High Level Assembler for z/OS & z/VM & z/VSE

Programmer's Guide

Version 1 Release 6

SC26-4941-06

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 397.

This edition applies to IBM High Level Assembler for z/OS & z/VM & z/VSE, Release 6, Program Number 5696-234 and to any subsequent releases until otherwise indicated in new editions. Make sure that 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.

IBM welcomes your comments. For information on how to send comments, see “How to send your comments to IBM” on page xix.

© Copyright IBM Corporation 1992, 2013.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xi

About this document xiii

Who should use this document xiii
Programming interface information xiii
Organization of this document. xiii
High Level Assembler documents xv
  Documents xv
  Collection kits xvi
Related publications xvi
Syntax notation xvi

How to send your comments to IBM xix

If you have a technical problem xix

Summary of changes xxi

Chapter 1. Introduction 1

Requirements 1
  System requirements. 1
  Machine requirements 1
  Storage requirements 2
Compatibility 2
  Assembler language support 2
  Migration considerations 2

Chapter 2. Using the assembler listing . 5

High Level Assembler option summary 6
External Symbol Dictionary (ESD) 9
Source and object 12
Relocation dictionary (RLD) 18
Ordinary symbol and literal cross reference. . . . 19
Unreferenced symbols defined in CSECTs 22
Macro and copy code source summary 22
Macro and copy code cross reference 23
  Effects of LIBMAC and PCONTROL(MCALL) options 24
DSECT cross reference. 26
USING map 27
General Purpose Register cross reference 28
Diagnostic cross reference and assembler summary 29
Terminal output 32

Chapter 3. Controlling your assembly with options 35

The sources of assembler options 35
  Precedence of assembler options 35
  Fixed installation default options 36
  *PROCESS OVERRIDE Statement Options . . . 36
  ASMAOPT options 36
  Invocation options 36
  *PROCESS statement options 37
  Default options 37
  Invoking the assembler dynamically 37
  Coding rules 37
Assembler options 38
  ADATA. 39
  ALIGN 39
  ASA (z/OS and CMS) 40
  BATCH. 40
  CODEPAGE 41
  COMPAT 42
  DBCS 43
  DECK 43
  DISK (CMS) 44
  DXREF 44
  ERASE (CMS) 45
  ESD 45
  EXIT. 46
  FLAG 48
  FOLD 51
  GOFF (z/OS and CMS) 51
  INFO 52
  LANGUAGE 53
  LIBMAC 54
  LINECOUNT. 54
  LIST 55
  MACHINE 56
  MXREF. 58
  OBJECT 59
  OPTABLE 59
  PCONTROL 61
  PESTOP 63
  PRINT (CMS). 63
  PROFILE 63
  RA2 64
  RENT 65
  RLD 65
  RXREF 66
  SECTALGN 66
  SEG (CMS) 66
  SIZE. 67
  SUPRWARN 69
  SYSPARM 69
  TERM 70
  TEST 71
  THREAD 71
  TRANSLATE 72
  TYPECHECK. 72
  USING 73
  WORKFILE 75
  XOBJECT (z/OS and CMS) 76
  XREF 76

Chapter 4. Providing user exits 79

Exit types 79

© Copyright IBM Corp. 1992, 2013

Specifying user exits 80
Loading user exits 81
Calling user exits 81
Exit parameter list 82
  Request info pointer 83
  Buffer pointer 90
  Error buffer pointer. 90
  Exit-specific information pointer 91
  DCB pointer 91
  Static assembler information pointer 91
  HLASM Services Interface pointer 92
Error handling 92
Exit-Specific Information Block 92
SOURCE exit processing 95
  OPEN 95
  CLOSE 96
  READ 96
  PROCESS 97
LIBRARY exit processing 98
  OPEN 98
  CLOSE 99
  READ 99
  PROCESS MACRO or PROCESS COPY 99
  FIND MACRO or FIND COPY 100
  END OF MEMBER 102
LISTING exit processing. 103
  OPEN 104
  CLOSE 105
  WRITE 105
  PROCESS 105
OBJECT (z/OS and CMS) and PUNCH exit processing 107
  OPEN 107
  CLOSE 108
  WRITE 108
  PROCESS 108
ADATA exit processing 110
  OPEN 110
  CLOSE 111
  WRITE 111
  PROCESS. 111
TERM exit processing 112
  OPEN 112
  CLOSE 113
  WRITE 113
  PROCESS. 113
Sample user exits 114
User exit coding example 114
  OBJECT and PUNCH exit—OBJEXIT 115
  ADATA Exit—ADEXIT 115
  TERM exit—TRMEXIT 116

Chapter 5. Providing external functions 135
External function processing 135
Linkage conventions 136
External function parameter list 136
  Request information list 138
  Pointer to user work area 140
  Pointer to static assembler information 140
  Pointer to msg buffer. 141
  Pointer to return string (SETCF) 141
  Pointer to parm string *n* (SETCF). 141

Chapter 6. Diagnosing assembly errors. 143
Assembly error diagnostic messages. 143
MNOTE statements 144
Suppression of error messages and MNOTE statements 146
Reference information for statements in error. . . 146
Abnormal assembly termination 146
MHELP - macro trace facility 146

Chapter 7. Assembling your program on z/OS 149
Input to the assembler 149
Output from the assembler 149
Invoking the assembler on z/OS 149
Invoking the assembler on TSO 151
Invoking the assembler dynamically. 152
Batch assembling 154
Input and output data sets 154
  Specifying the source data set: SYSIN 157
  Specifying the option file: ASMAOPT 157
  Specifying macro and copy code libraries: SYSLIB 158
  Specifying the listing data set: SYSPRINT . . . 158
  Directing assembler messages to your terminal: SYSTERM 158
  Specifying object code data sets: SYSLIN and SYSPUNCH 158
  Specifying the associated data data set: SYSADATA 159
  Specifying the utility data data set: SYSUT1 . . 159
Return codes 159

Chapter 8. Linking and running your program on z/OS 161
The program management binder 161
The loader 163
Creating a load module 163
  Creating a load module on z/OS 163
  Creating a load module on TSO 163
Input to the binder 164
  Data sets for binder processing 165
  Additional object modules as input 165
Output from the binder 166
  Binder processing options 166
  Specifying binder options through JCL 167
  Specifying binder options using the TSO LINK command 168
  AMODE and RMODE attributes 168
  Overriding the defaults 169
  Detecting binder errors 169
Running your assembled program 169
  Running your assembled program in batch . . 169
  Running your assembled program on TSO . . 169

Chapter 9. z/OS system services and programming considerations 171
Adding definitions to a macro library 171
Using cataloged procedures 172
  Cataloged procedure for assembly (ASMAC) 172
  Cataloged procedure for assembly and link (ASMACL) 173
  Cataloged procedure for assembly, link, and run (ASMACLG) 175
  Cataloged procedure for assembly and run (ASMACG) 176
  Overriding statements in cataloged procedures 178
  Examples of cataloged procedures 178
Operating system programming conventions . . . 180
  Saving and restoring general register contents 180
Ending program execution 180
Accessing execution parameters 181
Object program linkage 181
Modifying program modules 182

Chapter 10. Assembling your program on CMS 183
Input to the assembler 183
Output from the assembler 183
Accessing the assembler 183
Invoking the assembler on CMS 184
Batch assembling 184
Controlling your assembly 185
Input and output files 186
  Specifying the source file: SYSIN 188
  Specifying the option file: ASMAOPT 189
  Specifying macro and copy code libraries: SYSLIB 189
  Specifying the listing file: SYSPRINT 190
  Directing assembler messages to your terminal: SYSTERM 190
  Specifying object code files: SYSLIN and SYSPUNCH 190
  Specifying the associated data file: SYSADATA 190
  Specifying the utility data file: SYSUT1 191
Return codes 191
Diagnostic messages written by CMS 191

Chapter 11. Running your program on CMS 193
Using the CMS LOAD and START commands . . 193
Using the CMS GENMOD command 193
Using the CMS LKED and OSRUN commands . . 194
Using the CMS batch facility 194

Chapter 12. CMS system services and programming considerations 195
Using macros 195
  Assembler macros supported by CMS 195
  Adding definitions to a macro library 195
Operating system programming conventions . . . 195
  Saving and restoring general register contents 195
Ending program execution 196
Passing parameters to your assembler language program 196

Chapter 13. Assembling your program on z/VSE 199
Input to the assembler 199
Output from the assembler 199
Invoking the assembler in batch 199
Invoking the assembler on ICCF 200
Invoking the assembler dynamically 203
Batch assembling 203
Controlling your assembly 204
Input and output files 205
  Specifying the source file: SYSIPT 207
  Specifying macro and copy code libraries: LIBDEF job control statement 207
  Specifying the listing file: SYSLST 207
  Directing assembler messages to your console log: SYSLOG 207
  Specifying object code files: SYSLNK and SYSPCH 207
  Specifying the associated data file: SYSADAT 208
  Specifying the utility data file: SYSUT1 208
  Specifying the option file: ASMAOPT 208
Return codes 208

Chapter 14. Link-editing and running your program on z/VSE 209
The linkage editor 209
Creating a phase 209
Input to the linkage editor 209
  Inputting object modules 210
  Files for linkage editor processing 210
  Inputting additional object modules 210
  Linkage editor control statements 211
Output from the linkage editor 211
Running your assembled program 212

Chapter 15. z/VSE system services and programming considerations. . . 213
Adding definitions to a macro library 213
Processing E-decks 213
Operating system programming conventions . . . 213
  Saving and restoring general register contents 214
  Ending program execution 214
  Accessing execution parameters 215

Appendix A. Cross-system portability considerations 217
Using machine instructions 217
Using system macros 217
Migrating object programs 217

Appendix B. Object deck output . . . 219
ESD record format 219
TXT record format 221
RLD record format 221
END record format 222
SYM record format 223

