanation: One of the following errors has occurred:

- A sequence symbol doesn't begin with a period followed by one to 62 alphanumeric characters, the first being alphabetic.
- A sequence symbol in the name field was created by substitution.
- Operand of AGO is blank or sequence symbol in AIF is blank.

System Action: The sequence symbol in the name field is ignored. A sequence symbol in the operand field of an AIF or AGO statement causes the whole statement to be ignored.

Programmer Response: Supply a valid sequence symbol.

Severity: 12

ASMA113 Continue column blank

Explanation: A SET symbol declaration in a GBL or LCL statement began with an ampersand in the end column (normally column 71) of the previous record, but the continue column (normally column 16) of this record is blank.

System Action: This record and any following records of the statement are ignored. Any SET symbols that completely appear on the previous record(s), are processed normally.

Programmer Response: Begin this record in the continuation column.

Severity: 12

ASMA114 Invalid COPY operand

Explanation: The operand of a COPY statement is not a symbol of 1 to 8 alphanumeric characters, the first being alphabetic.

System Action: The COPY statement is ignored.

Programmer Response: Supply a valid operand. In open code the operand can be specified as a previously defined SET symbol.

Severity: 12

ASMA115 COPY operand too long

Explanation: The symbol in the operand field of a COPY statement is more than 8 characters long.

System Action: The COPY statement is ignored.

Programmer Response: Supply a valid operand.

Severity: 12

ASMA116 Illegal SET symbol - *xxxxxxxx*

Explanation: A SET symbol in the operand field of a GBL or LCL statement or in the name field of a SET statement does not consist of an ampersand followed by one to 62 alphanumeric characters, the first being alphabetic.

System Action: For a GBL or LCL statement, the incorrect SET symbol and all following SET symbols in a GBL or LCL statement are ignored. For a SET statement, the whole SET statement is ignored.

Programmer Response: Supply a SET symbol.

Severity: 8

ASMA117 Illegal subscript

Explanation: The subscript following a SET symbol contained unbalanced parentheses or an incorrect arithmetic expression.

System Action: This statement is ignored.

Programmer Response: Supply an equal number of left and right parentheses or a valid arithmetic expression.

Severity: 8

ASMA118 Source macro ended by 'MEND' in COPY code

Explanation: A library member, being copied by a COPY statement within a macro definition, contained a MEND statement.

System Action: The MEND statement is honoured and the macro definition stops. No more COPY code is read. The statements brought in before the end of the COPY code are processed. The macro definition is continued with the statement following the COPY statement.

Programmer Response: Make sure that each library member to be used as COPY code contains balanced MACRO and MEND statements.

Severity: 12

ASMA119 Too few MEND statements in COPY code

Explanation: A macro definition is started in a library member brought in by a COPY statement and the COPY code ends before a MEND statement is encountered.

System Action: A MEND statement is generated to end the macro definition. The statements brought in before the end of the COPY code are processed.

Programmer Response: Check to see if part of the macro definition was lost. Also, ensure that each macro definition to be used as COPY code contains balanced MACRO and MEND statements.

Severity: 12

ASMA120 EOD where continuation record expected

Explanation: An end-of-data occurred when a continuation record was expected.

System Action: The portion of the statement read in is assembled. The assembly stops if the end-of-data is on the PRIMARY INPUT. If a library member is being copied, the assembly continues with the statement after the COPY statement.

Programmer Response: Check to determine whether any statements were omitted from the source program or from the COPY code.

Severity: 12

ASMA121 Insufficient storage for editor work area

Explanation: The macro editor module of the assembler cannot get enough main storage for its work areas.

System Action: The assembly stops.

Programmer Response: Split the assembly into two or more parts or give the macro editor more working storage. This can be done by increasing the region size for the assembler, decreasing blocking factor or block size on the assembler data sets, or a combination of both.

Severity: 12

ASMA122 Illegal operation code format

Explanation: The operation code is not followed by a blank or is missing altogether, or the first record of a continued source statement is missing.

System Action: The statement is ignored.

Programmer Response: Ensure that the statement has a valid operation code and that all records of the statement are present.

Severity: 12

ASMA123 Variable symbol too long

Explanation: A SET symbol, symbolic parameter, or sequence symbol contains more than 62 characters following the ampersand or period.

System Action: This statement is ignored.

Programmer Response: Shorten the SET symbol or sequence symbol.

Severity: 12

ASMA124 Illegal use of parameter

Explanation: A symbolic parameter was used in the operand field of a GBL or LCL statement or in the name field of a SET statement. In other words, a variable symbol has been used both as a symbolic parameter and as a SET symbol.

System Action: The statement is ignored.

Programmer Response: Change the variable symbol to one that is not a symbolic parameter.

Severity: 12

ASMA125 Illegal macro name - macro uncallable

Explanation: The operation code of a macro prototype statement is not a valid symbol; that is, one to 63 alphanumeric characters, the first alphabetic.

System Action: The macro definition is edited. However, since the macro name is not correct, the macro cannot be called.

Programmer Response: Supply a valid macro name.

Severity: 12

ASMA126 Library macro name incorrect - *xxxxxxxx*

Explanation: The operation code of the prototype statement of a library macro definition is not the same as the operation code of the macro instruction (call). Library macro definitions are located by their member names. However, the assembler compares the macro instruction with the macro prototype.

System Action: The macro definition is edited using the operation code of the prototype statement as the macro name. Thus, the definition cannot be called by this macro instruction.

Programmer Response: Ensure that the member name of the macro definition is the same as the operation code of the prototype statement. This usually requires listing the macro definition from the library, use of the LIBMAC option to cause the macro definition to be listed, or a COPY of the member name.

Severity: 12

ASMA127 Illegal use of ampersand

Explanation: One of the following errors has occurred:

- An ampersand was found where all substitution should have already been done.
- The standard value of a keyword parameter in a macro prototype statement contained a single ampersand or a string with an odd number of ampersands.
- An unpaired ampersand occurred in a character (C) constant.

System Action: In a macro prototype statement, all information following the error is ignored. In other statements, the action depends on which field the error occurred in. If the error occurred in the name field, the statement is processed without a name. If the error occurred in the operation code field, the statement is ignored. If the error occurred in the operand field, another message is issued to specify the default. However, if the error occurred in a C-type constant, the operand in error and the following operands are ignored.

Programmer Response: Ensure that ampersands used in keyword standard values or in C-type constant values occur in pairs. Also, avoid substituting an ampersand into a statement unless there is a double ampersand.

Severity: 12

ASMA128 Excess right parenthesis

Explanation: An unpaired right parenthesis has been found.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored and an additional message relative to the statement type appears. However, if the error is in the standard value of a keyword on a macro prototype statement, only the operands in error and the following operands are ignored.

Programmer Response: Make sure that all parentheses are paired.

Severity: 12

ASMA129 Insufficient right parentheses

Explanation: An unpaired left parenthesis has been found. Parentheses must balance at each comma in a multiple operand statement.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored and an additional message relative to the statement type appears. However, if the error is in the standard value of a keyword on a macro prototype statement, only the operands in error and the following operands are ignored.

Programmer Response: Make sure that all parentheses are paired.

Severity: 12

ASMA130 Illegal attribute reference

Explanation: One of the following errors has occurred:

- The symbol following a I, L, S, or T attribute reference is not a valid variable symbol or ordinary symbol or literal that has been previously used in a machine instruction.
- The symbol following a K or N attribute reference is not a valid variable symbol.

- The symbol following a D attribute reference is not a valid variable symbol or ordinary symbol.
- The quotation mark is missing from a T attribute reference.

System Action: The statement is ignored.

Programmer Response: Supply a valid attribute reference.

Severity: 12

ASMA131 Parenthesis nesting depth exceeds 255

Explanation: There are more than 255 levels of parentheses in a SETA expression.

System Action: The statement is ignored.

Programmer Response: Rewrite the SETA statement using several statements to regroup the subexpressions in the expression.

Severity: 12

ASMA132 Invalid SETB expression

Explanation: A SETB expression in the operand field of a SETB statement or an AIF statement does not consist of valid character relational expressions, arithmetic relational expressions, and single SETB symbols, connected by logical operators.

System Action: The statement is ignored.

Programmer Response: Supply a valid SETB expression.

Severity: 12

ASMA133 Illegal substring reference

Explanation: A substring expression following a SETC expression does not consist of two valid SETA expressions separated by a comma and enclosed in parentheses.

System Action: The statement is ignored.

Programmer Response: Supply a valid substring expression. The second value in the substring expression can be *.

Severity: 12

ASMA134 Invalid relational operator

Explanation: Characters other than EQ, NE, LT, GT, LE, or GE are used in a SETB expression where a relational operator is expected.

System Action: The statement is ignored.

Programmer Response: Supply a valid relational operator.

Severity: 12

ASMA135 Invalid logical operator

Explanation: Characters other than AND, OR, NOT, or XOR are used in a SETB expression where a logical operator is expected.

System Action: The statement is ignored.

Programmer Response: Supply a valid logical operator.

Severity: 12

ASMA136 Illegal logical/relational operator

Explanation: Characters other than a valid logical or relational operator were found where a logical or relational operator was expected.

System Action: The statement is ignored.

Programmer Response: Supply a valid logical or relational operator.

Severity: 12

ASMA137 Illegal SETC expression

Explanation: The operand of a SETC statement or the character value used in a character relation is erroneous. It must be a valid type attribute (T') reference or a valid character expression enclosed in quotation marks.

System Action: The statement is ignored.

Programmer Response: Supply a valid expression.

Severity: 12

ASMA139 EOD during REPRO processing

Explanation: A REPRO statement was immediately followed by an end-of-data so that no valid record could be punched. The REPRO is either the last record of source input or the last record of a COPY member.

System Action: The REPRO statement is ignored.

Programmer Response: Remove the REPRO or ensure that it is followed by a record to be punched.

Severity: 12

ASMA140 END record missing

Explanation: End-of-file on the source input data set occurred before an END statement was read. One of the following has occurred:

- The END statement was omitted or misspelled.
- The END operation code was changed or deleted by OPSYN or by definition of a macro named END. The lookahead phase of the assembler marks what it thinks is the END statement. If an OPSYN statement or a macro definition redefines the END statement, premature end-of-input might occur because

the assembler does not pass the original END statement.

System Action: An END statement is generated. It is assigned a statement number but not printed. If any literals are waiting, they are processed as usual following the END statement.

Programmer Response: Check for lost records. Supply a valid END statement; or, if you use OPSYN to define another symbol as END, place it *before* the possible entry into the lookahead phase.

Severity: 4

ASMA141 Bad character in operation code

Explanation: The operation code contains a non-alphanumeric character, that is, a character other than A to Z, 0 to 9, $, #, _, or @. Embedded blanks are not allowed.

System Action: The statement is ignored.

Programmer Response: Supply a valid operation code. If the operation code is formed by variable symbol substitution, check the statements leading to substitution.

Severity: 8

ASMA142 Operation code not complete on first record

Explanation: The whole name and operation code, including a trailing blank, is not contained on the first record (before the continue column—usually column 72) of a continued statement.

System Action: The statement is ignored.

Programmer Response: Shorten the name, operation code, or both, or simplify the statement by using a separate SETC statement to create the name or operation code by substitution.

Severity: 8

ASMA143 Bad character in name field

Explanation: The name field contains a non-alphanumeric character, that is, a character other than A to Z, 0 to 9, $, #, @, or _.

System Action: If possible, the statement is processed without a name. Otherwise, it is ignored.

Programmer Response: Put a valid symbol in the name field.

Severity: 8

ASMA144 Begin-to-continue columns not blank

Explanation: On a continuation record, one or more columns between the begin column (usually column 1) and the continue column (usually column 16) are not blank.

System Action: The extraneous characters are ignored.

Programmer Response: Check whether the operand started in the wrong column or whether the preceding record contained an erroneous continuation character.

Severity: 8

ASMA145 Operator, right parenthesis, or end-of-expression expected

Explanation: One of the following has occurred:

- A letter, number, equal sign, quotation mark, or undefined character occurred following a term where a right parenthesis, an operator, a comma, or a blank ending the expression was expected.
- In an assembler instruction, a left parenthesis followed a term.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored and another message, relative to the operation code, is issued.

Programmer Response: Check for an omitted or misplaced operator. Subscripting is not allowed on this statement.

Severity: 8

ASMA146 Self-defining term too long or value too large

Explanation: A self-defining term is longer than 4 bytes, (8 hexadecimal digits, 32 bits, or 4 characters), or the value of a decimal self-defining term is greater than $2^{31}-1$.

System Action: A machine instruction assembles as zero. An assembler instruction is ignored. However, another message, relative to the operation code, is issued.

Programmer Response: Reduce the size of the self-defining term, or specify it in a DC statement.

Severity: 8

ASMA147 Symbol too long, or first character not a letter

Explanation: A symbol does not begin with a letter or an underscore (_) or is longer than 63 characters.

System Action: If the symbol is in the name field, the statement is processed as unnamed. If the symbol is in the operand field, an assembler operation or a macro definition model statement is ignored and a machine operation assembles as zero.

Programmer Response: Supply a valid symbol.

Severity: 8

ASMA148 Self-defining term lacks ending quote or has bad character

Explanation: A hexadecimal or binary self-defining term contains a character that is not permitted or is missing the final quotation mark, or a pure DBCS self-defining term contains SO and SI with no double-byte data between them.

System Action: A machine operation assembles as zero. An assembler operation is ignored and another message, relative to the operation code, is issued.

Programmer Response: Correct the incorrect term.

Severity: 8

ASMA149 Literal length exceeds 256 characters, including = sign

Explanation: A literal is longer than 256 characters.

System Action: The instruction assembles as zero.

Programmer Response: Shorten the literal, or change it to a DC statement.

Severity: 8

ASMA150 Symbol has non-alphanumeric character or invalid delimiter

Explanation: The first character following a symbol is not a valid delimiter (plus sign, minus sign, asterisk, slash, left or right parenthesis, comma, or blank).

System Action: A machine operation assembles as zero. An assembler operation is ignored, and another message, relative to this operation code, is issued.

Programmer Response: Ensure that the symbol does not contain a non-alphanumeric character and that it is followed by a valid delimiter.

Severity: 8

ASMA151 Literal expression modifiers must be absolute and predefined

Explanation: The duplication factor or length modifier in a literal is not a self-defining term, or an expression using self-defining terms or previously defined symbols.

System Action: The statement assembles as zero.

Programmer Response: Supply a valid self-defining term or ensure that symbols appear in the name field of a *previous* statement.

Severity: 8

ASMA152 External symbol too long or unacceptable character

Explanation: One of the following errors has occurred:

- An external symbol is longer than 8 characters, or contains a bad character. An external symbol might be the name of a CSECT, START, DXD, AMODE, RMODE, or COM statement, or the operand of an ENTRY, EXTRN, or WXTRN statement or a Q-type or V-type address constant.
- The operand of an ENTRY, EXTRN, or WXTRN statement or a Q-type or V-type address constant is an expression instead of a single term, or contains a bad character.

System Action: The symbol does not appear in the external symbol dictionary. If the error is in the name field, an attempt is made to process the statement as unnamed. If the error is in the operand field, the bad operand is ignored and, if possible, the following operands are processed. A bad constant assembles as zero.

Programmer Response: Supply a shorter name or replace the expression with a symbol.

Severity: 12

ASMA153 START statement illegal - CSECT already begun

Explanation: A START statement occurred after the beginning of a control section.

System Action: The statement is processed as a CSECT statement; any operand is ignored.

Programmer Response: Ensure that the START precedes all machine instructions and any assembler instruction, such as EQU, that initiates a control section. If you want EQU statements before the START, place them in a dummy section (DSECT).

Severity: 12

ASMA154 Operand must be absolute, predefined symbols; set to 0

Explanation: The operand on a START or MHELP statement is not correct. If there is another message with this statement, this message is advisory. If this message appears alone, it indicates one of the following:

- There is a location counter reference (*) in a START operand.
- An expression does not consist of absolute terms, predefined symbols, or both.
- The statement is too complex. For example, it might have too many forward references or cause arithmetic overflow during evaluation.
- The statement is circularly defined.
- A relocatable term is multiplied or divided.

System Action: The operand of the statement is treated as zero.

Programmer Response: Correct the error if it exists. Paired relocatable symbols in different LOCTRs, even though in the same CSECT or DSECT, are not valid where an absolute, predefined value is required.

Severity: 8

ASMA155 Previous use of symbol is not this section type

Explanation: The name on a CSECT, DSECT, COM, or LOCTR statement has been used previously, on a different type of statement. For example, the name on a CSECT has been used before on a statement other than CSECT, such as a machine instruction or a LOCTR.

System Action: This name is ignored, and the statement processes as unnamed.

Programmer Response: Correct the misspelled name, or change the name to one that does not conflict.

Severity: 12

ASMA156 Only ordinary symbols, separated by commas, allowed

Explanation: The operand field of an ENTRY, EXTRN, or WXTRN statement contains a symbol that does not consist of 1-to-8 alphanumeric characters, the first being alphabetic, or the operands are not separated by a comma.

System Action: The operand in error is ignored. If other operands follow, they process normally.

Programmer Response: Supply a correct symbol or insert the missing comma. If you want an expression as an ENTRY statement operand (such as SYMBOL+4), use an EQU statement to define an additional symbol.

Severity: 12

ASMA157 Operand must be a simply-relocatable expression

Explanation: If there is another message with this statement, this message is advisory. If this message appears alone, the operand of an ORG or END statement is not a simple relocatable expression, is too complex, or is circularly defined. The error might also be that the END operand symbol is not in a CSECT.

System Action: An ORG statement or the operand of an END statement is ignored.

Programmer Response: If an error exists, supply a correct expression. Paired relocatable symbols in different LOCTRs, even though in the same CSECT or DSECT, might cause circular definition when used in an ORG statement.

Severity: 12

ASMA158 Operand expression is defective; set to *

Explanation: The first operand of an EQU statement is defective. If another message appears with this statement, this message is advisory. If this message appears alone, one of the following errors has occurred:

- The statement is too complex. For example, it has too many forward references or causes an arithmetic overflow during evaluation.
- The statement is circularly defined.
- The statement contains a relocatable term that is multiplied or divided.

System Action: The symbol in the name field is equated to the current value of the location counter (*), and operands 2 and 3 of the statement, if present, are ignored.

Programmer Response: If an error exists, supply a correct expression for operand 1 of the statement.

Severity: 8

ASMA159 Operands must be absolute, proper multiples of 2 or 4

Explanation: The combination of operands of a CNOP statement is not one of the following valid combinations:

```
0,4        2,4
0,8        2,8
4,8        6,8
```

System Action: The statement is ignored. However, the location counter is adjusted to a halfword boundary.

Programmer Response: Supply a valid combination of CNOP operands.

Severity: 12

ASMA161 Only one TITLE statement can have a name field

Explanation: More than one TITLE statement has a name field. The named TITLE statement need not be the first one in the assembly, but it must be the only one named.

System Action: The name on this TITLE statement is ignored. The name used for deck identification is taken from the first named TITLE statement encountered.

Programmer Response: Delete the unwanted name.

Severity: 4

ASMA162 PUNCH operand exceeds 80 columns; ignored

Explanation: A PUNCH statement attempted to punch more than 80 characters into a record.

System Action: The statement is ignored. The record is not punched.

Programmer Response: Shorten the operand to 80 characters or fewer or use more than one PUNCH statement.

Severity: 12

ASMA163 Operand not properly enclosed in quotes

Explanation: The operand of a PUNCH or TITLE statement does not begin with a quotation mark, or the operand of a PUNCH, MNOTE, or TITLE statement does not end with a quotation mark, or the ending quotation mark is not followed by a blank.

System Action: The statement is ignored.

Programmer Response: Supply the missing quotation mark. Be sure that a quotation mark to be punched or printed as data is represented as two quotation marks.

Severity: 4

ASMA164 Operand is a null string - record not punched

Explanation: A PUNCH statement does not have any characters between its two single quotation marks, or a single quotation mark to be punched as data is not represented by two single quotation marks.

System Action: The statement is ignored.

Programmer Response: Correct the operand. If you want to “punch” a blank record, the operand of the PUNCH statement should be a blank enclosed in single quotation marks.

Severity: 4

ASMA165 Unexpected name field

Explanation: The name field on this statement is not blank and is not a sequence symbol. The name field can not be an ordinary symbol.

System Action: The name is equated to the current value of the location counter (*). However, if no control section has been started, the name is equated to zero.

Programmer Response: Remove the name field, or ensure the name is preceded with a period if you want it to be a sequence symbol.

Severity: 4

ASMA166 Sequence symbol too long

Explanation: A sequence symbol contains more than 62 characters following the period.

System Action: If the sequence symbol is in the name field, the statement is processed without a name. If it is in the operand field of an AIF or AGO statement, the whole statement is ignored.

Programmer Response: Shorten the sequence symbol.

Severity: 12

ASMA167 Required name missing

Explanation: This statement requires a name and has none. The name field might be blank because an error occurred during an attempt to create the name by substitution or because a sequence symbol was used as the name.

System Action: The statement is ignored.

Programmer Response: Supply a valid name or ensure that a valid name is created by substitution. If a sequence symbol is needed, put it on an ANOP statement ahead of this one and put a name on this statement.

Severity: 8

ASMA168 Undefined sequence symbol - *xxxxxxxx*

Explanation: The sequence symbol in the operand field of an AIF or AGO statement outside a macro definition is not defined; that is, it does not appear in the name field of an associated statement.

System Action: This statement is ignored; assembly continues with the next statement.

Programmer Response: If the sequence symbol is misspelled or omitted, correct it. When the sequence symbol is not previously defined, the assembler looks ahead for the definitions. The lookahead stops when an END statement or an OPSYN equivalent is encountered. Be sure that OPSYN statements and macro definitions that redefine END precede possible entry into look-ahead.

Severity: 16

ASMA170 Interlude error-logging capacity exceeded

Explanation: The table that the interlude phase of the assembler uses to keep track of the errors it detects is full. This does not stop error detection by other phases of the assembler.

System Action: If there are additional errors, normally detected by the interlude phase, in other statements either before or after this one, they are not flagged. Statement processing depends on the type of error.

Programmer Response: Correct the indicated errors, and run the assembly again to diagnose any further errors.

Severity: 12

ASMA171 Standard value too long

Explanation: The standard (default) value of a keyword parameter on a macro prototype statement is longer than 255 characters.

System Action: The parameter in error and the following parameters are ignored.

Programmer Response: Shorten the standard value.

Severity: 12

ASMA172 Negative duplication factor; default = 1

Explanation: The duplication factor of a SETC statement is negative.

System Action: The duplication factor is given a default value of 1.

Programmer Response: Supply a positive duplication factor.

Severity: 8

ASMA173 Delimiter error, expected blank

Explanation: Another character, such as a comma or a quotation mark, is used where a blank (end of operand) is required.

System Action: A machine instruction assembles as zero. An ORG statement is ignored. For an EQU or END statement, the incorrect delimiter is ignored and the operand processes normally. For a CNOP statement, the location counter is aligned to a halfword boundary.

Programmer Response: Replace the incorrect delimiter with a blank. Look for an extra operand or a missing left parenthesis.

Severity: 12

ASMA174 Delimiter error, expected blank or comma

Explanation: Another character, such as a quotation mark or ampersand, is used where a blank or a comma is required.

System Action: A machine instruction assembles as zero. For a USING or DROP statement, the incorrect delimiter is ignored and the operand is processed normally.

Programmer Response: Replace the incorrect delimiter with a blank or a comma. Look for an extra operand or a missing left parenthesis.

Severity: 12

ASMA175 Delimiter error, expected comma

Explanation: Another character, such as a blank or a parenthesis, is used where a comma is required.

System Action: A machine instruction assembles as zero. For a CNOP statement, the location counter is aligned to a halfword boundary.

Programmer Response: Replace the incorrect delimiter with a comma. Be sure each expression is syntactically correct and that no parentheses are omitted.

Severity: 12

ASMA176 Delimiter error, expected comma or left parenthesis

Explanation: Another character, such as a blank or a right parenthesis, is used in a machine instruction where a comma or a left parenthesis is required.

System Action: The machine instruction assembles as zero.

Programmer Response: Replace the incorrect delimiter with a comma or a left parenthesis. Look for syntax or a base that are not correct or length fields on the first operand.

Severity: 12

ASMA177 Delimiter error, expected blank or left parenthesis

Explanation: Another character, such as a comma or a right parenthesis, is used in a machine instruction when a blank or a left parenthesis is required.

System Action: The machine instruction assembles as zero.

Programmer Response: Replace the incorrect delimiter with a blank or a left parenthesis. Look for incorrect punctuation or incorrect length, index, or base field.

Severity: 12

ASMA178 Delimiter error, expected comma or right parenthesis

Explanation: Another character, such as a blank or a left parenthesis, is used in a machine instruction when a comma or a right parenthesis is required.

System Action: The machine instruction assembles as zero.

Programmer Response: Replace the incorrect delimiter with a comma or a right parenthesis. Look for a missing base field.

Severity: 12

ASMA179 Delimiter error, expected right parenthesis

Explanation: Another character, such as a blank or a comma, is used in a machine instruction when a right parenthesis is required.

System Action: The machine instruction assembles as zero.

Programmer Response: Replace the incorrect delimiter with a right parenthesis. Look for an index field used where it is not allowed.

Severity: 12

ASMA180 Operand must be absolute

Explanation: The operand of a SPACE or CEJECT statement or the first, third, or fourth operand of a CCW statement is not an absolute term.

System Action: A SPACE or CEJECT statement is ignored. A CCW statement assembles as zero.

Programmer Response: Supply an absolute operand. Paired relocatable terms can span LOCTRs but must be in the same control section.

Severity: 12

ASMA181 CCW operand value is outside allowable range

Explanation: One or more operands of a CCW statement are not within the following limits:

- 1st operand—0 to 255
- 2nd operand—0 to 16 777 215 (CCW, CCW0); or 0 to 2 147 483 647 (CCW1)
- 3rd operand—0-255 and a multiple of 8
- 4th operand—0-65 535

System Action: The CCW assembles as zero.

Programmer Response: Supply valid operands.

Severity: 12

ASMA182 Operand 2 must be absolute, 0-65535; ignored

Explanation: If there is another message with this statement, this message is advisory. If this message appears alone, the second operand of an EQU statement contains one of the following errors:

- It is not an absolute term or expression whose value is within the range of 0 to 65,535.
- It contains a symbol that is not previously defined.
- It is circularly defined.
- It is too complex; for example, it causes an arithmetic overflow during evaluation.
- It is derived from an absolute value.

System Action: Operand 2 is ignored, and the length attribute of the first operand is used. If the third operand is present, it processes normally.

Programmer Response: Correct the error if it exists. Paired relocatable symbols in different LOCTRs, even though in the same CSECT, are not valid where an absolute, predefined value is required.