Appendix C. Associated data file output 227
Record types 234
  Macro-only assemblies 237
ADATA record layouts 239
Common header section. 239
Job Identification Record—X'0000' 240
ADATA Identification Record—X'0001' 242
ADATA Compilation Unit Start/End Record—X'0002' 242
Output File Information Record—X'000A' 243
Options File Information—X'000B' 247
Options record—X'0010' 247
External Symbol Dictionary Record—X'0020' . . . 256
Source Analysis Record—X'0030' 257
Source Error Record—X'0032' 260
DC/DS record—X'0034' 261
DC Extension Record—X'0035'. 263
Machine Instruction Record—X'0036' 263
Relocation Dictionary Record—X'0040' 264
Symbol Record—X'0042'. 264
Symbol and Literal Cross Reference Record—X'0044' 266
Register Cross Reference Record—X'0045' 267
Library Record—X'0060'. 268
Library Member and Macro Cross Reference Record—X'0062' 269
User-supplied Information Record—X'0070' . . . 270
USING Map Record—X'0080' 270
Statistics record—X'0090' 271

Appendix D. Sample program 277

Appendix E. MHELP sample macro trace and dump 289

Appendix F. High Level Assembler messages 297
Message code format 297
Message descriptions 298
Assembly error diagnostic messages. 299
  Message not known 301
  Messages 301
Abnormal assembly termination messages. . . . 341
ASMAHL Command Error Messages (CMS) . . . 346

Appendix G. User interface macros 349

Appendix H. Sample ADATA user exits (z/OS and CMS) 351
Sample ASMAXADT user exit to filter records . . 351
  Preparing the exit 351
  Preparing the filter modules 352
  Preparing the sample filter module ASMAXFLU 355
  Invoking the exit 357
Sample ASMAXADC user exit to control record output. 357
  Preparing the exit 357
  Invoking the exit 357
  Messages 358
Sample ASMAXADR user exit to reformat records 359
  Preparing the exit 359
  Invoking the exit 359
  Messages 360

Appendix I. Sample LISTING user exit (z/OS and CMS) 361
Function 361
Preparing the exit 361
Invoking the exit 361
Messages 362

Appendix J. Sample SOURCE user exit (z/OS and CMS) 363
Function 363
Preparing the exit 363
Invoking the exit 363

Appendix K. How to generate a translation table 365
Preparing the translation table. 365

Appendix L. How to generate a Unicode translation table 367
Preparing the Unicode translation table. 367

Appendix M. TYPECHECK assembler option 373
Extensions to the DC, DS, and EQU assembler instructions 373
Type checking behavior for REGISTER 375
  Access Register type checking 376
  General Register type checking 377
  Control Register type checking 379
  Floating-Point Register type checking 380
Type checking behavior for MAGNITUDE. . . . 381

Appendix N. HLASM Services Interface 383
Communication and work areas 383
  Invoking the HLASM Services Interface . . . 384
  Get storage service 385
  Return storage service 385
  Time and date service 385
  Write to terminal service 386
Mapping the communication and work areas. . . 386

Appendix O. High Level Assembler for Linux on zSeries 387
Options 387
  Sources of assembler options 387
  Assembler options. 387
ELF32 option 388
ASMAXT2E messages 389

Appendix P. Transactional Memory exit ASMAXTXP 395

Notices 397
Trademarks 398

Bibliography 399

Glossary 401

Index 407

Figures

1. Option summary including options specified on *PROCESS statements 8
2. External Symbol Dictionary listing 10
3. Source and object listing section—121 format 13
4. Source and object listing section. 15
5. Example showing truncation of long ASMA435I message 16
6. Source and object listing section—133 format 18
7. Relocation dictionary (RLD) listing. 19
8. Ordinary symbol and literal cross reference 20
9. Unreferenced symbols defined in CSECTS 22
10. Macro and copy code source summary 22
11. Macro and copy code cross reference 23
12. Macro and copy code cross reference—with LIBMAC option 24
13. Assembly with LIBMAC and PCONTROL(MCALL) options 25
14. Assembly with LIBMAC and NOPCONTROL options 25
15. Assembly with NOLIBMAC and PCONTROL(MCALL) options 26
16. Assembly with NOLIBMAC and NOPCONTROL options 26
17. DSECT cross reference 26
18. USING map 27
19. General Purpose Register cross reference 28
20. Diagnostic cross reference and assembler summary 29
21. Sample terminal output in the NARROW format 33
22. Sample terminal output in the WIDE format 33
23. High Level Assembler product information page 53
24. Effect of the LIMIT suboption 74
25. Exit parameter list format 83
26. Exit-specific information block—LISTING exit 93
27. Exit-specific information block—other exit types. 93
28. Example of a user exit (part 1 of 17) 116
29. Example of a user exit (part 2 of 17) 117
30. Example of a user exit (part 3 of 17) 118
31. Example of a user exit (part 4 of 17) 119
32. Example of a user exit (part 5 of 17) 120
33. Example of a user exit (part 6 of 17) 121
34. Example of a user exit (part 7 of 17) 122
35. Example of a user exit (part 8 of 17) 124
36. Example of a user exit (part 9 of 17) 125
37. Example of a user exit (part 10 of 17) 126
38. Example of a user exit (part 11 of 17) 128
39. Example of a user exit (part 12 of 17) 129
40. Example of a user exit (part 13 of 17) 130
41. Example of a user exit (part 14 of 17) 131
42. Example of a user exit (part 15 of 17) 132
43. Example of a user exit (part 16 of 17) 133
44. Example of a user exit (part 16 of 17) 134
45. SETAF external function parameter list format 137
46. SETCF external function parameter list format 138
47. Sample macro parameter messages 144
48. Sample error diagnostic messages. 145
49. JCL for assembly, using cataloged procedure 150
50. JCL for assembly, using cataloged procedure, with NOOBJECT 150
51. JCL for assembly 151
52. Assembling on TSO 151
53. Sample program to call the assembler dynamically 153
54. High Level Assembler Files 155
55. Using the program management components 162
56. Sample job control for creating a load module 163
57. INCLUDE and LIBRARY control statements 166
58. Job control to pass PARM.C and PARM.L options 168
59. General job control to run a program on z/OS 169
60. Macro library addition procedure 171
61. Cataloged procedure for assembly (ASMAC) 173
62. Cataloged procedure for assembling and linking (ASMACL). 174
63. Cataloged procedure for assembly, link, and run (ASMACLG) 176
64. Cataloged procedure for assembly and running using the loader (ASMACG) . . . 177
65. Access to PARM field 181
66. Sample assembler linkage statements for calling subprograms 182
67. Example of creating two load modules on CMS 185
68. High Level Assembler Files 186
69. JCL to assemble a program 199
70. Entering ICCF commands 201
71. Sample procedure for assembling on ICCF 202
72. Sample program to call the assembler dynamically 203
73. High Level Assembler Files 205
74. Sample job control for creating a phase 209
75. Access to PARM field 215
76. SYM record format. 225
77. Sample assembler program for associated data output 236
78. Sample assembler program for macro-only assembly 238
79. Sample program using MHELP (part 1 of 8) 290
80. Sample program using MHELP (part 2 of 8) 291
81. Sample program using MHELP (part 3 of 8) 292
82. Sample program using MHELP (part 4 of 8) 293
83. Sample program using MHELP (part 5 of 8) 294
84. Sample program using MHELP (part 6 of 8) 295
85. Sample program using MHELP (part 7 of 8) 296
86. Sample program using MHELP (part 8 of 8) 297
87. Sample macro parameter messages 300
88. Creating a filter management table 351

© Copyright IBM Corp. 1992, 2013

89. Passing initial character string to filter routines 352
90. Generating an alternative CSECT name 352
91. Filter module parameter list format 353
92. Initial character string for ASMAXFLU 355
93. Sample output data set from ASMAXFLU 356
94. Sample Translation Table. 366
95. Sample Unicode translation table (part 1 of 4) 368
96. Sample Unicode translation table (part 2 of 4) 369
97. Sample Unicode translation table (part 3 of 4) 370
98. Sample Unicode translation table (part 4 of 4) 371
99. Behavior to assign and retrieve a symbol's types 374
100. Behavior to assign and retrieve a symbol's register types 375
101. Access Register type checking with AR activated 376
102. Access Register type checking with AR inactive 377
103. General Register type checking with GR activated 378
104. General Register type checking with GR inactive 378
105. General Register type checking with GR32 activated 379
106. General Register type checking with GR32 and GR64 activated 379
107. Control Register type checking with CR activated 380
108. Control Register type checking with CR inactive 380
109. Floating-Point Register type checking with FPR activated 381
110. Floating-Point Register type checking with FPR inactive 381
111. MAGNITUDE behavior 382
112. NOMAGNITUDE behavior 382
113. HLASM Services Interface Block 383

Tables

1. IBM High Level Assembler for z/OS & z/VM & z/VSE documents xv
2. Flags used in the option summary 9
3. Types of ESD entries when NOGOFF option specified 9
4. Types of ESD entries when GOFF option specified 10
5. Data set names on CMS 31
6. Data set names on z/VSE 31
7. Unicode-3 SBCS mapping code pages 41
8. Equivalent suboptions for MACHINE and OPTABLE options 57
9. z/OS and CMS exit types 80
10. z/VSE exit types 80
11. User-exit return codes 86
12. User exit reason codes 87
13. Error severity and associated diagnostic message 90
14. z/OS and CMS system variable symbols 94
15. z/VSE system variable symbols 94
16. SOURCE exit processing summary 98
17. LIBRARY exit processing summary 102
18. LISTING exit processing summary 106
19. OBJECT and PUNCH exit processing summary 109
20. ADATA exit processing summary 112
21. TERM exit processing summary 114
22. Message severity and associated messages 140
23. Assembler options and data sets required 151
24. Invoking the assembler dynamically 152
25. Assembler data set characteristics 156
26. Data sets used for linking 165
27. Options for controlling binder output 166
28. Link processing options 166
29. Assembler file characteristics 187
30. CP SET EMSG command options 191
31. Assembler options in JCL 204
32. Assembler file characteristics 206
33. Files used for link-editing 210
34. Linkage editor control statements 211
35. AMODE/RMODE flags 220
36. Organization value byte 223
37. ADATA record—common header section 239
38. Assembler operation code values 260
39. Request to get storage 385
40. Request to return storage 385
41. Request for time and date 385
42. Write a message to the terminal or to SYSTERM 386

© Copyright IBM Corp. 1992, 2013

About this document

This document describes how to use the IBM® High Level Assembler for z/OS® & z/VM® & z/VSE® licensed program, from here on referred to as "High Level Assembler", or "the assembler". It is intended to help you assemble, link, and run your High Level Assembler programs. It is meant to be used with the *HLASM Language Reference*.