Severity: 8

ASMA183 Operand 3 must be absolute, 0-255; ignored

Explanation: If there is another message with this statement, this message is advisory. If this message appears alone, the third operand of an EQU statement contains one of the following errors:

- It is not an absolute term or expression whose value is within the range of 0 to 255.
- It contains a symbol that is not previously defined.
- It is circularly defined.
- It is too complex; for example, it causes an arithmetic overflow during evaluation.

System Action: The third operand is ignored, and the type attribute of the EQU statement is set to U.

Programmer Response: Correct the error if it exists. Note that paired relocatable symbols in different LOCTRs, even though in the same CSECT, are not valid where an absolute, predefined value is required.

Severity: 8

ASMA184 COPY disaster

Explanation: The assembler copied a library member (processed a COPY statement) while looking ahead for attribute references. However, when the complete text was analyzed, the COPY operation code had been changed by an OPSYN statement or read by an AREAD statement, and the COPY should not have been processed. (Lookahead phase ignores OPSYN statements.) This message follows the first record of the COPY code.

System Action: The library member assembles. If it included an ICTL statement, the format of that ICTL is used.

Programmer Response: Move COPY statements, or OPSYN statements that modify the meaning of COPY, to a point in the assembly before the possible entry into look-ahead mode.

Severity: 16

ASMA185 Operand 2 is erroneous - *xxxxxxxx*

Explanation: The second operand is incorrect, or two operands appear where there should be only one.

System Action: The second operand is ignored.

Programmer Response: Remove or correct the second operand.

Severity: 4

ASMA186 AMODE/RMODE already set for this ESD item

Explanation: A previous AMODE instruction has the same name field as this AMODE instruction, or a previous RMODE instruction has the same name field as this RMODE instruction.

System Action: The instruction in error is ignored.

Programmer Response: Remove the conflicting instruction or specify the name of another control section.

Severity: 8

ASMA187 The name field is invalid - *xxxxxxxx*

Explanation: The name field of an AMODE instruction does not refer to a valid control section in this assembly, or the name field of an RMODE instruction does not refer to a valid control section in this assembly.

System Action: The instruction in error is ignored, and the name field does not appear in the cross-reference listing.

Programmer Response: Specify a valid control section in the name field of the AMODE or RMODE instruction.

Severity: 8

ASMA188 Incompatible AMODE and RMODE attributes

Explanation: A previous AMODE 24 instruction has the same name field as this RMODE ANY instruction, or a previous RMODE ANY instruction has the same name field as this AMODE 24 instruction.

System Action: The instruction in error is ignored.

Programmer Response: Change the AMODE and RMODE attributes so they are no longer incompatible. All combinations except AMODE 24 and RMODE ANY are valid.

Severity: 8

ASMA189 OPSYN not permitted for REPRO

Explanation: REPRO is specified in either the name field or the operand field of an OPSYN instruction, but a REPRO statement has been previously encountered in the source module. Once a REPRO statement has been encountered, the REPRO symbolic operation code cannot be redefined using the OPSYN instruction.

System Action: The OPSYN instruction is ignored.

Programmer Response: Remove the OPSYN instruction, or remove the previously encountered REPRO statement.

Severity: 8

ASMA190 CATTR instruction invalid because no section started

Explanation: A CATTR instruction must be preceded by a CSECT, START, or RSECT instruction.

System Action: The CATTR instruction is ignored.

Programmer Response: Remove the CATTR instruction, or precede it with a CSECT, START, or RSECT instruction.

Severity: 8

ASMA191 CATTR instruction operands ignored

Explanation: You specified operands on a CATTR instruction which has the same class name as a previous CATTR instruction.

System Action: The assembler ignores the operands, and continues as if you did not specify any operands.

Programmer Response: You can correct this error by:

- Removing the operands from the CATTR instruction in error.
- Changing the class name for the CATTR instruction in error.
- Removing the CATTR instruction in error.

Severity: 4

ASMA201 SO or SI in continuation column - no continuation assumed

Explanation: When High Level Assembler is invoked with the DBCS option, the double-byte delimiters SO and SI are treated as blanks in the continuation column, and *not* as continuation indicators.

System Action: The SO or SI in the continuation column assembles as a blank, and the next line is not treated as a continuation line.

Programmer Response: If continuation is required, then rearrange the source line so that a non-blank EBCDIC character can be used to indicate continuation.

If continuation is not required, check that everything preceding the SO or SI is complete and valid data.

Severity: 4

ASMA202 Shift-in not found at extended continuation; check data truncation

Explanation: The assembler has detected an extended continuation indicator that is not on a source statement containing double-byte data. The extended continuation indicator feature is provided to permit continuation of double-byte data, and single-byte data adjacent to double-byte data. If you use extended continuation indicators anywhere else, the assembler issues this message. As this situation can be caused by a coding error, the assembler might unintentionally treat the data as extended continuation indicators.

System Action: The extended continuation indicators do not assemble as part of the operand.

Programmer Response: Change the continuation indicator if the unintentional truncation occurred.

Severity: 4

ASMA203 Unbalanced double-byte delimiters

Explanation: A mismatched SO or SI has been found. This could be the result of truncated or nested double-byte data. This error does NOT occur because valid double-byte data is truncated to fit within the explicit length specified for C-type DC, DS, and DXD statements and literals - that condition produces error ASMA208.

System Action: The operand in error, and the following operands are ignored.

Programmer Response: Correct the incorrect double-byte data.

Severity: 8

ASMA204 Invalid double-byte data

Explanation: All data between SO and SI must be valid double-byte characters. A valid double-byte character is defined as either double-byte blank (`X'4040'`), or two bytes each of which must be in the range `X'41'` to `X'FE'` inclusive.

This error does not apply to the operands of macro instructions.

System Action: The operand in error, and the following operands are ignored.

Programmer Response: Correct the incorrect double-byte data.

Severity: 8

ASMA205 Extended continuation end column must not extend into continue column

Explanation: The extended continuation indicator extended into the continue column.

System Action: The extended continuation indicator is ignored. The following record or records might be treated as incorrect. The extended continuation indicators are treated as part of the source statement.

Programmer Response: If the data in the extended continuation is to be regarded as valid input then another non-blank character must be used in the continuation indication column to identify the data as valid and to continue to the next record. If the data is not to be part of the constant then remove the characters of the extended continuation and add the correct data to the continue record to the point where the extended continuation is needed. This message might be encountered when converting code that assembled with the NODBCS option to code that is to be assembled with the DBCS option.

Severity: 8

ASMA206 G-type constant must not contain single-byte data

Explanation: A G-type constant or self-defining term, after substitution has occurred, must consist entirely of double-byte data, correctly delimited by SO and SI. If SO or SI are found in any byte position other than the first and last respectively (excepting redundant SI/SO pairs which are removed) then this error is reported.

System Action: The operand in error, and the following operands are ignored.

Programmer Response: Either remove the single-byte data from the operand, or change the constant to a C-type.

Severity: 8

ASMA207 Length of G-type constant must be a multiple of 2

Explanation: A G-type constant must contain only double-byte data. If assembled with a length modifier which is not a multiple of 2, incorrect double-byte data is created.

System Action: The operand in error, and the operands following are ignored.

Programmer Response: Either correct the length modifier, or change the constant to a C-type.

Severity: 8

ASMA208 Truncation into double-byte data is not permitted

Explanation: The explicit length of a C-type constant in a DS, DC or DXD statement or literal must not cause the nominal value to be truncated at any point within double-byte data.

System Action: The operand in error, and the following operands are ignored.

Programmer Response: Either correct the length modifier, or change the double-byte data so that it is not truncated.

Severity: 8

ASMA253 Too many errors

Explanation: No more error messages can be issued for this statement, because the assembler work area in which the errors are logged is full.

System Action: If more errors are detected for this statement, the messages, annotated text, or both, are discarded.

Programmer Response: Correct the indicated errors, and rerun the assembly. If there are more errors on this statement, they will be detected in the next assembly.

Severity: 16

ASMA254 * MNOTE *****

Explanation: The text of an MNOTE statement, which is appended to this message, has been generated by your program or by a macro definition or a library member copied into your program. An MNOTE statement enables a source program or a macro definition to signal the assembler to generate an error or informational message.

System Action: None.

Programmer Response: Investigate the reason for the MNOTE. Errors flagged by MNOTE often cause the program to fail if it is run.

Severity: An MNOTE is assigned a severity code of 0 to 255 by the writer of the MNOTE statement.

ASMA300 USING overridden by a prior active USING on statement number *nnnnnn*

Explanation: The USING instruction specifies the same base address as a previous USING instruction at statement number *nnnnnn*, and the base register specified is lower-numbered than the previously specified base register.

System Action: The assembler uses the higher-numbered base register for address resolution of symbolic addresses within the USING range.

Programmer Response: Check your USING statements to ensure that you have specified the correct base address and base register and that you have not omitted a needed DROP statement for the previous base register. You can suppress this message by reducing the value specified in the WARN sub-option of the USING option by 1.

Severity: 4

ASMA301 Prior active USING on statement number *nnnnnn* **overridden by this USING**

Explanation: The USING instruction specifies the same base address as a previous USING instruction at statement number *nnnnnn*, and the base register specified is higher-numbered than the previous base register.

System Action: The assembler uses the higher-numbered base register for address resolution of symbolic addresses within the USING range.

Programmer Response: Check your USING statements to ensure that you have specified the correct base address and base register and that you have not omitted a needed DROP statement for the previous base register. You can suppress this message by reducing the value specified in the WARN sub-option of the USING option by 1.

Severity: 4

ASMA302 USING specifies register 0 with a non-zero absolute or relocatable base address

Explanation: The assembler assumes that when register 0 is used as a base register, it contains zero. Therefore, regardless of the value specified for the base address, displacements are calculated from base 0.

System Action: The assembler calculates displacements as if the base address specified were absolute or relocatable zero.

Programmer Response: Check the USING statement to ensure you have specified the correct base address and base register. You can suppress this message by reducing the value specified in the WARN suboption of the USING option by 2.

Severity: 4

ASMA303 Multiple address resolutions may result from this USING and the USING at statement number *nnnnnn*

Explanation: The USING instruction specifies a base address that lies within the range of an earlier USING instruction at statement number *nnnnnn*. The assembler might use multiple base registers when resolving implicit addresses within the range overlap.

System Action: The assembler computes displacements from the base address that gives the smallest displacement, and uses the corresponding base register when it assembles addresses within the range overlap.

Programmer Response: Check your USING instructions for unintentional USING range overlaps and check that you have not omitted a needed DROP statement. You can suppress this message by reducing the value specified in the WARN suboption of the USING option by 4.

Severity: 4

ASMA304 Displacement exceeds LIMIT value specified

Explanation: The address referred to by this statement has a valid displacement that is higher than the displacement limit specified in the USING(LIMIT(xxx)) option.

System Action: The instruction assembles correctly.

Programmer Response: This error diagnostic message is issued by your request. You can suppress this message by reducing the value specified in the WARN suboption of the USING option by 8.

Severity: 4

ASMA305 Operand 1 does not refer to location within reference control section

Explanation: The first operand in a dependent USING statement does not refer to a location within a reference control section defined by a DSECT, DXD, or COM instruction.

System Action: The USING statement is ignored.

Programmer Response: Change the USING statement to specify a location within a reference control section.

Severity: 8

ASMA310 Name already used in prior ALIAS - *xxxxxxxx*

Explanation: The name specified in the ALIAS statement has already been used in a previous ALIAS statement.

System Action: The statement is ignored.

Programmer Response: Change the program so that the name is used in only one ALIAS statement.

Severity: 4

ASMA311 Illegal ALIAS string

Explanation: The ALIAS string is illegal for one of the following reasons:

- The string is null.
- The string is not in the form `C'cccccccc'` or `X'hhhhhhhh'`.
- The string is in the form `X'hhhhhhhh'` but an odd number of hexadecimal digits has been specified.
- The string contains a character outside the valid range of `X'42'` to `X'FE'`.
- The string has been used in the name entry on a previous CSECT, DSECT, COM or LOCTR instruction.

System Action: The statement is ignored.

Programmer Response: Change the program so that the string conforms to the required syntax.

Severity: 8

ASMA312 ALIAS name is not declared as an external symbol - *xxxxxxxx*

Explanation: The name specified on the ALIAS statement is not declared as an external symbol, either explicitly via an EXTRN, CSECT, etc., or implicitly via a V-type constant.

System Action: The statement is ignored.

Programmer Response: Change the program so that the name is declared as an external symbol.

Severity: 8

ASMA400 Error in invocation parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* is not a recognized assembler option, or is incorrectly specified.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, using the installation default value for the erroneously specified option.

Programmer Response: Correct the parameter error and resubmit the assembly.

Severity: 4

ASMA401 Fixed option cannot be overridden by invocation parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* cannot be specified as an invocation parameter because the option it is attempting to override was fixed when High Level Assembler was installed.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, using the installation default value for the erroneously specified option.

Programmer Response: Correct the parameter error and resubmit the assembly.

Severity: 2

ASMA402 Invalid print line length *xxxxxx* returned by LISTING exit; exit processing bypassed

Explanation: When invoked with an OPEN request, the LISTING exit specified a print line length that was either outside the range 121 to 255, or was not permitted for the device to which the listing file is assigned.

System Action: The assembler bypasses the exit when processing listing records, and writes the assembly listing to the standard listing file. The print line length is determined by the assembler.

Programmer Response: Correct the error in the LISTING exit.

Severity: 4

ASMA403 WORK file blocksize has been set to *xxxxxx*

Explanation: The blocksize specified in the job control language for the work file is not permitted. The valid range is 2008 bytes to 32760 bytes, or the maximum track capacity for the device on which the work file resides, whichever is lesser.