Throughout this information, we use these indicators to identify platform-specific information:

- Prefix the text with platform-specific text (for example, "Under CMS...")
- Add parenthetical qualifications (for example, "(CMS)")
- A definition list, for example:

z/OS	Informs you of information specific to z/OS.
z/VM	Informs you of information specific to z/VM.
z/VSE	Informs you of information specific to z/VSE.

CMS is used in this manual to refer to Conversational Monitor System on z/VM.

Who should use this document

The *HLASM Programmer's Guide* is for application programmers coding in the High Level Assembler language. To use this document, you should be familiar with the basic concepts and facilities of your operating system.

Programming interface information

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

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

Organization of this document

Here is how this document is organized:

- **Part 1. Understanding and using the assembler**
  - Chapter 1, "Introduction," on page 1 describes High Level Assembler, and defines the environmental requirements for using the assembler.
  - Chapter 2, "Using the assembler listing," on page 5 describes the content and format of the assembler listing.
  - Chapter 3, "Controlling your assembly with options," on page 35 describes the assembler options that you can use to control the assembly of your program.
  - Chapter 4, "Providing user exits," on page 79 describes how you can provide user exits to compliment the assembler's data set processing.
  - Chapter 5, "Providing external functions," on page 135 describes how to provide user-supplied routines in conditional assembly instructions to set the value of SET symbols.
  - Chapter 6, "Diagnosing assembly errors," on page 143 describes the purpose and format of error messages, MNOTEs, and the MHELP trace facility.

© Copyright IBM Corp. 1992, 2013

- **Part 2. Developing Assembler Programs on z/OS**
  - Chapter 7, “Assembling your program on z/OS,” on page 149 describes the different methods of assembling your program on z/OS, including invoking the assembler with job control statements, invoking the assembler on TSO/E, invoking the assembler dynamically, and batch assembling.
  - Chapter 8, “Linking and running your program on z/OS,” on page 161 describes linking, creating load modules, input and output for the linkage editor and binder, detecting linking errors, and running your program on z/OS.
  - Chapter 9, “z/OS system services and programming considerations,” on page 171 describes the z/OS 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 on z/OS. This chapter also discusses programming topics such as standard entry and exit procedures.
- **Part 3. Developing Assembler Programs on CMS**
  - Chapter 10, “Assembling your program on CMS,” on page 183 describes how to invoke the assembler on CMS.
  - Chapter 11, “Running your program on CMS,” on page 193 describes how to load and run your program on CMS.
  - Chapter 12, “CMS system services and programming considerations,” on page 195 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.
- **Part 4. Developing Assembler Programs on z/VSE**
  - Chapter 13, “Assembling your program on z/VSE,” on page 199 describes how to invoke the assembler on z/VSE.
  - Chapter 14, “Link-editing and running your program on z/VSE,” on page 209 describes link-editing, creating load modules, input and output for the linkage editor, detecting link-edit errors, and running your program on z/VSE.
  - Chapter 15, “z/VSE system services and programming considerations,” on page 213 describes the VSE 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 on VSE. This chapter also discusses programming topics such as standard entry and exit procedures.
- **Appendixes**
  - Appendix A, “Cross-system portability considerations,” on page 217 contains information that helps you prepare your program for running under a different operating system.
  - Appendix B, “Object deck output,” on page 219 describes the format of the object module generated by the assembler.
  - Appendix C, “Associated data file output,” on page 227 describes the format of the associated data file records generated by the assembler.
  - Appendix D, “Sample program,” on page 277 provides a sample program that demonstrates many of the assembler language features.
  - Appendix E, “MHELP sample macro trace and dump,” on page 289 provides a sample program listing which shows the primary functions of MHELP.
  - Appendix F, “High Level Assembler messages,” on page 297 describes the error diagnostic messages, abnormal termination messages, and CMS command error messages issued by the assembler.
  - Appendix G, “User interface macros,” on page 349 lists the macros that are provided as Programming Interfaces with High Level Assembler.
  - Appendix H, “Sample ADATA user exits (z/OS and CMS),” on page 351 provides a description of the sample ADATA user exits supplied with High Level Assembler.
  - Appendix I, “Sample LISTING user exit (z/OS and CMS),” on page 361 provides a description of the sample LISTING user exit supplied with High Level Assembler.

  - Appendix J, "Sample SOURCE user exit (z/OS and CMS)," on page 363 provides a description of the sample SOURCE user exit supplied with High Level Assembler to read variable length input files.
  - Appendix K, "How to generate a translation table," on page 365 provides instructions for generating a translation table to convert the characters contained in character data constants and literals.
  - Appendix M, "TYPECHECK assembler option," on page 373 provides information about how the TYPECHECK option controls the type checking of machine instruction operands performed by the assembler.
  - Appendix N, "HLASM Services Interface," on page 383 describes the cooperative interface between High Level Assembler and its I/O exits.
  - Appendix O, "High Level Assembler for Linux on zSeries," on page 387 provides information relevant to the Linux on zSeries® implementation of High Level Assembler.
- The "Glossary" on page 401 defines the terms used in this document.
- The "Bibliography" on page 399 lists the IBM Publications referred to within this document.

High Level Assembler documents

High Level Assembler runs under z/OS, z/VM, and z/VSE. Its publications for the z/OS, z/VM, and z/VSE operating systems are described in this section.

Documents

Here are the High Level Assembler documents. The table shows tasks, and which document can help you with that particular task. Then there is a list showing each document and a summary of its contents.

Table 1. IBM High Level Assembler for z/OS & z/VM & z/VSE documents

Task	Document	Order Number
Installation and customization	HLASM V1R6 Installation and Customization Guide	SC26-3494
	HLASM V1R6 Programmer's Guide	SC26-4941
	HLASM V1R6 Toolkit Feature Installation Guide	GC26-8711
Application Programming	HLASM V1R6 Programmer's Guide	SC26-4941
	HLASM V1R6 Language Reference	SC26-4940
	High Level Assembler for Linux on zSeries User's Guide V1 R5	SC18-9611
	HLASM V1R6 Toolkit Feature User's Guide	GC26-8710
	HLASM V1R6 Toolkit Feature Interactive Debug Facility User's Guide	GC26-8709
Diagnosis	HLASM V1R6 Installation and Customization Guide	SC26-3494
Warranty	HLASM V1R6 Licensed Program Specifications	GC26-4944

HLASM 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.

HLASM Language Reference

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

HLASM Licensed Program Specifications

Contains a product description and product warranty information for High Level Assembler.

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

HLASM for Linux on zSeries User's Guide
Describes differences and extensions you need to consider when using High Level Assembler for Linux on zSeries.

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

HLASM Toolkit Feature User's Guide
Describes how to use the High Level Assembler Toolkit Feature.

HLASM Toolkit Feature Debug Reference Summary
Contains a reference summary of the High Level Assembler Interactive Debug Facility.

HLASM Toolkit Feature Interactive Debug Facility User's Guide
Describes how to use the High Level Assembler Interactive Debug Facility.

Collection kits

The High Level Assembler publications are available in these collection kits:
- *z/OS V1Rx and Software Products DVD Collection*, SK3T-4271 (Book and PDF)
- *z/OS V1Rx Information Center DVD*, SK5T-7089
- *z/VM Collection on DVD*, SK5T-7054
- *z/VM VxRy Information Center DVD*, SK5T-7098
- *z/VSE Collection*, SK3T-8348

For more information about High Level Assembler, see the High Level Assembler Web site, at

`http://www-306.ibm.com/software/awdtools/hlasm/`

Related publications

See "Bibliography" on page 399 for a list of publications that supply information you might need while using High Level Assembler.

Syntax notation

Throughout this book, syntax descriptions use this structure:
- 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) or uppercase and lowercase (for example, PATHFile). They must be spelled exactly as shown. Lowercase letters are optional (for example, you could enter the PATHFile keyword as PATHF, PATHFI, PATHFIL, or PATHFILE).