System Action: The blocksize for the work file has been set to the specified value.

Programmer Response: Supply a valid blocksize for the work file.

Severity: 4

ASMA404 Invalid term line length *xxxxxx* returned by TERM exit; exit processing bypassed

Explanation: When invoked with an OPEN request, the TERM exit specified a line length that was either zero or greater than 255, or was not permitted for the device to which the terminal file is assigned.

System Action: The assembler bypasses the exit when processing terminal records, and writes the terminal records to the standard terminal file. The line length is determined by the assembler.

Programmer Response: Correct the error in the TERM exit.

Severity: 4

ASMA410 WORK file not defined to the assembler

Explanation: JCL statements for the assembler work file has not been provided in the job control language for the assembly job step.

- If you are running the assembler under MVS, the DD statement for the work file is missing, or the TSO ALLOCATE command has not been issued.
- If you are running the assembler under CMS, the FILEDEF command for the work file has not been issued.

System Action: The assembler attempts to complete the assembly in virtual storage, without using the work file. However, if there is not enough virtual storage for the assembly to complete, another message is issued and the assembly ends abnormally.

Programmer Response: Supply valid JCL for the work file. Check whether your installation has changed the default ddname for the work file, and ensure that you are using the correct ddname.

Severity: 4

ASMA411 WORK file is not on DASD

Explanation: The JCL statement for the work file indicates that the work file does not reside on DASD.

System Action: The assembler attempts to complete the assembly in storage, without using the work file. However, if there is not enough virtual storage for the assembly to complete, another message is issued and the assembly ends abnormally.

Programmer Response: Assign the work file (SYSUT1) to DASD and supply the correct JCL for the work file. Check whether your installation has changed the default DDname for the work file, and ensure that you are using the correct DDname.

Severity: 4

ASMA412 Unable to open WORK file

Explanation: The assembler encountered an error when attempting to open the assembler work file.

System Action: The assembler attempts to complete the assembly in storage, without using the work file. However, if there is not enough virtual storage for the assembly to complete, another message is issued and the assembly ends abnormally.

Programmer Response: Check the JCL for the work file. Ensure that the work file is assigned to DASD and that the DASD volume is not write-protected.

Severity: 4

ASMA413 Unable to open INPUT file

Explanation: The assembler encountered an error when attempting to open the assembler input file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the input file.

Severity: 16

ASMA414 Unable to open LISTING file

Explanation: The assembler encountered an error when attempting to open the assembler listing file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the listing file.

Severity: 16

ASMA415 Unable to open TERM file

Explanation: The assembler encountered an error when attempting to open the assembler terminal output file. This is usually caused by a job control language error.

System Action: The assembly continues and no terminal file is produced.

Programmer Response: Check the JCL for the terminal output file.

Severity: 2

ASMA416 Unable to open DECK file

Explanation: The assembler encountered an error when attempting to open the assembler deck output file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the deck output file.

Severity: 16

ASMA417 Unable to open OBJECT file

Explanation: The assembler encountered an error when attempting to open the assembler object output file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the object output file.

Severity: 16

ASMA418 Unable to open ADATA file

Explanation: The assembler encountered an error when attempting to open the associated data file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the SYSADATA ddname.

Severity: 16

ASMA419 Unable to open TRACE file

Explanation: The assembler encountered an error when attempting to open the internal trace file. This is usually caused by a job control language error.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check the JCL for the SYSTRACE ddname.

Severity: 16

ASMA420 Error in *PROCESS statement parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* is not a recognized assembler option, or is incorrectly specified.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, using the installation default value or the invocation parameter value for the erroneously specified option.

Programmer Response: Correct the parameter error and resubmit the assembly.

Severity: 2

ASMA421 Fixed option cannot be overridden by *PROCESS statement parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* cannot be specified as a *PROCESS statement parameter because the option it is attempting to override was fixed when High Level Assembler was installed.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, using the installation default value for the erroneously specified option.

Programmer Response: Remove the option from the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA422 Option *xxxxxxxx* **is not valid on a *PROCESS statement**

Explanation: The following options cannot be specified on a *PROCESS statement:

ADATA│NOADATA	OBJECT│NOOBJECT
ASA│NOASA	OPTABLE
DECK│NODECK	SIZE
EXIT│NOEXIT	SYSPARM
LANGUAGE	TERM│NOTERM
LINECOUNT	TRANSLATE│NOTRANSLATE
LIST│NOLIST	XOBJECT│NOXOBJECT

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, using the installation default value or the invocation parameter value for the erroneously specified option.

Programmer Response: Remove the option from the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA424 Continuation column is not blank. *PROCESS statements can not be continued.

Explanation: The continuation column (usually column 72) is not blank for a *PROCESS statement. *PROCESS statements can not be continued.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues and processes the options specified.

Programmer Response: Recode the *PROCESS statement, leaving the continuation column blank. If you need to specify more options than can fit on the *PROCESS statement, add another *PROCESS statement to your code. You can specify a maximum of 10 *PROCESS statements.

Severity: 4

ASMA425 Option conflict in invocation parameters. *yyyyyyyy* **overrides an earlier setting.**

Explanation: The option *yyyyyyyy* specified as an invocation parameter overrides an earlier setting of the option in the invocation parameters.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues using the right most of the conflicting options.

Programmer Response: Correct the parameter error and resubmit the assembly.

Severity: 2

ASMA426 Option conflict in *PROCESS statements. *yyyyyyyy* **overrides an earlier setting.**

Explanation: The option *yyyyyyyy* specified on an *PROCESS statement overrides an earlier setting of the option on the same statement or a previous *PROCESS statement.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues using the last conflicting option encountered.

Programmer Response: Correct the *PROCESS statement error and resubmit the assembly.

Severity: 2

ASMA429 SYSPRINT LRECL should be at least 133 when XOBJECT option is specified

Explanation: The XOBJECT assembler option has been specified, however the logical record length of the listing file, SYSPRINT, is less than 133.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues, however the lines in the *source and object* section are truncated.

Programmer Response: Specify a record length of 133 for SYSPRINT.

Severity: 4

ASMA430 Continuation statement does not start in continue column.

Explanation: The operand on the continued record ends with a comma and a continuation statement is present but the continue column is blank. The continue column is column 16, unless you redefined it with an ICTL instruction.

System Action: Any remaining continuation lines belonging to this statement are ignored.

Programmer Response: Check that the continuation was coded as intended.

Severity: 4

ASMA431 Continuation statement may be in error - continuation indicator column is blank.

Explanation: A list of one or more operands ends with a comma, but the continuation indicator column is blank. The continuation indicator column is column 72, unless you redefined it with an ICTL instruction.

System Action: The next statement assembles as a standard assembler source statement.

Programmer Response: Check that the continuation was coded as intended.

Severity: 4

ASMA432 Continuation statement may be in error - comma omitted from continued statement.

Explanation: The continuation record starts in the continue column (usually column 16) but there is no comma present following the operands on the previous record.

System Action: Any remaining continuation lines belonging to this statement are ignored.

Programmer Response: Check that the continuation was coded as intended.

Severity: 4

ASMA433 Statement not continued - continuation statement may be in error

Explanation: The continued record is full but the continuation record does not start in the continue column (usually column 16).

System Action: Any remaining continuation lines belonging to this statement are ignored.

Programmer Response: Check that the continuation was coded as intended.

Severity: 4

ASMA434 XOBJECT option specified, option LIST(133) will be used

Explanation: You specified the XOBJECT option, and the LIST suboption is 121.

System Action: The assembler sets the LIST suboption to 133. If option PESTOP is specified, the assembly is stops. If option NOPESTOP is specified, the assembly continues.

Programmer Response: To prevent this warning message, run the assembly again specifying XOBJECT and LIST(133).

Severity: 2

ASMA435 Record *n* **in** *xxxxxxx* **on volume:** *vvvvvv*

Explanation: The data set *xxxxxxxx* which is located on volume serial *vvvvvv*, contains an error on record number *n*. The volume serial might not be available.

System Action: See the System Action section of the error message(s) which immediately precede this message.

Programmer Response: Refer to the Programmer Response section of the error messages which immediately precede this message.

Severity: 0

ASMA436 Attempt to override invocation parameter in *PROCESS statement. Option *yyyyyyyy* **ignored.**

Explanation: The option *yyyyyyyy* specified in a *PROCESS statement overrides the option specified in an invocation parameter.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues using the option specified in the invocation parameters.

Programmer Response: Remove the option from the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA437 Attempt to override invocation parameter in *PROCESS statement. Suboption yyyyyyyy of xxxxxxxx option ignored.

Explanation: The suboption *yyyyyyyy* of option *xxxxxxxx* specified in a *PROCESS statement overrides the suboption specified in an invocation parameter.

System Action: If option PESTOP is specified, the assembly stops. If option NOPESTOP is specified, the assembly continues using the suboption specified in the invocation parameters.

Programmer Response: Remove the option or suboption from the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA700 *exit-type***:** *exit supplied text*

Explanation: The user supplied exit for *exit-type* exit has requested the assembler to issue this message with the *exit supplied text*.

System Action: None

Programmer Response: Check the user exit documentation for the cause of this message and for the correct response.

Severity: 0

ASMA701 *exit-type : exit supplied text*

Explanation: The user supplied exit for *exit-type* exit has requested the assembler to issue this message with the *exit supplied text*.

System Action: The assembly continues.

Programmer Response: Check the user exit documentation for the cause of this message and for the correct response.

Severity: 4

ASMA702 *exit-type* **:** *exit supplied text*

Explanation: The user supplied exit for *exit-type* exit has requested the assembler to issue this message with the *exit supplied text*.

System Action: None

Programmer Response: Check the user exit documentation for the cause of this message and for the correct response.

Severity: 8

ASMA703 *exit-type* **:** *exit supplied text*

Explanation: The user supplied exit for *exit-type* exit has requested the assembler to issue this message with the *exit supplied text*.

System Action: None

Programmer Response: Check the user exit documentation for the cause of this message and for the correct response.

Severity: 12

ASMA704 *exit-type* **:** *exit supplied text*

Explanation: The user supplied exit for *exit-type* exit has requested the assembler to issue this message with the *exit supplied text*.

System Action: None

Programmer Response: Check the installation documentation for the cause of this message and for the correct response.

Severity: 16

Abnormal Assembly Termination Messages

Whenever an assembly cannot complete, High Level Assembler provides a message and, in some cases, a specially formatted dump for diagnostic information. This might indicate an assembler malfunction or it might indicate a programmer error. The statement causing the error is identified and, if possible, the assembly listing up to the point of the error is printed. The messages in this book give enough information to enable you to correct the error and reassemble your program, or to determine that the error is an assembler malfunction.

Messages

ASMA931 Unable to load specified operation code table - *xxxxxxxx*

Explanation: The assembler attempted to load the named operation code table, but the load failed.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check that the specified operation code table is in a library accessible by the assembler.

Severity: 20

ASMA932 Unable to load specified EXIT module - *xxxxxxxx*

Explanation: The assembler attempted to load the named exit module, but the load failed.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check that the specified exit module is in a library accessible by the assembler.

Severity: 20

ASMA933 UNABLE TO LOAD SPECIFIED MESSAGES MODULE - *xxxxxxxx*

Explanation: The assembler attempted to load the named messages module, but the load failed. The name of the messages module is determined from the value specified in the LANGUAGE option.

Note: This message is only produced in uppercase English.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check that you have correctly specified the correct messages module using the LANGUAGE option, and that the specified messages module is in a library accessible by the assembler.

Severity: 20

ASMA934 UNABLE TO LOAD DEFAULT OPTIONS MODULE - *xxxxxxxx*

Explanation: The assembler attempted to load the named default options module, but the load failed.

Note: This message is only produced in uppercase English.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check that the default options module is in a library accessible by the assembler.

Severity: 20

ASMA935 One or more required files not available

Explanation: The assembler encountered an error when attempting to open a required file.

System Action: Before this message is issued, one or more associated messages are issued that describe which file or files could not be opened. After this message is issued, the assembly stops.

Programmer Response: Check the associated message or messages.

Severity: 20

ASMA936 Assembly terminated due to errors in invocation parameters

Explanation: The assembler detected an error in one or more of the parameters specified when the assembler was invoked, and the installation default value for the PESTOP assembler option is YES.

System Action: Before this message is issued, one or more associated messages are issued that describe which parameter or parameters were in error. After this message is issued, the assembly stops.

Programmer Response: Check the associated message or messages. Invoke the assembler with correct invocation parameters. Do not attempt to override the fixed installation defaults.

Severity: 20

ASMA937 Unable to load specified translation table - *xxxxxxxx*

Explanation: The assembler attempted to load the translation table called *xxxxxxxx*, but the load failed. The name of the translation table is determined from the value specified in the TRANSLATE option.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check you have correctly specified the translation table module using the TRANSLATE option, and the module is in a library accessible by the assembler.

Severity: 20

ASMA938 Module *xxxxxxxx* is not a valid translation table

Explanation: The translation table specified in the TRANSLATE option is not valid.

System Action: The assembly stops.

Programmer Response: Ensure the translation table is generated according to the instructions described in the *Programmer's Guide*.

Severity: 20

ASMA939 Unable to load external function module - xxxxxxxx

Explanation: The assembler attempted to load the external function module xxxxxxxx, but the load failed.

System Action: The assembly stops and no listing is produced.

Programmer Response: Check that the specified module is in a library accessible by the assembler, and that the external function name has been spelled correctly in the SETAF or SETCF statement.