  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─┘
```

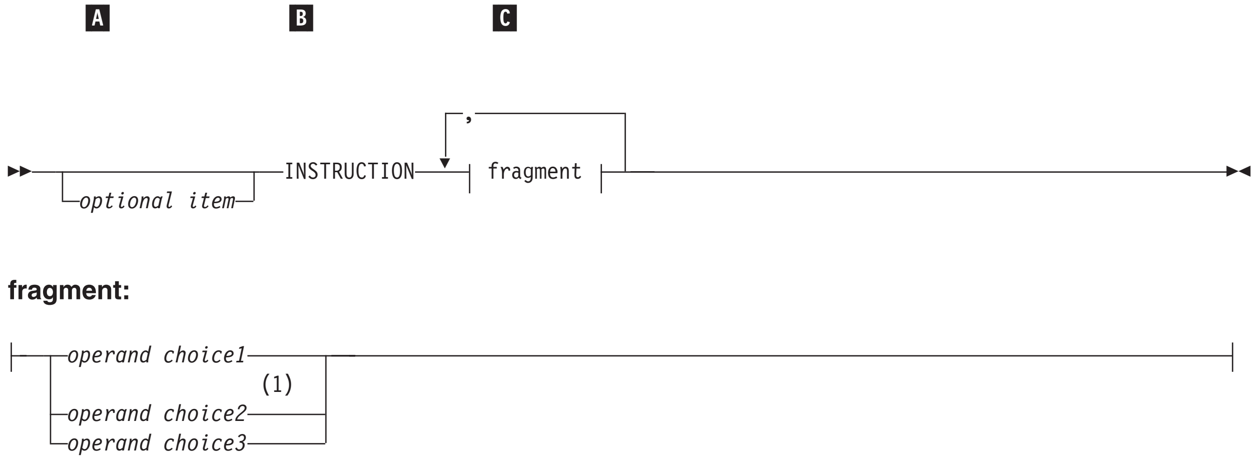
- An arrow returning to the left above the main line indicates an item that can be repeated. When the repeat arrow contains a separator character, such as a comma, you must separate items with the separator character.

```
                  ┌─,───────────────┐
                  ▼                 │
►►──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.

Format

The following example shows how the syntax is used.



Notes:

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 “fragment” is a required operand. Allowable choices for this operand are given in the fragment of the syntax diagram shown below “fragment” 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.

How to send your comments to IBM

If you especially like or dislike anything about this book, feel free to send us your comments.

You can comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information that is in this book and to the way in which the information is presented. Speak to your IBM representative if you have suggestions about the product itself.

When you send us comments, you grant to IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

You can get your comments to us quickly by sending an e-mail to **idrcf@hursley.ibm.com**. Alternatively, you can mail your comments to:

User Technologies,
IBM United Kingdom Laboratories,
Mail Point 095, Hursley Park,
Winchester, Hampshire,
SO21 2JN, United Kingdom

Please ensure that you include the book title, order number, and edition date.

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:

- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support web page

© Copyright IBM Corp. 1992, 2013

Summary of changes

Date of Publication
August 2013

Form of Publication
Seventh Edition, SC26-4941-HLASM Programmer's Guide

New features

- High Level Assembler for Linux on System z®

New options

- WORKFILE

Changed assembler instructions

- DC/DS
  - Decimal floating-point constants
  - Unsigned binary constants
  - Tags allowing use of macros and machine or assembler instructions with identical mnemonics.

Changed assembler statements

- Parts with text
- OPTABLE option for ACONTROL

Services Interface

- HLASM Services Interface for I/O exits added

Miscellany

- Qualifiers identified in symbol cross-reference.

© Copyright IBM Corp. 1992, 2013

Chapter 1. Introduction

IBM High Level Assembler for z/OS & z/VM & z/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)
- z/Architecture®
- Linux for System z

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 these operating systems. Unless otherwise stated, the assembler also operates under subsequent versions, releases, and modification levels of these systems:
- OS/390® Version 2 Release 10.0 (5647-A01)
- VM/ESA Version 3 Release 1.0 (5654-A17)
- z/VM Version 5 Release 2 (5741-A05)
- VSE/ESA Version 2 Release 6 (5690-VSE)
- z/VSE Version 3 Release 1 andVersion 4 (5686-CF8)
- z/OS Version 1 Release 2.0 (5694-A01)

High Level Assembler supports the operation codes available with the following mode processors:
- Extended Architecture (370-XA)
- Enterprise Systems Architecture/370 (ESA/370)
- Enterprise Systems Architecture/390 (ESA/390)
- z/Architecture

Machine requirements

For assembling High Level Assembler programs: Programs written using High Level Assembler can be assembled, including use of the z/Architecture processor machine instructions, 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:
- z/OS
- z/VM
- z/VSE
- OS/390
- VM/ESA
- VSE/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 z/Architecture, Extended Architecture (370-XA), Enterprise Systems Architecture/370 (ESA/370), Enterprise Systems

© Copyright IBM Corp. 1992, 2013

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 6250 bpi
- 3480 tape cartridge

An appropriate tape device is required for installation.

Double-byte data: Double-byte data can be displayed, entered, or both, in their multicultural 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 610 KB of main storage. 410 KB 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
- Print output
- Object module output
- Associated data output
- I/O exits
- External functions

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

Installation: Please refer to *HLASM 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, High Level Assembler Release 2, High Level Assembler Release 3, High Level Assembler Release 4, High Level Assembler Release 5, Assembler H Version 2 or DOS/VSE Assembler to High Level Assembler Release 6, 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 6 symbolic operation codes, or SET symbols with names that conflict with the names of High Level Assembler Release 6 system variable symbols.

Except for these possible conflicts, and with appropriate High Level Assembler option values, assembler language source programs written for High Level Assembler Release 1, High Level Assembler Release 2, High Level Assembler Release 3, High Level Assembler Release 4, High Level Assembler Release 5, 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 6.

Object Programs: Object programs generated by High Level Assembler Release 6 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.

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 about the listing options, refer to Chapter 3, “Controlling your assembly with options,” on page 35.

The High Level Assembler listing consists of up to twelve 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
- General Purpose Register Cross Reference
- 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 (z/OS and CMS) 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 are printed on each page.

© Copyright IBM Corp. 1992, 2013

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.

RXREF
Produces the General Purpose Register Cross Reference 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 on 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 in the ASMAOPT file (z/OS and CMS) or library member ASMAOPT.USER (z/VSE).
- 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.

On z/OS and CMS, High Level Assembler provides a sample LISTING exit that allows you to suppress the option summary or print it at the end of the listing. See Appendix I, “Sample LISTING user exit (z/OS and CMS),” on page 361.

Figure 1 on page 8 shows an example of the High Level Assembler Option Summary. The example includes assembler options that have been specified in the following option sources:

- ASMAOPT file
- Invocation parameters
- PROCESS statements including an example specifying the OVERRIDE keyword.

The example shows a number of error diagnostic messages relating to the conflicts and errors in the options specified.

```
                                       High Level Assembler Option Summary                                         Page    1
                                                                                             1          2
                                                                                         HLASM R6.0  2008/07/11 17.48
0  Overriding ASMAOPT Parameters - sysparm(thisisatestsysparm),rxref
   Overriding Parameters-  NOOBJECT,language(en),size(4meg),xref(short,unrefs),nomxref,norxref,adata,noadata
   Process Statements-     OVERRIDE(ADATA,MXREF(full))
                           ALIGN
                           noDBCS
                           MXREF(FULL),nolibmac
                           FLAG(0)
                           noFOLD,LANGUAGE(ue)
                           NORA2
                           NODBCS
                           XREF(FULL)

      3
 ** ASMA400W Error in invocation parameter - size(4meg)
 ** ASMA438N Attempt to override ASMAOPT parameter.  Option norxref ignored.
 ** ASMA425N Option conflict in invocation parameters. noadata overrides an earlier setting.
 ** ASMA423N Option ADATA, in a *PROCESS OVERRIDE statement conflicts with invocation or default option. Option is not
 permitted in a *PROCESS statement and has been ignored.
 ** ASMA422N Option LANGUAGE(ue) is not valid in a *PROCESS statement.
 ** ASMA437N Attempt to override invocation parameter in a *PROCESS statement. Suboption FULL of XREF option ignored.

  Options for this Assembly

  4
3 NOADATA
5   ALIGN
  NOASA
    BATCH
    CODEPAGE(047C)
  NOCOMPAT
5 NODBCS
  NODECK
    DXREF
    ESD
  NOEXIT
5   FLAG(0,ALIGN,CONT,EXLITW,NOIMPLEN,NOPAGE0,PUSH,RECORD,NOSUBSTR,USING0)
5 NOFOLD
  NOGOFF
  NOINFO
3   LANGUAGE(EN)
5 NOLIBMAC
    LINECOUNT(60)
    LIST(121)
    MACHINE(,NOLIST)
1   MXREF(FULL)
3 NOOBJECT
    OPTABLE(UNI,NOLIST)
  NOPCONTROL
  NOPESTOP
  NOPROFILE
5 NORA2
  NORENT
    RLD
2   RXREF
    SECTALGN(8)
    SIZE(MAX)
  NOSUPRWARN
2   SYSPARM(thisisatestsysparm)
  NOTERM
```

```
1                                         High Level Assembler Option Summary                                   Page    2
-                                                                                            HLASM R6.0  2008/07/11 17.48
0  NOTEST
     THREAD
   NOTRANSLATE
     TYPECHECK(MAGNITUDE,REGISTER)
     USING(NOLIMIT,MAP,WARN(15))
 3   XREF(SHORT,UNREFS)
   NOWORKFILE

   5
   No Overriding DD Names
1                                              External Symbol Dictionary                                 Page    3
-Symbol   Type   Id     Address  Length   Owner Id Flags Alias-of                           HLASM R6.0  2008/07/11 17.48
0A         SD 00000001 00000000 00000000             00
1                                                                                                         Page    4
   Active Usings: None
0  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                 HLASM R6.0  2008/07/11 17.48
                                          6
0                                     1 *PROCESS OVERRIDE(ADATA,MXREF(full))
                                      2 *PROCESS ALIGN
                                      3 *PROCESS noDBCS
                                      4 *PROCESS MXREF(FULL),nolibmac
                                      5 *PROCESS FLAG(0)
                                      6 *PROCESS noFOLD,LANGUAGE(ue)
                                      7 *PROCESS NORA2
                                      8 *PROCESS NODBCS
                                      9 *PROCESS XREF(FULL)
 000000               00000 00000    10 A        CSECT
                  R:F  00000          11          USING *,15
```

*Figure 1. Option summary including options specified on *PROCESS statements*

The highlighted numbers in the example are:

1 The product description. Shown on 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 The date and the time of the assembly.