Severity: 20

ASMA940 *exit-type* **has requested termination during** *operation* **processing; exit error text: < none |** *error text* **>**

Explanation: The user supplied exit for *exit-type* failed when processing an *operation* request. The exit might have provided *error text* to assist in determination of the failure.

System Action: The assembly stops.

Programmer Response: Check the specified exit program for the cause of failure.

Severity: 20

ASMA941 *external function name* **has requested termination during processing.**

Explanation: The user supplied external function *external function name* failed during processing.

System Action: The assembly stops.

Programmer Response: Check the specified external function program for the cause of failure.

Severity: 20

ASMA942 ASMADOPT MODULE IS NOT IN RELEASE 2 FORMAT

Explanation: The default options module ASMADOPT is not in the required format for Release 2.

Note: This message is only produced in uppercase English.

System Action: The assembly terminates

Programmer Response: Ensure that you have the correct version of the ASMADOPT module available. You might need to reassemble your default options module with the ASMAOPT macro provided with High Level Assembler Release 2.

Severity: 20

ASMA943 Unable to find listing header ***nnn***

Explanation: The assembler tried to produce a heading line in the assembler listing but could not find the heading. This can be caused if the assembler load module has been corrupted.

System Action: The assembly is aborted.

Programmer Response: Reassemble the program; it might assemble correctly. If it does not reassemble without error, save the output from the assembly, and the input deck and give them to your IBM service representative.

Severity: 20

ASMA944 LOAD OF ASMA93 MODULE FAILED; INSUFFICIENT MAIN STORAGE OR MODULE NOT FOUND

Explanation: The assembler attempted to load the module ASMA93, but the load failed either because there was insufficient main storage available to complete the load, or the module could not be found.

Note: This message is only produced in uppercase English.

System Action: The assembly stops and no listing is produced.

Programmer Response: Under MVS, ensure that the correct High Level Assembler load library is available in the standard load module search order. If it is, consider increasing the region size.

Under CMS, ensure that the correct mini disk containing the High Level Assembler modules is being accessed. If it is, consider increasing your virtual machine storage size.

Severity: 20

ASMA950 End of statement flag was expected in Macro Edited Text, but was not found - MACRO EDITOR is suspect
ASMA951 The MACRO GENERATOR has encountered untranslatable Macro Edited Text
ASMA952 Bad SET symbol name field or LCL/GBL operand - check the Macro Edited Text
ASMA953 Bad subscript on SET symbol - check the Macro Edited Text
ASMA954 Character expression followed by bad subscripts - check the Macro Edited Text
ASMA955 A right parenthesis with no matching left parenthesis was found in an expression - check the Macro Edited Text
ASMA956 Multiple subscripts or bad SET symbol terminator - check the Macro Edited Text
ASMA957 Bad terminator on created SET symbol - check the Macro Edited Text
ASMA958 Bad terminator on parameter - check the Macro Edited Text
ASMA959 Unexpected end of data on WORK file - internal storage management suspect
ASMA960 A bad internal file number has been passed to the ***xxxxxxxx*** **internal storage management routine**
ASMA961 An invalid storage request has been made, or the free storage chain pointers have been destroyed
ASMA962 A zero block address or bad block number has been passed to an internal storage management routine
ASMA963 Invalid pointer at entry to utility routine
ASMA964 Macro Edited Text Flag is not ICTL

Explanation: The assembly stops because of one of the errors described in ASMA950 through ASMA964. This usually is caused by an error in the assembler itself. Under certain conditions, however, the assembly can be rerun successfully.

System Action: The assembly stops and a formatted abnormal termination dump is produced. Depending on where the error occurred, the assembly listing up to the failing statement might also be produced. The dump usually indicates which statement was being processed at the time of abnormal termination. It also might include contents of the assembler registers and work areas and other status information for use by an IBM support representative.

Programmer Response: Check the statement that was being processed at the time of abnormal termination. Correct any errors in it or, if the statement is long or complex, rewrite it. Reassemble the program; it might assemble correctly. However, even if the program assembles correctly, there might be a problem with the assembler. Save the abnormal termination dump, the assembly listing (if one was produced), and the source program, and contact IBM for support.

Severity: 20

ASMA970 Statement complexity exceeded, break the statement into segments, and rerun the assembly

Explanation: The statement is too complex to be evaluated by the macro generator phase of the assembler. It overflowed the evaluation work area of the assembler. Normally, there is no assembler malfunction; the statement can be corrected and the program reassembled successfully.

System Action: A special abnormal termination dump (High Level Assembler interrupt and diagnostic dump) follows the message. The statement causing termination is SETA, SETB, SETC, AGO, or AIF. The dump does not indicate which statement caused termination; however, it might show the last statement generated in the macro. The dump might also include contents of the assembler registers and work areas and other status information for use by IBM or your assembler maintenance programmers in determining the cause of the termination. However, it is not needed unless the error persists. This information could be helpful in diagnosing and fixing an assembler error.

Programmer Response: Check the statement that caused termination. Rewrite the statement or split it into two or more statements. Reassemble the program; it should assemble correctly. However, if the error persists, there might be an assembler malfunction. Save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and give them to your IBM program support representative.

Severity: 20

ASMA971 Insufficient storage available for Macro Editor work area

ASMA972 Virtual storage exhausted; increase the SIZE option

Explanation: The size of the dynamic storage area allocated for assembler buffer areas, tables, and work areas, as specified in the SIZE option, is not enough for the assembly to complete.

System Action: A special abnormal termination dump (High Level Assembler interrupt and diagnostic dump) follows the message. The dump usually indicates the statement being processed when the assembler determined there was not enough dynamic storage available to continue. Depending on where the error occurred, the assembly listing up to the statement being processed might also be produced. The other information in the dump, such as register and work area contents, is not needed.

Programmer Response: Increase the value specified in the SIZE option, or split the assembly into two or more assemblies. Check for conditional assembly language loops in open code that could cause the symbol table to overflow.

Severity: 20

ASMA973 WORK file maximum block count exceeded

Explanation: The maximum block count of 65,535 has been exceeded for SYSUT1.

System Action: The assembly stops and no listing is produced.

Programmer Response: Increase the work file block size, or split the assembly into two or more smaller assemblies.

Severity: 20

ASMA974 Insufficient storage available to satisfy the SIZE option

Explanation: The assembler attempted to acquire the amount of storage specified in the SIZE option, but there was not enough available storage below the 16MB line in the region (MVS) or virtual machine (CMS).

System Action: The assembly stops and no listing is produced.

Programmer Response: Increase the region size (MVS) or virtual machine size (CMS), or reduce the size requested in the SIZE option.

Severity: 20

ASMA975 SIZE option specifies insufficient storage for assembly

Explanation: The SIZE option was specified as MAX-*nnn*K or MAX-*nn*M, but the amount of storage available to the assembler using this formula is not enough for the assembly to continue. The assembler requires a minimum of either 200K bytes or 10 times the work file blocksize, plus 20K, of working storage in the region (MVS), or virtual machine (CMS) to proceed.

System Action: The assembly stops and no listing is produced.

Programmer Response: Increase the region size (MVS) or virtual machine size (CMS), or reduce the amount of storage to be reserved in the MAX-*nnn*K or MAX-*nn*M form of the SIZE option.

Severity: 20

ASMA976 Statement too complex for expression analysis

Explanation: The statement is too complex to be analyzed by the expression analysis routine of the assembler. It overflowed the analysis work area. The size of the analysis work area is the same as the work file block size. Normally, there is no problem with the assembler. The statement can be rewritten to simplify it, and the program reassembled successfully.

System Action: The assembly stops and a formatted abnormal termination dump is produced. The dump indicates which statement was being processed at the time of abnormal termination. It also includes the contents of the assembler registers and work areas and other status information that might be required by an IBM support representative if the problem persists.

Programmer Response: Check the statement that was being processed at the time of abnormal termination. Rewrite the statement or split it into two or more statements. Alternatively, increase the work file block size. Reassemble the program; it should assemble correctly. However, if the problem persists, there might be a problem with the assembler. Save the abnormal termination dump, the assembly listing (if one was produced), and the source program, and contact IBM for support.

Severity: 20

ASMA990 Location Counter does not match symbol table value

Explanation: A difference has been detected between the symbol table and the location counter. The assembly stops and a special abnormal termination dump (High Level Assembler interrupt and diagnostic dump) is taken. The listing is not completed.

System Action: The High Level Assembler interrupt and diagnostic dump shows the statement that was being printed when the difference between the location counter and the symbol table was detected.

Programmer Response: Reassemble the program using NOALIGN. If alignment is needed, use CNOP or DS to force alignment.

Severity: 20

ASMA998 The assembler could not resume reading a LIBRARY member because it could not find the member again

Explanation: The assembly stops, because the assembler cannot find a COPY member that it has already read. This usually is caused by an error in the assembler itself or by an Operating System I/O error. Under certain conditions, however, the assembly can be rerun successfully.

System Action: A special abnormal termination dump (High Level Assembler interrupt and diagnostic dump) follows the message. The dump usually indicates which statement caused termination. It also might include contents of the assembler registers and work areas and other status information for use by IBM or your assembler maintenance programmers in determining the cause of the termination.

Programmer Response: Reassemble the program; it might assemble correctly. If it does not reassemble without error, save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and contact your IBM service representative.

Severity: 20

ASMA999 Assembly terminated - SYNAD exit taken - permanent I/O error on xxxxxxx data set

Explanation: The assembly was stopped because of a permanent I/O error on the data set indicated in the message. This is usually caused by a machine or an operating system error. The assembly usually can be rerun successfully. This message also appears on the console output device.

System Action: A special abnormal termination dump (High Level Assembler interrupt and diagnostic dump) follows the message. Depending on where the error occurred, the assembly listing up to the bad statement might also be produced. The dump usually indicates which statement caused termination. It also might include contents of the assembler registers and work areas and other status information for use by IBM or your assembler maintenance programmers in determining the cause of the termination.

Programmer Response: If the I/O error is on SYSIN or SYSLIB, you might have concatenated the input or library data sets incorrectly. Make sure that the DD statement for the data set with the largest block size (BLKSIZE) is placed in the JCL before the DD statements of the data sets concatenated to it. Also, make sure that all input or library data sets have the same device class (all DASD or all tape).

Reassemble the program; it might assemble correctly. If it does not reassemble without error, save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and give them to your IBM service representative. Also, if the program assembles correctly, submit a copy of the listing and input deck of the correct assembly.

Severity: 20

ASMAHL Command Error Messages

ASMACMS002E File *fn ft fm* **not found**

Explanation: The file name you included in the ASMAHL command does not correspond to the names of any of the files on your disks.

Supplemental Information: The variable file name, file type, and file mode in the text of the message indicate the file that could not be found.

System Action: RC=28. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMAHL with the correct file name.

ASMACMS003E Invalid option *option*

Explanation: You have included an option that is not correct with your ASMAHL command.

Supplemental Information: The variable option in the text of the message indicates the option that is not correct.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the ASMAHL command, and reissue the command with the correct option.

ASMACMS004E Improperly formed option *option*

Explanation: You have included an improperly formed option with your ASMAHL command.

Supplemental Information: The variable option in the text of the message indicates the improperly formed option.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the ASMAHL command, and reissue the command with the correct option.

ASMACMS005E Truncation of options may have occurred because of tokenized PLIST format

Explanation: The options have been passed to the ASMAHL command in tokenized PLIST format. Any options passed might have been truncated to 8 characters. This message is only issued when an error has been detected in one of the options that was specified.

System Action: The options are accepted as entered but might have been truncated.

Programmer Response: If the options have been truncated, invoke the ASMAHL command with the extended parameter list. If the SYSPARM option has been truncated, specify SYSPARM(?).

ASMACMS006E No read/write disk accessed

Explanation: Your virtual machine configuration does not include a read/write disk for this terminal session, or you failed to specify a read/write disk.

System Action: RC=36. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Issue an ACCESS command specifying a read/write disk.

ASMACMS007E File '*fn ft fm***' does not contain fixed length 80 character records**

Explanation: The source file you specified in the ASMAHL command does not contain fixed-length records of 80 characters.

Supplemental Information: The variable file name, file type, and file mode in the text of the message indicate the file that is in error.

System Action: RC=32. The command cannot be processed.

Programmer Response: You must reformat your file into the correct record length. CMS XEDIT or COPYFILE can be used to reformat the file.

ASMACMS010E file name omitted and FILEDEF '*ddname***' is undefined**

Explanation: You have not included a file name in the ASMAHL command, and no FILEDEF could be found for the *ddname* specified.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMAHL command and specify a file name, or issue a FILEDEF for the *ddname* specified.

ASMACMS011E file name omitted and FILEDEF '*ddname***' is not for DISK.**

Explanation: You have not included a file name in the ASMAHL command, and the FILEDEF for the *ddname* specified is not for DISK.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMAHL command and specify a file name, or reissue the FILEDEF for the ddname specified with a device type of 'DISK'.

ASMACMS038E Filename conflict for the SYSIN FILEDEF.

Explanation: The file name specified on the ASMAHL command conflicts with the file name on the FILEDEF for the SYSIN ddname.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the FILEDEF command or the ASMAHL command specifying the same file name.

ASMACMS040E Saved segment *xxxxxxxx* does not exist

Explanation: The specified saved segment has not been included in the System Names Table (SNT).

System Action: RC=40. Processing of the command terminates.

Programmer Response: See your system administrator.

ASMACMS041E The storage for saved segment *xxxxxxxx* is already in use

Explanation: The storage for the specified saved segment has already been used by another saved segment.

System Action: RC=40. Processing of the command terminates.

Programmer Response: See your system administrator.

ASMACMS042E SEGMENT error *nnn* loading saved segment *xxxxxxxx*

Explanation: An error occurred when the ASMAHL command attempted to load the specified saved segment.

System Action: RC=40. Processing of the command terminates.

Programmer Response: See your system administrator.

ASMACMS043E DIAGNOSE error *nnn* loading saved segment *xxxxxxxx*

Explanation: An error occurred when the ASMAHL command attempted to load the specified saved segment.

System Action: RC=40. Processing of the command terminates.

Programmer Response: See your system administrator.

ASMACMS044E NUCXLOAD error *nnn* loading *xxxxxxxx* module

Explanation: An error occurred when the ASMAHL command attempted to load the specified module.

System Action: RC=40. Processing of the command terminates.

Programmer Response: See your system administrator.

ASMACMS052E Option list exceeds 512 characters.

Explanation: The string of options that you specified with your ASMAHL command exceeded 512 characters in length.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your ASMAHL command with fewer options specified.

ASMACMS062E Invalid character *c* in file name *xxxxxxxx*

Explanation: A character that is not permitted was specified in the file name specified on the ASMAHL command.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the option with its correct parameters, and reissue the command with the correct parameter.

ASMACMS070E Left parenthesis '(' required before option list

Explanation: An option was specified after the file name but before the left parenthesis on the ASMAHL command.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Issue the ASMAHL command again with the option specified after the left parenthesis. Only the file name can be specified before the left parenthesis.

ASMACMS074E Required module *xxxxxxxx* MODULE not found

Explanation: The ASMAHL command was unable to load the specified module.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Verify you have accessed the disk containing the assembler and issue the ASMAHL command again.

ASMACMS075E Device *device* invalid for *xxxxxxxx*

Explanation: The device specified in your FILEDEF command cannot be used for the input or output operation that is requested in your program. For example, you have tried to read data from the printer or write data to the reader.

Supplemental Information: The variable device name in the text of the message indicates the incorrect device that was specified.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your FILEDEF command, specifying the correct device for the required input operation.

ASMACMS076E *xxxxxxxx* MODULE IS NOT IN RELEASE 2 FORMAT

Explanation: The module *xxxxxxxx* is not in the required format for Release 2.

Note: This message is only produced in uppercase English.

System Action: RC=40. Processing of the command terminates

Programmer Response: Ensure that you have the correct version of the module available. Check the disks you have linked, and make sure you are not accessing modules from an earlier release of High Level Assembler. If the module is ASMADOPT, you might need to reassemble your default options module with the ASMAOPT macro provided with High Level Assembler Release 2. If you cannot resolve the problem, contact your High Level Assembler maintenance programmer, or your IBM service representative.

Appendix H. User Interface Macros

The macros identified in this appendix are provided as programming interfaces by High Level Assembler.

Warning: Do not use as programming interfaces any High Level Assembler macros other than those identified in this appendix.

The following macros intended for customers are all General-Use Programming Interfaces.

ASMADATA Maps the records in the associated data file.

ASMAEFNP Maps the parameter list passed to external function routines for the SETAF and SETCF conditional assembler instructions.

ASMAXFMB Generates the Filter Management Table used by the sample ADATA user exit ASMAXADT.

ASMAXITP Maps the parameter list passed to the assembler user exits.

Appendix I. Sample ADATA User Exit

ASMAXADT is a sample ADATA exit supplied with High Level Assembler.

Function

The sample ADATA exit handles the details of interfaces to the assembler, and provides associated data (ADATA) records to any of a number of *filter modules* that inspect the records to extract the information they require. This allows filter modules to be added or modified without impacting either the exit or the other filter modules.

The design of the exit:

- Supports multiple simultaneous filter modules.
- Simplifies the ADATA record interface for each filter, because you don't need to know about the complex details of interacting directly with the assembler.
- Supports filter modules written in high level languages.

The three components that make up the functional ADATA exit are:

1. The exit routine, ASMAXADT, which is invoked by High Level Assembler
2. A table of filter module names, contained in a *Filter Management Table* (FMT) module ASMAXFMT. The FMT is loaded by the exit routine.
3. The filter modules. These are loaded by the exit as directed by the FMT. A sample filter module, ASMAXFLU, is provided with High Level Assembler.

Preparing the Exit

Before the exit can be used it must be assembled and link-edited, and the load module placed in a library in the standard search order. ASMAXADT, as supplied, has the following attributes:

```
REUSABLE, REENTERABLE, AMODE(24), RMODE(24)
```

Refer to Chapter 4, “Providing User Exits” on page 66 for further information about coding and preparing user exits.

Preparing the Filter Management Table

The names of the filter modules to be invoked by the user exit are contained in the Filter Management Table (FMT). The FMT is generated by using the macro ASMAXFMB. The names of the filter modules are specified as operands to the ASMAXFMB macro. Figure 77 shows an example of how to create an FMT that causes the filters `MYFILT`, `YOURFILT`, `HERFILT`, `HISFILT`, and `OURFILT` to be invoked by the exit.

```
ASMAXFMT Title 'ADATA Exit Filter Management Table'
         ASMAXFMB MYFILT,YOURFILT,HERFILT,HISFILT,OURFILT
         END
```

Figure 77. Creating a Filter Management Table

The object file produced from the assembly must be link-edited, and the load module placed in a library in the standard search order. ASMAXFMT, as supplied, has the following attributes:

```
REUSABLE, NON-REENTERABLE, NON-SHARABLE
```

You can specify an `initial character string`, as part of the `filter` operand, that is passed to the filter routine during initialization. Figure 78 shows two filter routines; `MYFILT`, that receives the characters `A,B,C`, and `ASMAXFLU`, that receives the characters `DUMP`.

```
ASMAXFMT Title 'ADATA Exit Filter Management Table'
         ASMAXFMB (MYFILT,'A,B,C'),(ASMAXFLU,'DUMP')
         END
```

Figure 78. Passing Initial Character String to Filter Routines

The default FMT control section (CSECT) name is ASMAXFMT. You can specify a different CSECT name using the `SECT` keyword on the ASMAXFMB macro. Figure 79 shows how to generate a CSECT name of `MYFMT`.

```
ASMAXFMT Title 'ADATA Exit Filter Management Table'
         ASMAXFMB SECT=MYFMT,(MYFILT,'A,B,C'),YOURFILT
         END
```

Figure 79. Generating an Alternative CSECT Name

Preparing the Filter Modules

The exit routine loads the Filter Management Table (FMT) module. The filter modules specified in the FMT are then loaded by the exit routine. Each filter module is called by the exit in three ways: once to process an OPEN request; multiple times to process ADATA records; and once to process a CLOSE request.

Call Interface: The filter modules must be placed in a library in the standard search order.

Each filter is called by the exit using the standard call interface in the following form:

```
CALL filter(exit_type,action,return_code,handle,record_length,record)
```

The exit branches to the filter module using the BASR assembler instruction.

Registers on Entry: Standard OS linkage conventions are used, and the registers on entry to the filter module are:

- R13 contains the address of a standard 18-word save area
- R14 contains the return address to the exit
- R15 contains the filter's entry point address
- R1 contains the address of a list of six fullwords that address:
  1. A fullword containing the `exit_type`
  2. A fullword integer containing the `action` code
  3. A fullword integer where the filter puts the `return_code`

4. A 4-fullword handle area
5. A fullword integer containing the ADATA record_length
6. The ADATA record

   The high-order bit of the last fullword address is set to one.

Figure 80 shows the six fullwords in the parameter list.

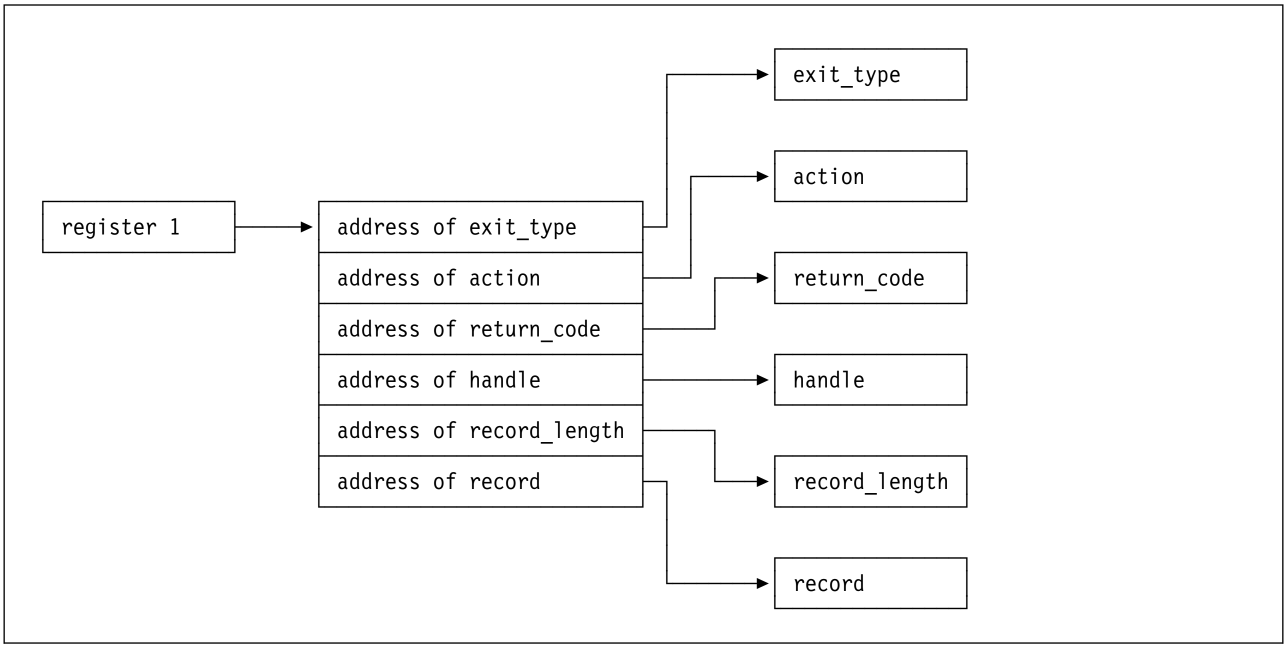

Figure 80. Filter Module Parameter List Format

Parameters on Entry: The six parameters are:

exit_type
: (Input) The address of a fullword of storage that indicates the exit type. The value is always 4, to indicate an ADATA exit.

action
: (Input only) The address of a fullword integer that can be one of the following three values:

0 OPEN Request. Open and initialize the filter. No ADATA record is available with this call.

The exit accepts the following return codes:

0 The open action was successful. The exit subsequently calls the filter module to inspect and process each ADATA record.

12 The open action was unsuccessful. The filter module is assumed to have closed itself, and is not called again.

1 CLOSE Request. The exit is requesting the filter module to close itself. No ADATA record is available with this call and no further calls are made to the filter module.

The exit accepts the following return codes:

0 The filter module has closed successfully. The exit can delete the filter.

12 The filter module is assumed to have closed itself, and is not called again. The exit can delete the filter.

3 PROCESS Request. A record is available to the filter module for processing. The ADATA record should not be modified.

The exit accepts the following return codes:

0 The filter module has completed its processing of this record, and is ready to accept further records.

12 The filter module is assumed to have closed itself, and is not called again.

return_code
(Output only) The address of a fullword integer where the filter module should place a return code. Valid return codes are described under each `action`.

handle
(Input/Output) The address of a 4-fullword area of storage that is initialized to zero before the OPEN (`action=0`) call to the filter. Its contents are preserved across subsequent calls. The `handle` can be used in any way by the filter module, for example, to address working storage for a reenterable filter module.

record_length
(Input only) The address of a fullword integer containing the length of the ADATA `record`. A length is provided for PROCESS (`action=3`) calls, and for OPEN (`action=0`) calls when you supply an `initial character string`.

record
(Input only) The address of the ADATA record. This points to the ADATA record for PROCESS (`action=3`) calls, and to the `initial character string` for OPEN (`action=0`) calls.

Information Messages: If all the filter modules request termination before the last ADATA record is processed the following message is issued and the assembly continues:

1 `ASMA700I All SYSADATA filter modules requested early termination`

Error Diagnostic Messages: When the Filter Management Table routine detects an error it directs the assembler to issue message `ASMA940U` and the assembly stops. The following messages might be issued:

1 `ASMA940U SYSADATA exit not coded at same level of interface definition (2) as assembler`

The exit uses version 2 of the exit definition, but the assembler uses a different version.

2 `ASMA940U SYSADATA exit called for other than SYSADATA`

The exit was invoked with a valid type, but the type is not one that the exit can process. This is probably caused by an incorrect ADEXIT() suboption of the EXIT assembler option.

3 `ASMA940U SYSADATA exit not initialized, and not entered for OPEN`

The exit has not yet been initialized, but was not entered with an OPEN request (`action=0`). There may be a failure in communication between the assembler and the exit.

4 `ASMA940U SYSADATA exit initialized, but was entered for OPEN`

The exit has been initialized, but was unexpectedly entered with an OPEN request (`action=0`). There may be a failure in communication between the assembler and the exit.

5 `ASMA940U SYSADATA exit - Invalid action or operation type requested`

An action was requested that is inconsistent with the type of action the exit is able or was expecting to take. There may be a failure in communication between the assembler and the exit.

6 `ASMA940U SYSADATA exit - Expecting input record, zero record length found`

The exit was expecting an input record, but the record length was zero. There may be a failure in communication between the assembler and the exit.