3 Error diagnostic messages for overriding parameters and *PROCESS statements. These messages immediately follow the list of *PROCESS statement options. The error diagnostic messages are:

ASMA400W -
The value specified as the size option is not valid. The valid option is SIZE(4M).

ASMA438N -
The option RXREF is specified in the ASMAOPT file and the conflicting option NORXREF is specified as an invocation parameter. The ASMAOPT options have precedence over the invocation parameters and the NORXREF option is ignored.

ASMA425N -
The ADATA option specified as an invocation parameter overrides the option NOADATA which was also specified as an invocation parameter. When conflicting options are received from the same source, the last occurrence takes precedence.

ASMA423N -
The option ADATA has been specified in a *PROCESS statement with the OVERRIDE option. The option cannot be set by a *PROCESS statement, and the option conflicts with an invocation or default option. This message is printed when an option that cannot be set by a *PROCESS statement (See "*PROCESS statement options" on page 37) is included in a *PROCESS OVERRIDE statement and the option conflicts with an invocation or default option. If the option does not conflict with the invocation or default option no message is printed.

ASMA422N -
The option LANGUAGE is not permitted in a *PROCESS statement.

ASMA437N -
The option XREF(FULL) which is specified in the last *PROCESS statement conflicts with the option NORXREF which is specified as an invocation parameter. The option XREF(FULL) is ignored.

4 A flag beside each option indicates the source of the option. This table shows the sources:

Table 2. Flags used in the option summary

Flag	Meaning
1	The option came from a *PROCESS OVERRIDE statement.
2	The option came from the ASMAOPT options file (z/OS and CMS) or ASMAOPT.USER library member (z/VSE).
3	The option came from the invocation parameters.
4	The permanent job control options set by the z/VSE command STDOPT.
5	The option came from a *PROCESS statement.
(space)	The option came from the installation defaults.

5 On z/OS and CMS, if the assembler has been called by a program 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
```

6 The *PROCESS statements are written as comment statements in the Source and Object section of the listing.

See “Precedence of assembler options” on page 35.

External Symbol Dictionary (ESD)

This section of the listing contains the external symbol dictionary information passed to the linkage editor or loader, or z/OS Program Management Binder, in the object module.

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

The ESD entries describe the control sections, external references, classes, parts, and entry points in the assembled program. There are nine types of ESD entries (SD, ED, LD, ER, PC, PR, CM, XD, and WX). Table 3 shows the ESD entries when you specify the NOGOFF option. Table 4 on page 10 shows the ESD entries when you specify the GOFF option. For each of the different types of ESD entries, an X indicates which of the fields have values.

Table 3. Types of ESD entries when NOGOFF option specified

SYMBOL	TYPE	ID	ADDR	LENGTH	Owner 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				

Table 4. Types of ESD entries when GOFF option specified

SYMBOL	TYPE	ID	ADDR	LENGTH	Owner ID	FLAGS
X	SD	X				X[1]
X	ED	X	X	X	X	X
X	LD	X	X		X	X
X	PR	X	X	X	X	X
X	ER	X			X	
X	CM	X	X		X	X
X	XD	X	X	X		
X	WX	X			X	

Note:

1. Only if the SD ESDID entry is associated with CM ESDID entry.

Figure 2 is an example of the *External Symbol Dictionary* generated with the GOFF assembler option, and is followed by a description of its contents.

Note: This sample is intended to show the various fields and possible values and is not meant to be an actual listing.

```
SAMP01                                        External Symbol Dictionary                                     Page    2
1        2      3        4        5         6          7         8
Symbol   Type   Id     Address  Length   Owner Id Flags Alias-of                            HLASM R6.0  2008/07/11 17.48
SAMP01    SD 00000001
B_PRV     ED 00000002                   00000001
B_TEXT    ED 00000003 00000000 000000E0  00000001   00
SAMP01    LD 00000004 00000000           00000003   00
ENTRY1    LD 00000005 00000000           00000003   00
KL_INST   SD 00000006
B_PRV     ED 00000007                    00000006
B_TEXT    ED 00000008 00000000 00000000  00000006   00
KL_INST   CM 00000009 00000000           00000008   00
          SD 0000000A
B_PRV     ED 0000000B                    0000000A
B_TEXT    ED 0000000C 000000E0 00000000  0000000A   00
Date0001  ER 0000000D                    0000000A       RCNVDATE
RCNVTIME  ER 0000000E                    0000000A
```

Figure 2. External Symbol Dictionary listing

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

When you specify the GOFF assembler option on z/OS or CMS, the assembler generates two ED entries, the first with a symbol name of B_PRV and the second with a symbol name of B_TEXT.

2 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.

When you specify the GOFF assembler option, on z/OS or CMS, the assembler generates an SD entry type for the symbol which appears in the field name of a COM instruction.

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

When you specify the GOFF assembler option, on z/OS or CMS, the assembler generates an entry type of LD for each CSECT and RSECT name.

ER External reference. The symbol appeared as the operand of an EXTRN statement, or appeared as an operand of 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 could depend on, or could affect or be affected by the location counter) before a CSECT, RSECT, START, DSECT, or COM.

When you specify the GOFF assembler option, on z/OS or CMS, the assembler does not generate an LD entry for a private code section. For private code, the assembler creates an SD entry type with a blank name.

PR Part definition. The symbol appeared in the PART parameter of a CATTR statement.

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 the operand of a Q-type address constant.

The external dummy section is also called a pseudo register in the applicable *Linkage Editor and Loader* document, and *z/OS DFSMS Program Management*.

Note: The alignment value of an external dummy section is displayed in the address field of the External Symbol Dictionary. However, its value is encoded in different fields of the object file, not the address.

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

ED Element definition (one for each class). The symbol appeared as an operand in a CATTR statement.

When you specify the NOGOFF assembler option, on z/OS or CMS, the assembler does not generate an entry type of ED.

For further information, refer to *z/OS DFSMS Program Management*.

3 The external symbol dictionary identification number (ESDID). The number is a unique eight-digit hexadecimal number identifying the entry.

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

5 The assembled length, in bytes, of the section, element, or DXD (in hexadecimal notation).

6 The ESDID of the control section in which the LD ID is the ESDID of the following, depending on the type:

LD The control section in which the label is defined.

ED The control section to which this element belongs.

CM The control section in which the common control section is defined.

ER The control section in which the external reference was declared.

PR The class to which this part belongs.

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

```
Bit 2:    0  = use the RMODE bit (5)
          1  = RMODE is 64
Bit 3:    0  = use the AMODE bits (6-7)
          1  = AMODE is 64
```

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

8 When symbol **1** is defined in an ALIAS instruction, this field shows the name of the external dummy section, control section, entry point, part, or external symbol 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.

On z/OS and CMS, 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 generalized object format data set (see "GOFF (z/OS and CMS)" on page 51). The 133-character format also contains the first eight characters of the macro name in the identification-sequence field for statements generated by macros. Figure 3 on page 13 shows an example of the Source and Object section of the listing. This section shows the source statements of the module, and the object code of the assembled statements.

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 in "LANGUAGE" on page 53.

Figure 3 on page 13 shows an example of the Source and Object section in 121-character format, and in mixed-case.