7 `ASMA940U Unable to load` *`xxxxxxxx`* `module. SYSADATA exit failed`

The assembler was unable to load the Filter Management Table module *xxxxxxxx*. No SYSADATA processing is possible.

8 `ASMA940U All SYSADATA filter modules failed to open`

All of the filter modules loaded by the exit failed to open. No SYSADATA processing is possible.

Preparing the Sample Filter Module ASMAXFLU

You can use the supplied filter routine, ASMAXFLU, to:

- Write the names of the primary input and library data sets to a data set.
- Dump the first 32 bytes of each ADATA record to a data set. This function is only performed if you specify `DUMP` as the initial character string. Figure 81 shows how to specify `DUMP` as the initial character string.

```
ASMAXFMT Title 'ADATA Exit Filter Management Table'
         ASMAXFMB (ASMAXFLU,'DUMP')
         END
```

Figure 81. Initial Character String for ASMAXFLU

Output from ASMAXFLU: The output from ASMAXFLU is written to a data set defined by the ddname XFLUOUT. The data set record length is 80 bytes. The first record in the data set is a header record, and the last record in the data set is a trailer record. The dump, header, and trailer records are prefixed with an asterisk.

The data set records have the following format:

Columns **Contents**

1 Record type: 'P'=Primary Input, 'L'=Library

2 Blank

3-10 Date, in *YYYYMMDD* format (blank for type 'L')

11-14 Time, in *HHMM* format (blank for type 'L')

15-58 Data set name

59-66 Member name

67-72 Volume ID where file was found

73-80 Sequencing information

Figure 82 shows a sample data set containing records written by ASMAXFLU:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* ASMAXFLU Filter Header Record
*Dump 10000202 00000000 00000000 00000000 00000000 00000000 00000000 00000000
*Dump 10000102 00000000 AA3261D2 C4482402 00250000 00000000 00000000 00000000
*Dump 10000002 00000000 F1F9F9F4 F1F1F1F8 F1F8F5F0 F5F6F9F6 60F2F3F4 F14BF24B
P 199411181850TEST     ASSEMBLE A1                                MY-Z-D00000001
*Dump 10001002 00000000 A0CF0049 84BC2600 0F000000 40404040 40404040 40400000
*Dump 10002002 00000000 00000000 00000001 00000000 0000001C 00000000 00000000
*Dump 10003002 00000000 00000001 00000000 00000001 00000000 00010000 00000000
*Dump 10003602 00000000 00000001 00000000 00000000 00000000 00044510 C0120000
*Dump 10003002 00000000 00000009 00000000 000006BC 00000006 00030001 00000004
*Dump 10003402 00000000 00000001 00018000 00000000 00000009 00000004 00000001
*Dump 10004202 00000000 00000001 00000001 01D10000 00010001 00000000 00000000
*Dump 10004402 00000000 00040000 00010001 40000000 00000000 E3C5E2E3 E4000000
*Dump 10006002 00000000 00010014 00060001 00060000 0000D4E8 D4C1C340 404040D4
L             MYMAC    MACLIB   A1                        AMAC    MY-Z-D00000002
*Dump 10006002 00000000 00010014 00060002 00060000 0000D6E2 D4C1C3D9 D64040D4
L             OSMACRO  MACLIB   S2                        WTO     ESA19000000003
*Dump 10008002 00000000 00000000 00000002 00000000 00000000 00000008 00000001
*Dump 10009002 00000000 00001082 000000C8 00000009 000006C0 00000000 00000076
*Dump 10000202 00000000 00010000 00000029 00000000 00000000 00000000 00000000
* ASMAXFLU Filter Trailer Record
```

Figure 82. Sample Output Data Set from ASMAXFLU

Error Messages: When ASMAXFLU detects an error it writes an error message to the XFLUOUT data set. The following messages might be written:

- `ASMAXFLU called with unknown ADATA Definition Level.`

  Check the value of ADATA_LEVEL in the ADATA record header.

- `ASMAXFLU called for other than Assembler ADATA?`

  Check the value of ADATA_VERSION in the ADATA record header.

- `ASMAXFLU library record has no member names?`

  Check the value of ADMXREF_NUM in the X'0060' ADATA record.

- `ASMAXFLU library record missing member data?`

  Check the value of the member name length in the X'0060' ADATA record.

- `ASMAXFLU Job-ID record has no file names?`

  Check the value of ADJID_NUM_INPUT_FILES in the X'0000' ADATA record.

- `ASMAXFLU called with unrecognized action code.`

  The `action` code is not 0, 1, or 3.

- `ASMAXFLU called with unrecognized exit type.`

  The `exit_type` is not 4.

Assembling and Link-Editing ASMAXFLU: You must assemble and link-edit ASMAXFLU, placing the load module in a library in the standard search order. ASMAXFLU, as supplied, has the following attributes:

```
NON-REUSABLE,NON-REENTERABLE, AMODE(24), RMODE(24)
```

See page 316 for details about preparing filter modules.

Invoking the Exit

To invoke the exit, specify the EXIT assembler option as follows:

```
EXIT(ADEXIT(ASMAXADT))
```

If you don't want to use the default filter management table ASMAXFMT, you can specify a different name as follows:

```
EXIT(ADEXIT(ASMAXADT(fmt_name))
```

where *fmt_name* is the load module name of the filter management table. See Figure 79 on page 316, which shows you how to generate an alternative filter management table.

Appendix J. Sample LISTING User Exit

ASMAXPRT is a sample LISTING exit supplied with High Level Assembler.

Function

The sample LISTING exit suppresses printing of the *High Level Assembler Options Summary*, or the *Diagnostic Cross Reference and Assembler Summary*, or both. It can also print the *Options Summary* page at the end of the listing, instead of its normal position at the beginning of the listing.

Preparing the Exit

Before the exit can be used it must be assembled and link-edited, and the load module (phase) placed in a library in the standard search order. ASMAXPRT, as supplied, has the following attributes:

```
REUSABLE, REENTERABLE, AMODE(31), RMODE(ANY)
```

Refer to Chapter 4, “Providing User Exits” on page 66 for further information about coding and preparing user exits.

Invoking the Exit

To invoke the exit specify the EXIT assembler option as follows:

```
EXIT(PRTEXIT(ASMAXPRT(parameter-string)))
```

where *parameter-string* controls what action the exit performs.

Parameter String

```
        ┌─,──────────┐
►►──(───▼─┬─NOOPTION──┬─┴──)──►◄
          ├─NOSUMMARY─┤
          └─OPTEND────┘
```

Default

None. At least one keyword is required.

Abbreviations

NOOP, NOSUM

The abbreviations shown here are the minimum number of characters allowed. You can, for example, specify NOOPTI or NOSUMM.

NOOPTION

Suppress the *Options Summary*

NOSUMMARY

Suppress the *Diagnostic Cross Reference and Assembler Summary*

OPTEND

Print the *Options Summary* at the end of the assembler listing, instead of at the beginning.

Messages

ASMAXPRT might issue message `ASMA701` as follows:

1. ** WARNING ** LISTING: ASMAXPRT - Invalid Option Specified: *xxxxxxxx*

   This message is issued because the value *xxxxxxxx* specified as an exit string of the EXIT assembler option is not recognized by ASMAXPRT.

   The exit uses the keyword options processed until the error was detected. Any values in the exit string after *xxxxxxxx* are ignored.

2. ** WARNING ** LISTING: ASMAXPRT - No options specified

   This message is issued because ASMAXPRT expects one or more keyword options in the exit string of the EXIT assembler option.

3. ** WARNING ** LISTING: ASMAXPRT - Exit buffer is full

   This message is issued because ASMAXPRT, as supplied, only supports a maximum of 60 lines for the *Options Summary* page. To increase this value or change it to allow an unlimited number of lines, modify the exit source, assemble it and link-edit it.

   This error might cause an incomplete *Options Summary* page.

Appendix K. Sample SOURCE User Exit

ASMAXINV is a sample SOURCE exit supplied with High Level Assembler.

Function

The sample SOURCE exit reads variable-length source data sets. Each record that is read is passed to the assembler as an 80-byte source statement. If any record in the input data set is longer than 71 characters the remaining part of the record is converted into continuation records.

The exit also reads a data set with a fixed record length of 80 bytes.

Preparing the Exit

Before the exit can be used it must be assembled and link-edited, and the load module (phase) placed in a library in the standard search order. ASMAXINV, as supplied, has the following attributes:

```
REUSABLE, REENTERABLE, AMODE(24), RMODE(24)
```

Refer to Chapter 4, “Providing User Exits” on page 66 for further information about coding and preparing user exits.

Invoking the Exit

To invoke the exit specify the EXIT assembler option as follows:

```
EXIT(INEXIT(ASMAXINV))
```


Appendix L. How to Generate a Translation Table

High Level Assembler uses the EBCDIC character set to represent characters contained in character (C-type) data constants (DCs) and literals. The TRANSLATE assembler option lets you specify a module containing a translation table which the assembler uses to convert these characters into another character set.

High Level Assembler provides an ASCII translation table, however, you can supply your own translation table. The translation table module must be named ASMALT*xx*, where *xx* is the suffix specified in the TRANSLATE assembler option. See “TRANSLATE” on page 61.

Preparing the Translation Table: The user-supplied translation table must be assembled and link-edited into a library in the standard load module search order. The full name of the translation table load module name must occupy bytes 257 to 264 of the module. The first byte of the module must be the first byte of the translation table.

A sample translation table to convert a subset of EBCDIC characters into ASCII characters is shown in Figure 83 on page 326. Specify the TRANSLATE(U1) assembler option to use this translation table.