```
 1        2
SAMP01   Sample Listing Description                                                                            Page    3
  Active Usings: None
   3       4               5         6          7                                               8               9
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
000000                00000 000E0     2 Samp01   Csect
                                     22 Entry1   SAMPMAC Parm1=YES                                              00002300
000000 18CF                          23+Entry1   LR 12,15                                                       01-SAMPM
                                     24+         ENTRY Entry1                                                   01-SAMPM
                       12
                 R:C  00000          25+         USING Entry1,12              Ordinary Using                  01-SAMPM
000002 0000 0000            00000    26+         LA Savearea,10                                                 01-SAMPM
10   ** ASMA044E Undefined symbol - Savearea
10   ** ASMA029E Incorrect register specification - Savearea
11   ** ASMA435I Record 6 in SAMP01   MACLIB   A1(SAMPMAC) on volume: EAR191
000006 50D0 A004            00004    27+         ST 13,4(,10)                                                   01-SAMPM
00000A 50A0 D008            00008    28+         ST 10,8(,13)                                                   01-SAMPM
00000E 18DA                          29+         LR 13,10                                                       01-SAMPM
               R:A35  00010          30+         USING *,10,3,5               Ordinary Using,Multiple Base    01-SAMPM
** ASMA303W Multiple address resolutions may result from this USING and the USING on statement number 25 13
 ** ASMA435I Record 10 in SAMP01   MACLIB   A1(SAMPMAC) on volume: EAR191
                                                                                                                 14
                                     31+         DROP 10,3,5                  Drop Multiple Registers         01-SAMPM
                                     32          COPY  SAMPLE                                                   00002400
                                     33=*        Line from member SAMPLE
               C 02A  00000 0002A    34          Using IHADCB,INDCB       Establish DCB addressability          00002500
               C 07A  00000 0007A    35 ODCB     Using IHADCB,OUTDCB                                            00002600
                                     36          push  using                                                    00002700
                    15
               R:2  00000            37 PlistIn  Using Plist,2            Establish Plist addressability        00002800
               R:3  00000            38 PlistOut Using Plist,3                                                  00002900
SAMP01   Sample Listing Description                                                                            Page    4
16     Active Usings (1):Entry1,R12  IHADCB(X'FD6'),R12+X'2A'  PlistIn.plist,R2  PlistOut.plist,R3
  ODCB.IHADCB(X'F86'),R12+X'7A'
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
000010 1851                          40 ?Branch  LR    R5,R1              Save Plist pointer                    00003100
** ASMA147E Symbol too long, or first character not a letter - ?Branch
** ASMA435I Record 30 in SAMP01   ASSEMBLE A1 on volume: EAR191
000012 5820 5000            00000    41          L     R2,0(,R5)          R2 = address of request list          00003200
000016 47F0 C022            00022    42          B     Open                                                     00003300
                                    697          End                                                            00055100
0000D0 00000001                     698                =f'1'
0000D4 00000000                     699                =V(RCNVDATE)
0000D8 00000000                     700                =V(RCNVTIME)
0000DC 00000002                     701                =f'2'
```

Figure 3. Source and object listing section—121 format

1 The deck identification, if any, consisting of 1–8 characters. 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 The information taken from the operand field of a TITLE statement.

3 Location field. This field is the value of the location counter that represents the assembled address (in hexadecimal notation) of the object code.

- For ORG statements, the value of the location counter before the ORG is placed in the location column, and the value of the location counter after the ORG is placed in the Addr2 field.
- If the END statement contains an operand, the operand value (requested entry point) appears in the location field.
- 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 The object code produced by the source statement. The entries, which are shown left-aligned and in hexadecimal notation, are machine instructions or assembled constants. Machine instructions are printed in full with a space inserted after every four 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 “PRINT instruction” in the *HLASM 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 **15** for more details.

5 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:
  - For a USING instruction, the value of the first operand.
  - For a CSECT, START, LOCTR, or RSECT instruction, the start address of the control section.
  - For an ORG instruction, the value of the location counter before the ORG.
  - For an EQU instruction, the value assigned.
- 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.
  - For a CSECT, START, LOCTR, or RSECT instruction, the Addr2 field contains the end address of the control section.
  - For an ORG instruction, the Addr2 field contains the next address as specified by the operand field.
  - For an EQU instruction, the Addr2 field contains the length assigned.

If the assembler option LIST(121) is in effect, both address fields contain six digits; however, if the high-order digit is 0, it is not printed. If the assembler option LIST (133) is in effect, both address fields contain eight digits. For USING and EQU instructions, the Addr2 field can contain up to eight digits.

6 The statement number. The column following the statement number contains one of these values:

- A plus sign (+) 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 (-) indicates that the statement was read by a preceding AREAD instruction.
- An equals sign (=) indicates that the statement was included by a COPY instruction.
- A greater-than sign (>) indicates that the statement was generated as the result of a preceding AINSERT instruction. If the statement is read by an AREAD instruction, this takes precedence and a minus sign is printed.

7 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 with 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 only 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 instruction statements in the source program that contain variable symbols are listed twice: first, as they appear in the source input, and second, with values substituted for the variable symbols. See Figure 4 for an example of this.
- All error diagnostic messages appear in line except those suppressed by the FLAG option. Chapter 6, “Diagnosing assembly errors,” on page 143 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 4.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      1 &A SETC 'abcdefghijklmnop'                                              00001000
                                      2 &A       LA   4,1       Comment                                          00002000
000000 4140 0001            00001      +abcdefghijklmnop LA 4,1                                                X00002000
                                       +                       Comment
                                      3 &b SETC 'abc'                                                            00003000
                                      4 &b       LA   4,1       Comment                                          00004000
000004 4140 0001            00001      +abc      LA   4,1       Comment                                          00004000
```

Figure 4. 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 The release level of High Level Assembler.

9 The date and time at the start of the assembly.

10 The error diagnostic messages immediately follow 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 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.

If the input data set containing the source in error is a z/OS UNIX System Services file, message ASMA435I can continue over more than one print line. If the inclusion of the path name in the message causes the message to be more than 255 bytes in length, the message is truncated.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
000004 5810 0001           00001     6         1     1,1
** ASMA309W Operand 1 resolved to a displacement with no base register
** ASMA033I Storage alignment for 1 unfavorable
** ASMA435I Record 6 in /u/carland/assembler/source/dataset.which/has/a.very/longname/thus/causing/this/message/tobe/prin
ted/OVER/more/than/one/line/ on volume:
```

Figure 5. Example showing truncation of long ASMA435I message

12 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-aligned in the object code columns, preceded by the characters “R:”.

11 A second instance of the informational message ASMA435I (also mentioned in **1**). Conditional assembly statements and comment statements contribute to the record count of macro definitions, as suggested by the record number which is greater than the number of generated statements.

14 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 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.

16 The current USING PUSH level is printed after the heading and before the first active USING. If the USING PUSH level is zero, it is not shown.

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
- 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

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 (n):` *`label.sectname+offset(range),registers`*

where:

n	Is the current PUSH level. If the PUSH level is zero, it is not shown. If no USING statements are active, the heading appears as `Active Usings: None`.
label	Is the label name specified for a Labeled USING. If the USING is not labeled, this field is omitted.
sectname	Is 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	Is the offset from the specified section that is used to resolve the USING. This field is omitted if it is zero.
(range)	Is the number of bytes addressed by this base register for instructions with 12-bit displacement fields. It is only shown if the default value (any multiple of X'1000') is not used.
registers	Is 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 character string 'MORE ...' is appended to the last line.

In Figure 6 on page 18, the USINGs at statements 25 and 30 are ordinary USINGs. The USING at statement 34 is a dependent USING, and that at statement 35 is a labeled dependent USING. The USINGs at statements 37 and 38 are labeled USINGs.

Figure 6 on page 18 also shows an example of the *Source and Object* section when the same assembly is run with assembler option LIST(133), and is followed by a description of differences with respect to Figure 3 on page 13:

```
SAMP01   Sample Listing Description                                                                            Page    3
  Active Usings: None
     1
  Loc    Object Code      Addr1    Addr2    Stmt  Source Statement                          HLASM R6.0  2008/07/11 17.48
00000000                00000000 000000E0      2 Samp01   Csect
                                              22 Entry1   SAMPMAC Parm1=YES                                              00002300
00000000 18CF                                 23+Entry1   LR 12,15                                                       01-SAMPMAC
                                              24+         ENTRY Entry1                                                   01-SAMPMAC
                               2
                    R:C 00000000              25+         USING Entry1,12                Ordinary Using                  01-SAMPMAC

00000002 0000 0000               00000000     26+         LA Savearea,10                                                 01-SAMPMAC
** ASMA044E Undefined symbol - Savearea
** ASMA029E Incorrect register specification - Savearea
** ASMA435I Record 6 in SAMP01   MACLIB   A1(SAMPMAC) on volume: EAR191
00000006 50D0 A004               00000004     27+         ST 13,4(,10)                                                   01-SAMPMAC
0000000A 50A0 D008               00000008     28+         ST 10,8(,13)                                                   01-SAMPMAC
0000000E 18DA                                 29+         LR 13,10                                                       01-SAMPMAC
                                                                                                                            3

                  R:A35 00000010              30+         USING *,10,3,5                 Ordinary Using,Multiple Base    01-SAMPMAC

** ASMA303W Multiple address resolutions may result from this USING and the USING on statement number 25
** ASMA435I Record 10 in SAMP01   MACLIB   A1(SAMPMAC) on volume: EAR191
                                              31+         DROP 10,3,5                    Drop Multiple Registers         01-SAMPMAC
                                              32          COPY  SAMPLE                                                   00002400
                                              33=*        Line from member SAMPLE
                  C 02A 00000000 0000002A     34          Using IHADCB,INDCB       Establish DCB addressability          00002500
                  C 07A 00000000 0000007A     35 ODCB     Using IHADCB,OUTDCB                                            00002600
                                              36          push  using                                                    00002700
                    R:2 00000000              37 PlistIn  Using Plist,2            Establish Plist addressability        00002800
                    R:3 00000000              38 PlistOut Using Plist,3                                                  00002900

SAMP01   Sample Listing Description                                                                            Page    4
  Active Usings (1):Entry1,R12  IHADCB(X'FD6'),R12+X'2A'  PlistIn.plist,R2  PlistOut.plist,R3
  ODCB.IHADCB(X'F86'),R12+X'7A'
  Loc    Object Code      Addr1    Addr2    Stmt  Source Statement                          HLASM R6.0  2008/07/11 17.48
00000010 1851                                 40 ?Branch  LR    R5,R1              Save Plist pointer                    00003100
** ASMA147E Symbol too long, or first character not a letter - ?Branch
** ASMA435I Record 30 in SAMP01   ASSEMBLE A1 on volume: EAR191
00000012 5820 5000               00000000     41          L     R2,0(,R5)          R2 = address of request list          00003200
00000016 47F0 C022               00000022     42          B     Open                                                     00003300
                                             697          End                                                            00055100
000000D0 00000001                            698                =f'1'
000000D4 00000000                            699                =V(RCNVDATE)
000000D8 00000000                            700                =V(RCNVTIME)
000000DC 00000002                            701                =f'2'
```

Figure 6. Source and object listing section—133 format

1 The Addr1 and Addr2 columns show eight-character operand addresses.

2 The assembled address of the object code occupies eight characters.

3 The first eight characters of the macro name are shown in the identification-sequence field.

Relocation dictionary (RLD)

This section of the listing describes the relocation dictionary information passed to the linkage editor or loader, or z/OS 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.

Note: The Relocation dictionary listing shown in Figure 7 on page 19 is not related to the program segment shown in Figure 6.

```
TSSC004                                           Relocation Dictionary                                    Page    6
     1        2        3        4     5
   Pos.Id   Rel.Id   Address  Type  Action                                             HLASM R6.0  2008/07/11 17.48
  00000003 00000002 00000000   J 4    ST
  00000003 00000005 0000001C  RI 2     +
  00000003 00000005 00000020   A 4     +
  00000003 00000005 00000026  RI 2     +
  00000003 00000006 0000005A  RI 4     -
  00000003 00000006 00000060   A 4     -
  00000003 00000007 0000005A  RI 4     +
  00000003 00000007 00000060   A 4     +
  00000003 00000002 00000000   J 4    ST
  00000003 00000002 0000006C   V 4    ST
```

Figure 7. Relocation dictionary (RLD) listing

1 The external symbol directory ID (in hexadecimal notation) of the element or part within which this address constant resides.

2 The external symbol directory ID (in hexadecimal notation) of the element or part which is used as the basis for relocation.

3 The assembled address (in hexadecimal notation) of the field where the address constant is stored.

4 The type and length of the address constant. The type contains one of these values:

A	A-type address constant
V	V-type address constant
Q	Q-type address constant
J	J-type address constant or CXD instruction
R	R-type address constant
RI	Relative Immediate offset

5 The relocation action which contained one of these values:

+	The relocation operand is added to the address constant
-	The relocation operand is subtracted from the address constant
ST	The relocation operand overwrites the address constant

Ordinary symbol and literal cross reference

This section of the listing concerns symbols and literals that are defined and used in the program.

```
SAMP01                         Ordinary Symbol and Literal Cross Reference                                                           Page    6
1          2        3        4   5   6    7     8         9     10
Symbol   Length   Value     Id     R Type Asm  Program   Defn References                                           HLASM R6.0  2008/07/11 17.48
DCBBIT0       1 00000080 FFFFFFFF A   U                   68  101   109   122   146   179   181   182   184   207   210   230   234   249   286
                                                              344
DCBBIT1       1 00000040 FFFFFFFF A   U                   69  102   110   124   147   148   157   163   179   181   183   184   212   230   232
                                                              234   252   253   254   289   290   344
DCBBIT2       1 00000020 FFFFFFFF A   U                   70  103   111   125   126   127   147   148   152   158   163   179   180   185   214
                                                              235   236   257   258   259   293   294   345
DCBBIT3       1 00000010 FFFFFFFF A   U                   71  104   125   127   128   147   160   186   217   235   238   261   262   263   297
                                                              298   345
DCBBIT4       1 00000008 FFFFFFFF A   U                   72  112   161   162   163   187   218   240   245   246   266   267   301   302   304
                                                              305   346
DCBBIT5       1 00000004 FFFFFFFF A   U                   73  113   168   190   191   220   240   242   243   246   270   272   273   274   308
                                                              309   310   311   346
DCBBIT6       1 00000002 FFFFFFFF A   U                   74  105   169   170   173   190   192   221   277   278   279   280   314   315   316
                                                              317   347
DCBBIT7       1 00000001 FFFFFFFF A   U                   75  106   169   171   173   194   225   282   283   320   321   323   324
Entry1        2 00000000 00000003     I                   23   24    25U
IHADCB        1 00000000 FFFFFFFF     J                   56   34U   35U   83   132   203   328   335   353
INDCB         2 0000002A 00000003     H  H                46   34U
Open          2 00000022 00000003     H  H               366   42B
OUTDCB        2 0000007A 00000003     H  H                48   35U
plist         1 00000000 FFFFFFFE     J                  360   37U   38U
PlistIn  ***QUALIFIER*** 00000001     U                   37   38
RCNVDATE      1 00000000 0000000D     T                  699  699
RCNVTIME      1 00000000 0000000E     T                  700  700
r1            1 00000001 00000003 A   U                  369   40
r2            1 00000002 00000003 A   U                  370   41M
r5            1 00000005 00000003 A   U                  373   40M   41
Samp01        1 00000000 00000003     J                    2   20    45    47   356   363   365
Savearea ***UNDEFINED*** 00000000 A   U                        26
=f'1'         4 000000D0 00000003     F                  698  396
=f'2'         4 000000DC 00000003     F                  701  400
=V(RCNVDATE)
              4 000000D4 00000003     V                  699  397
=V(RCNVTIME)
              4 000000D8 00000003     V                  700  399
```

Figure 8. Ordinary symbol and literal cross reference

1 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 was not previously defined.

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

In the following example, the assembler lists the symbol name `inPUT`, because the third statement defines `inPUT` as an external symbol. The assembler also lists the symbol name `INput`, because the fifth statement defines `INput` as an ordinary symbol.

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

2 Shows, in decimal notation, the byte length of the field represented by the symbol. This field is blank for labeled USINGs (see symbol `WA`).

3 Shows the hexadecimal address assigned to the symbol or literal, or the hexadecimal value to which the symbol is equated. This field is blank for labeled USING symbols. The Value column can contain the UNDEFINED or QUALIFIER strings, depending on the properties of the symbol.

4 For symbols and literals defined in an executable control section or a 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 Column title R is an abbreviation for “Relocatability Type”. Symbols `fl12nd` and `WA` are absolute symbols and are flagged “A” in the R column. Symbol `jix` is the result of a complex relocatable expression and is flagged “C” in the R column. Symbol `IOerror` is relocatable and is not flagged.

6 Indicates the type attribute of the symbol or literal. Refer to “Type attribute (T')” in the *HLASM Language Reference* for details.

7 Indicates the assembler type of the symbol, including type extensions (if any). Refer to *HLASM Language Reference* for details.

8 Indicates the program type of the symbol. Refer to *HLASM Language Reference* for details.

9 Is the statement number in which the symbol or literal was defined.

10 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 this column contains the message:

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

A symbol name can appear in the cross reference section as both an external symbol name and an ordinary symbol name. In this situation there is no duplication.

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 Length (**2**) and Value (**3**) 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 symbols are shown in symbol name order. To obtain this section of the listing, specify the XREF(UNREFS) assembler option.

```
SAMP01                            Unreferenced Symbols Defined in CSECTs                                       Page   19
   1        2
  Defn  Symbol                                                                              HLASM R6.0  2008/07/11 17.48
    47  ODCB
    49  PlistIn
    50  PlistOut
     7  R0
    10  R3
    16  Unreferenced_Long_Symbol
```

Figure 9. 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 is useful for checking that you have included the correct version of a macro or copy code member.

```
SAMP01                                    Macro and Copy Code Source Summary                                   Page   27
   1     2                                              3         4
 Con Source                                         Volume    Members                       HLASM R6.0  2008/07/11 17.48
  L1 SAMP01   MACLIB   A1                           FAL191    SAMPLE   SAMPMAC
  L2 OSMACRO  MACLIB   S2                           MNT190    DCBD     IHBERMAC
```

Figure 10. Macro and copy code source summary

1 Contains a number representing the concatenation order of macro and copy code libraries. (This number is not shown when the Source **2** is “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. If the name is more than 44 characters in length, the path name of the library is truncated. The truncated path name is suffixed with an ellipsis to indicate that it has been truncated.

2 Shows either the name of each library from which the assembler reads a macro or a copy code member or, for in-line macros, the words “PRIMARY INPUT”.

Note: The file names in the example are from CMS (not z/OS or z/VSE).

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

4 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. You can use the assembler option MXREF(XREF) or MXREF(FULL) to generate this section of the listing.

```
SAMP01                                    Macro and Copy Code Cross Reference                               Page   28
1         2     3                4       5
Macro     Con  Called By        Defn  References                                    HLASM R6.0  2008/07/11 17.48
A              PRIMARY INPUT     826  971, 973, 998
AINSERT_TEST_MACRO
               PRIMARY INPUT       3  16
AL             PRIMARY INPUT     873  981, 983
DCBD       L3  PRIMARY INPUT       -  113
IHBERMAC   L3  DCBD                -  113
L              PRIMARY INPUT     816  966, 968
MAC1           PRIMARY INPUT      28  36
N              PRIMARY INPUT     933  991
O              PRIMARY INPUT     953  993
SAMPLE     L1  PRIMARY INPUT       -  85C   6
SAMPMAC    L1  PRIMARY INPUT       -  64
SAVE       L3  PRIMARY INPUT       -  42
SL             PRIMARY INPUT     883  986, 988
ST             PRIMARY INPUT     836  976, 978
TYPCHKRX       PRIMARY INPUT     745  775, 845, 892
X              PRIMARY INPUT     943  996
XIT1       L1  PRIMARY INPUT       -  30C
XIT2       L2  PRIMARY INPUT       -  32C
XIT3       L1  PRIMARY INPUT       -  34C
```

Figure 11. Macro and copy code cross reference

1 The macro or copy code member name.

2 Shows the value representing the input source concatenation, as listed in the Macro and Copy Code Source Summary (refer to Figure 10 on page 22) and under the sub-heading “Datasets Allocated for this Assembly” in the Diagnostic Cross Reference and Assembler Summary (refer to Figure 20 on page 29).

3 Shows either the name of the macro that calls this macro or copy code member, or the words “PRIMARY INPUT” indicating 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 One of these:

- The statement number for macros defined in the primary input file,
- A dash (-) indicating the macro or copy code member was retrieved from a library.

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

Lookahead Processing: If a COPY instruction is encountered during lookahead, this 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 number shown against the copied member is one less than the statement number containing the inner macro call instruction. See “Effects of LIBMAC and PCONTROL(MCALL) options” on page 24 for examples of assemblies using different combinations of the LIBMAC and PCONTROL(MCALL) options.

6 Statement numbers have a suffix of “C” when the reference is to a member named on a COPY instruction.

Figure 12 on page 24 shows the format of the Macro and Copy Code Cross Reference when you specify the assembler option, LIBMAC.