```
&LT      SETC  'ASMALTU1'
&LT      CSECT
         DC    256X'00'
         ORG   &LT+64
         DC    X'20'             EBCDIC: X'40' blank
         ORG   &LT+75
         DC    X'2E3C282B'       EBCDIC: .<(+
         ORG   &LT+80
         DC    X'26'             EBCDIC: &
         ORG   &LT+90
         DC    X'21242A293B'     EBCDIC: !$*);
         ORG   &LT+96
         DC    X'2D2F'           EBCDIC: -/
         ORG   &LT+106
         DC    X'7C2C255F3E3F'   EBCDIC: ],%_>?
         ORG   &LT+121
         DC    X'603A23402C3D'   EBCDIC:  :#@'=
         ORG   &LT+127
         DC    X'22'             EBCDIC: "
         ORG   &LT+129
         DC    X'616263646566'   EBCDIC: abcdef
         ORG   &LT+135
         DC    X'676869'         EBCDIC: ghi
         ORG   &LT+145
         DC    X'6A6B6C6D6E6F'   EBCDIC: jklmno
         ORG   &LT+151
         DC    X'707172'         EBCDIC: pqr
         ORG   &LT+161
         DC    X'7E7374757677'   EBCDIC: ~stuvw
         ORG   &LT+167
         DC    X'78797A'         EBCDIC: xyz
         ORG   &LT+192
         DC    X'7B41424344'     EBCDIC: {ABCD
         ORG   &LT+197
         DC    X'4546474849'     EBCDIC: EFGHI
         ORG   &LT+208
         DC    X'7D4A4B4C4D'     EBCDIC: }JKLM
         ORG   &LT+213
         DC    X'4E4F505152'     EBCDIC: NOPQR
         ORG   &LT+224
         DC    X'5C'             EBCDIC: \
         ORG   &LT+226
         DC    X'53545556'       EBCDIC: STUV
         ORG   &LT+230
         DC    X'5758595A'       EBCDIC: WXYZ
         ORG   &LT+240
         DC    X'3031323334'     EBCDIC: 01234
         ORG   &LT+245
         DC    X'3536373839'     EBCDIC: 56789
         ORG   &LT+256
         DC    CL8'&LT'          Table name = Module name
         END
```

Figure 83. Sample translation table

Glossary

This glossary defines term that are used in the High Level Assembler publications. Some of these terms might not be used in this publication.

This glossary has three main types of definitions that apply:

- To the assembler language in particular (usually distinguished by reference to the words "assembler," "assembly," etc.)
- To programming in general
- To data processing as a whole

If you do not understand the meaning of a data processing term used in any of the definitions below, refer to *Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing, which was prepared by Subcommittee X3K5 on Terminology and Glossary of American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

absolute expression. An expression is absolute if its value does not change upon program relocation.

absolute value. Is the value of a term when that value does not change upon program relocation.

addressing mode (24-bit). A System/370 addressing mode (AMODE) of the extended architecture that allows a program to run using 24-bit addresses. When operating in 24-bit mode, S/370* addressing architecture is applied. Other facilities of the extended architecture (see below) may be utilized. Only the low-order 24 bits of an address are used; the high-order bits are ignored.

addressing mode (31-bit). An extended architecture addressing mode (AMODE) that allows a program to run using 31-bit addresses, other facilities of the extended architecture, or both. When operating in 31-bit mode, extended architecture addressing is applied, and all but the high-order bit of an address are used to address storage.

assemble. To prepare a machine language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

***assembler**. A computer program that assembles.

assembler instruction. An assembler language source statement that causes the assembler to do a specific operation. Assembler instructions are not translated into machine instructions.

assembler language. A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer. The assembler language also contains statements that represent assembler instructions and macro instructions.

automatic library call. The process by which the linkage editor or binder resolves external references by including additional members from the automatic call library.

bimodal program execution. A function of the extended architecture (see "addressing mode (31-bit)") that allows a program to run in 24-bit or 31-bit addressing mode. The addressing mode is under program control.

binder. The component of DFSMS/MVS which is responsible for linking and editing programs, to create either record format load modules or program objects. The DFSMS/MVS binder is a functional replacement for the MVS/DFP* linkage editor.

bracketed DBCS. DBCS characters enclosed with a shift-out (SO) character and a shift-in character (SI) to identify them from SBCS, and containing no SBCS characters except SO and SI.

conditional assembly language. A programming language that the assembler processes during conditional assembly. The conditional assembly language can be used to perform general arithmetic and logical computations, generate machine and assembler instructions from model statements, and provide variable symbols to represent data and vary the content of model statements during generation. It can be used in macro definitions, and in open code.

control program. A program that is designed to schedule and supervise the performance of data processing work by a computing system; an operating system.

control section (CSECT). That part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

data attributes. Values assigned by the assembler which describe the characteristics of ordinary symbols and variable symbols that represent data.

***diagnostic**. Pertaining to the detection and isolation of a malfunction or mistake.

double-byte character set (DBCS). DBCS is a means of providing support for Ideographic Languages which contain too many symbols to be represented by a single byte character set such as EBCDIC. A valid double-byte character is defined as either DBCS blank (X'4040'), or a pair of bytes, each of which must be in the range X'41' to X'FE', inclusive.

double-byte data. Double-byte character strings are commonly referred to as double-byte data.

dummy control section (DSECT). A control section that an assembler can use to map an area of storage without producing any object code or data for that area. Synonymous with dummy section.

edited text. Source statements modified by the assembler for internal use. The initial processing of the assembler is referred to as editing.

enterprise systems architecture. A hardware architecture for the IBM 3090* processor. A major characteristic is 31-bit addressing. See also "addressing mode (31-bit)."

***entry point**. A location in a module to which control can be passed from another module or from the control program.

extended architecture. A hardware architecture for the IBM 3081 processor. A major characteristic is 31-bit addressing. See also "addressing mode (31-bit)."

external symbol dictionary (ESD). Control information associated with an object or load module which identifies the external symbols in the module.

global dictionary. An internal table used by the assembler during macro generation to contain the current values of all unique global SETA, SETB, and SETC variables from all text segments.

global vector table. A table of pointers in the skeleton dictionary of each text segment showing where the global variables are located in the global dictionary.

hierarchical file system. In MVS/ESA OpenEdition, a Hierarchical File System (HFS) is a collection of files organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the root directory. MVS views an entire file hierarchy as a collection of hierarchical file system data sets (HFS data sets). Each HFS data set is a mountable file system. The Hierarchical File System is described in the *MVS/ESA OpenEdition MVS User's Guide*, SC23-3013.

instruction. *(1) A statement that specifies an operation and the values and locations of its operands. (2) See also "assembler instruction," "machine instruction," and "macro instruction."

job control language (JCL). A language used to code job control statements.

***job control statement**. A statement in a job that is used in identifying the job or describing its requirements to the operating system.

language. A set of representations, conventions, and rules used to convey information.

***language translator**. A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

library macro definition. A macro definition that is stored in a macro library. The IBM-supplied supervisor and data management macro definitions are examples of library macro definitions.

linkage editor. A processing program that prepares the output of language translators to enable it to run. It combines separately produced object or load modules; resolves symbolic cross references among them; replaces, deletes, and adds control sections; generates overlay structures on request; and produces executable code (a load module) that is ready to be fetched into main storage and run.

linker. Used in this publication as collective term for *binder* and *linkage editor.*

load module. The output of a single linkage editor run. A load module is in a format suitable for loading into virtual storage and running.

loader. A processing program that does the basic editing functions of the linkage editor, and also fetches and gives control to the processed program. It accepts object modules and load modules created by the linkage editor and generates executable code directly in storage. The loader does not produce load modules for program libraries.

local dictionary. An internal table used by the assembler during macro generation to contain the current values of all local SET symbols. There is one local dictionary for open code, and one for each macro definition.

location counter. A counter whose value indicates the assembled address of a machine instruction or a constant or the address of an area of reserved storage, relative to the beginning of the control section.

***machine instruction**. An instruction that a machine can recognize and execute.

***machine language**. A language that is used directly by the machine.

macro definition. A set of statements that defines the name of, format of, and conditions for generating a sequence of assembler language statements from a single source statement. This statement is a macro instruction that calls the definition. (See also "library macro definition" and "source macro definition.")

macro generation (macro expansion). An operation in which the assembler generates a sequence of assembler language statements from a single macro instruction, under conditions described by a macro definition.

macro instruction (macro call). An assembler language statement that causes the assembler to process a predefined set of statements (called a macro definition). The statements normally produced from the macro definition replace the macro instruction in the source program.

macro library. A library containing macro definitions. The supervisor and data management macro definitions supplied by IBM (GET, LINK, etc.) are contained in the system macro library. Private macro libraries can be concatenated with the system macro library.

macro prototype statement. An assembler language statement that specifies the mnemonic operation code and the format of all macro instructions that are used to call a macro definition.

MACRO statement. An assembler language statement that indicates the beginning of a macro definition. (Also known as a macro definition header).

main storage. All program addressable storage from which instructions may be executed and from which data can be loaded directly into registers.

MEND statement. An assembler language statement that indicates the end of a macro definition. (Also known as a macro definition trailer).

model statement. A statement from which assembler language statements are generated during conditional assembly.

object module. The machine-language output of a single run of an assembler or a compiler. An object module is used as input to the linkage editor, loader, or binder.

open code. The portion of a source module that lies outside of and after any source macro definitions that may be specified.

***operating system**. Software that controls the running of computer programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services. (see "control program.")

ordinary symbol attribute reference dictionary. A dictionary used by the assembler. The assembler puts an entry in it for each ordinary symbol encountered in the name field of a statement. The entry contains the attributes (type, length, etc.) of the symbol.

partitioned data set (PDS). A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

PDSE (partitioned data set extended). A system-managed data set that contains an indexed directory and members that are similar to the directory and members of partitioned data sets.

processing program. (1) A general term for any program that is not a control program. (2) Any program capable of operating in the problem program state. This includes IBM-distributed language translators, application programs, service programs, and user-written programs.

program. A general term for any combination of statements that can be interpreted by a computer or language translator, and that serves to do a specific function.

program fetch. A program that prepares programs for execution by loading them at specific storage locations and readjusting each (relocatable) address constant.

program library. A partitioned data set or PDSE that always contains named members.

program management binder. See *binder*.

program module. Used in this publication as collective term for *load module* and *program object*.

program object. All or part of a computer program in a form suitable for loading into main storage for execution. Program objects are stored in PDSE program libraries, and are produced by the Program Management Binder.

pure DBCS. DBCS characters not delimited by SO and SI. These characters must be known to be DBCS by some other method, such as the position in a record, or a field type descriptor in a database environment.

real storage. The storage of a System/370 computer from which the central processing unit can directly obtain instructions and data, and to which it can directly return results.

read-only control section (RSECT). That part of a program specified by the programmer to be a read-only executable control section. The assembler automatically checks the control section for possible coding violations of program reenterability, regardless of the setting of the RENT assembler option.

reenterable. An attribute that allows a program to be used concurrently by more than one task. This attribute is sometimes called *reentrant*.

refreshable. An attribute that allows a program to be replaced with a new copy without affecting its operation.

reusability. An attribute of a program that defines the scope to which it can be reused or shared by multiple tasks within an address space.

relocatable expression. An expression is relocatable if its value changes because the control section in which it appears is relocated.

relocatable value. Is the value of a term when that value changes because the control section in which it appears is relocated.

***relocation dictionary**. The part of an object or load module that identifies all addresses that must be adjusted when a relocation occurs.

residence mode. An extended architecture addressing mode (RMODE) that allows a program to specify the residence mode (below 16 megabytes or anywhere) to be associated with a control section.

return code. A value placed in the return code register at the completion of a program. The value is established by the user and may be used to influence the running of succeeding programs or, in the case of an abnormal end of task, may simply be printed for programmer analysis.

severity code. A code assigned by the assembler to each error detected in the source code. The highest code encountered during assembly becomes the return code of the assembly step.

shift-in (SI). The shift-in (SI) EBCDIC character (`X'0F'`) delimits the end of double-byte data.

shift-out (SO). The shift-out (SO) EBCDIC character (`X'0E'`) delimits the start of double-byte data.

skeleton dictionary. A dictionary built by the assembler for each text segment. It contains the global vector table, the sequence symbol reference dictionary, and the local dictionary.

source macro definition. A macro definition included in a source module, either physically or as the result of a COPY instruction.

source module. The source statements that constitute the input to a language translator for a particular translation.

source statement. A statement written in a programming language.

***statement**. A meaningful expression or generalized instruction in a programming language.

symbol file. A data set used by the assembler for symbol definitions and references and literals.

symbolic parameter. In assembler programming, a variable symbol declared in the prototype statement of a macro definition.

system macro definition. Loosely, an IBM-supplied library macro definition which provides access to operating system facilities.

text segment. The range over which a local dictionary has meaning. The source module is divided into text segments with a segment for open code and one for each macro definition.

***translate**. To transform statements from one language into another without significantly changing the meaning.

virtual storage. Address space appearing to the user as real storage from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

ward. A set of DBCS characters which have the same high-order byte value. The first byte of a double-byte character is known as the ward byte. A ward contains 190 characters. Ward `X'42'` defines the double-byte representation of those EBCDIC characters which are in the range `X'41'` to `X'FE'`.

Bibliography

High Level Assembler Publications

General Information, GC26-4943

Installation and Customization Guide, SC26-3494

Language Reference, SC26-4940

Licensed Program Specifications, GC26-4944

Programmer's Guide, SC26-4941

High Level Assembler Publications for VSE

IBM High Level Assembler for MVS & VM & VSE General Information, GC26-8261

IBM High Level Assembler for MVS & VM & VSE Installation and Customization Guide, SC26-8263

IBM High Level Assembler for MVS & VM & VSE Language Reference, SC26-8265

IBM High Level Assembler for MVS & VM & VSE Licensed Program Specifications, GC26-8262

IBM High Level Assembler for MVS & VM & VSE Programmer's Guide, SC26-8264

Related Publications (Architecture)

Enterprise Systems Architecture/390 Principles of Operation,SA22-7205

IBM Enterprise Systems Architecture/370 and System/370 Vector Operations, SA22-7125

IBM System/370 Enterprise Systems Architecture Principles of Operation, SA22-7200

IBM System/370 Extended Architecture Principles of Operation, SA22-7085

IBM System/370 Principles of Operation, GA22-7000

Related Publications (MVS)

MVS/ESA Version 4:

MVS/ESA JCL User's Guide, GC28-1653

MVS/ESA Application Development Reference: Services for Assembler Language Programs, GC28-1642

MVS/ESA JCL Reference, GC28-1654

MVS/ESA System Codes, GC28-1664

MVS/ESA System Messages Volumes 1 - 5, GC28-1656, GC28-1657, GC28-1658, GC28-1659, GC28-1660

MVS/ESA Version 5:

MVS/ESA JCL Reference, GC28-1479

MVS/ESA JCL User's Guide, GC28-1473

MVS/ESA Programming: Assembler Services Guide, GC28-1466

MVS/ESA Programming: Assembler Services Reference, GC28-1474

MVS/ESA Programming: Authorized Assembler Services Guide, GC28-1467

MVS/ESA Programming: Authorized Assembler Services Reference Volumes 1 - 4, GC28-1475, GC28-1476, GC28-1477, GC28-1478

MVS/ESA System Codes, GC28-1486

MVS/ESA System Commands, GC28-1442

MVS/ESA System Messages Volumes 1 - 5 , GC28-1480, GC28-1481, GC28-1482, GC28-1483, GC28-1484

MVS/ESA OpenEdition:

MVS/ESA OpenEdition MVS User's Guide, SC23-3013

MVS/DFP:

MVS/DFP Version 3.3: Utilities, SC26-4559

MVS/DFP Version 3.3: Linkage Editor and Loader, SC26-4564

DFSMS/MVS:

DFSMS/MVS Program Management, SC26-4916

TSO/E:

TSO/E Command Reference, SC28-1881

Related Publications (VM)

VM/ESA CMS Application Development Guide, SC24-5450

VM/ESA CMS Application Development Guide for Assembler, SC24-5452

VM/ESA CMS Application Development Reference, SC24-5451

VM/ESA CMS Application Development Reference for Assembler, SC24-5453

VM/ESA CMS Command Reference, SC24-5461

VM/ESA CMS User's Guide, SC24-5460

VM/ESA CP Command and Utility Reference, SC24-5519

VM/ESA XEDIT Command and Macro Reference, SC24-5464

VM/ESA XEDIT User's Guide, SC24-5463

Index

Special Characters

Numerics

A

B

C

D

E

We'd Like to Hear from You

High Level Assembler for MVS & VM & VSE
Programmer's Guide
MVS & VM Edition
Release 2

Publication No. SC26-4941-01

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - IBMMail: USIB2VVG at IBMMAIL
  - IBMLink: HLASMPUB at STLVM27
  - Internet: COMMENTS@VNET.IBM.COM

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

High Level Assembler for MVS & VM & VSE
Programmer's Guide
MVS & VM Edition
Release 2

Publication No. SC26-4941-01

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

Company or Organization

Phone No.

Readers' Comments
SC26-4941-01

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370-40
Program Number: 5696-234

Printed in U.S.A.

High Level Assembler for MVS & VM Publications

GC26-4943	General Information
GC26-4944	Licensed Program Specifications
SC26-4940	Language Reference
SC26-4941	Programming Guide
SC26-3494	Installation and Customization

SC26-4941-01

Spine information:

IBM HLASM **Programmer's Guide (MVS & VM)** Release 2 SC26-4941-01