```
SAMP01                                    Macro and Copy Code Cross Reference                                Page   81
Macro     Con  Called By        Defn  References                                             HLASM R6.0  2008/07/11 17.48
                                    1
A              PRIMARY INPUT    3667  3812, 3814, 3839
AINSERT_TEST_MACRO
               PRIMARY INPUT       3  16
AL             PRIMARY INPUT    3714  3822, 3824
DCBD      L3   PRIMARY INPUT     224X 2329
IHBERMAC  L3   DCBD             2331X 2954
L              PRIMARY INPUT    3657  3807, 3809
MAC1           PRIMARY INPUT      28  36
N              PRIMARY INPUT    3774  3832
O              PRIMARY INPUT    3794  3834
SAMPLE    L1   PRIMARY INPUT       -  195C
SAMPMAC   L1   PRIMARY INPUT     153X 174
SAVE      L3   PRIMARY INPUT      43X 130
SL             PRIMARY INPUT    3724  3827, 3829
ST             PRIMARY INPUT    3677  3817, 3819
TYPCHKRX       PRIMARY INPUT    3586  3616, 3686, 3733
X              PRIMARY INPUT    3784  3837
XIT1      L1   PRIMARY INPUT       -  30C
XIT2      L2   PRIMARY INPUT       -  32C
XIT3      L1   PRIMARY INPUT       -  34C
```

Figure 12. Macro and copy code cross reference—with LIBMAC option

1 The "X" flag indicates the macro was read from a macro library and embedded in the input source program immediately preceding the invocation of that macro. For example, in Figure 12, you can see that `SAMPMAC` was called by the `PRIMARY INPUT` stream from LIBRARY `L1`, at statement number `174`, after being embedded in the input stream at statement number `153`. 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 combinations 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.

The following four figures illustrate the effects of using the various combinations of the LIBMAC and PCONTROL(MCALL) assembler options.

Figure 13 on page 25 shows the output when you specify the LIBMAC and PCONTROL(MCALL) options.

```
                                                                                                               Page    2
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      1          MACRO                                                          00001000
                                      2          MACOUTER                                                       00002000
                                      3          MACINNER                                                       00003000
                                      4          MEND                                                           00004000
                                      5          MACOUTER                                                       00001000
                                      6          MACRO                                                          00001000
                                      7          MACINNER                                                       00002000
                                      8          COPY COPYCODE                                                  00003000
                                      9          MNOTE *,'MNOTE FROM MEMBER COPYCODE'                           00001000
  +----------------------------+     10          MEND                                                           00004000
  | This line produced because |---->11+         MACINNER                                                       01-00003
  | PCONTROL(MCALL) specified  |     12++*,MNOTE FROM MEMBER COPYCODE                                           02-00008
  +----------------------------+     13          END                                                            00002000

                                          Macro and Copy Code Source Summary                                   Page    3
 Con Source                                        Volume    Members                        HLASM R6.0  2008/07/11 17.48
  L1 TEST     MACLIB   A1                          ADISK     COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                  Page    4
Macro     Con  Called By       Defn  References                                             HLASM R6.0  2008/07/11 17.48
COPYCODE   L1  MACINNER           -  8C
MACINNER   L1  MACOUTER          7X 11
MACOUTER   L1  PRIMARY INPUT     2X 5
```

Figure 13. Assembly with LIBMAC and PCONTROL(MCALL) options

Figure 14 shows the output when you specify the LIBMAC and NOPCONTROL options.

```
                                                                                                               Page    2
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      1          MACRO                                                          00001000
                                      2          MACOUTER                                                       00002000
                                      3          MACINNER                                                       00003000
                                      4          MEND                                                           00004000
                                      5          MACOUTER                                                       00001000
                                      6          MACRO                                                          00001000
                                      7          MACINNER                                                       00002000
                                      8          COPY COPYCODE                                                  00003000
                                      9          MNOTE *,'MNOTE FROM MEMBER COPYCODE'                           00001000
                                     10          MEND                                                           00004000
                                     11++*,MNOTE FROM MEMBER COPYCODE                                           02-00008
                                     12          END                                                            00002000

                                          Macro and Copy Code Source Summary                                   Page    3
 Con Source                                        Volume    Members                        HLASM R6.0  2008/07/11 17.48
  L1 TEST     MACLIB   A1                          ADISK     COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                  Page    4
Macro     Con  Called By       Defn  References                                             HLASM R6.0  2008/07/11 17.48
COPYCODE   L1  MACINNER           -  8C
MACINNER   L1  MACOUTER          7X 10
MACOUTER   L1  PRIMARY INPUT     2X 5
```

Figure 14. Assembly with LIBMAC and NOPCONTROL options

Figure 15 on page 26 shows the output when you specify the NOLIBMAC and PCONTROL(MCALL) options.

```
                                                                                                               Page    2
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      1          MACOUTER                                                       00001000
     This line produced because ────→2+          MACINNER                                                       01-MACOU
     PCONTROL(MCALL) specified        3++,MNOTE FROM MEMBER COPYCODE                                            02-MACIN
                                      4          END                                                            00002000

                                          Macro and Copy Code Source Summary                                   Page    3
 Con Source                                        Volume    Members                        HLASM R6.0  2008/07/11 17.48
  L1 TEST     MACLIB   A1                          ADISK     COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                  Page    4
Macro     Con  Called By       Defn  References                                             HLASM R6.0  2008/07/11 17.48
COPYCODE   L1  MACINNER           -  1C
MACINNER   L1  MACOUTER           -  2
MACOUTER   L1  PRIMARY INPUT      -  1
```

Figure 15. Assembly with NOLIBMAC and PCONTROL(MCALL) options

Figure 16 shows the output when you specify the NOLIBMAC and NOPCONTROL options.

```
                                                                                                               Page    2
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      1          MACOUTER                                                       00001000
                                      2++,MNOTE FROM MEMBER COPYCODE                                            02-MACIN
                                      3          END                                                            00002000

                                          Macro and Copy Code Source Summary                                   Page    3
 Con Source                                        Volume    Members                        HLASM R6.0  2008/07/11 17.48
  L1 TEST     MACLIB   A1                          ADISK     COPYCODE MACINNER MACOUTER

                                          Macro and Copy Code Cross Reference                                  Page    4
Macro     Con  Called By       Defn  References                                             HLASM R6.0  2008/07/11 17.48
COPYCODE   L1  MACINNER           -  1C
MACINNER   L1  MACOUTER           -  1
MACOUTER   L1  PRIMARY INPUT      -  1
```

Figure 16. Assembly with NOLIBMAC and NOPCONTROL 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 began.

```
                                                Dsect Cross Reference                                          Page   26
  1         2          3        4
Dsect     Length      Id       Defn                                                         HLASM R6.0  2008/07/11 17.48
AXPRIL   0000003C  FFFFFFFD     655
AXPSIL   00000410  FFFFFFFC     771
AXPXITP  00000014  FFFFFFFE     641
IHADCB   00000060  FFFFFFFB     799
Statement
         00000050  FFFFFFFA    1370
WORKAREA 000001A8  FFFFFFFF     595
```

Figure 17. 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 indicates the control section ID assigned to the dummy control section. You can use this field with the ID field in the Ordinary Symbol and Literal Cross Reference (see Figure 8 on page 20) 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   27
                                                                                            HLASM R6.0  2008/07/11 17.48
  1       2         3      4       5           6        7           8      9    10     11   12
Stmt  -----Location----- Action ----------------Using------------------ Reg Max      Last  Label and Using Text
        Count      Id           Type          Value    Range     Id         Disp     Stmt
 170  00000000  00000001 USING  ORDINARY    00000000 00001000 00000001  15 0002A     171  asmaxinv,r15
 175  00000030  00000001 DROP                                           15                asmaxinv,r15
 185  00000034  00000001 USING  ORDINARY    00000000 00001000 00000001  12 00000          asmaxinv,r12
 186  00000034  00000001 USING  ORDINARY    00000000 00001000 FFFFFFFD   7 00034     508  axpril,r07
 187  00000034  00000001 USING  ORDINARY    00000000 00001000 FFFFFFFA   8 00048     464  Statement,r08
 188  00000034  00000001 USING  ORDINARY    00000000 00001000 FFFFFFFC  10 00404     474  axpsil,r10
 189  00000034  00000001 USING  ORDINARY    00000000 00001000 FFFFFFFB  11 00052     465  ihadcb,r11
 190  00000034  00000001 USING  ORDINARY    00000000 00001000 00000001  12 00589     519  asmaxinv,r12
 202  0000004E  00000001 USING  LABELED     00000000 00001000 FFFFFFFF   1 00000          WA.WorkArea,r01
 203  0000004E  00000001 USING  LAB+DEPND  +0000014A 00000EB6 FFFFFFFB   1                local.ihadcb,WA.mydcb
 204  0000004E  00000001 PUSH
 205  00000054  00000001 DROP                                            1                local
 212  0000006A  00000001 POP                                             1                WA
 213  0000006A  00000001 USING  ORDINARY    00000000 00001000 FFFFFFFF  13 0014A     527  WorkArea,r13
 214  0000006A  00000001 USING  ORDINARY    00000000 00001000
```

Figure 18. 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 USING, or POP USING statement was encountered.

3 Shows the value of the ESDID of the current section when the USING, DROP, PUSH USING, or POP USING 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 Shows the range of the USING. For more information, see "USING instruction" in the *HLASM Language Reference*.

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

9 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 two asterisks (**).

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.

10 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.

11 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.

12 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 either of the assembler options, USING(NOMAP) or NOUSING.

General Purpose Register cross reference

This section of the listing shows all references in the program to each of the general registers. Additional flags indicate the type of reference. This is a useful tool in checking the logic of your program; it helps you see if your use of registers is in order.

```
                                        General Purpose Register Cross Reference                   Page    8
Register  References (M=modified, B=branch, U=USING, D=DROP, N=index)          HLASM R6.0  2008/07/11 17.48
    1       2
    0(0)  115
    1(1)  118  120  121  122  124  126  127  128  130  131  133  135  136  137
    2(2)   36   37   38   39   40   41   42   43   44M  45   46   47   48   49   50   51
           52M  53   54   55M  56   57   58   59M  60   61   62   63   64   65   66   67
           68   69   70   71   72M  73   74   75   76   77   78   79   80   81   82   83
           84   85   86   87   88   89M  90   91   92   93M  94   95   96   97   98   99
          100  101  102  103  104  105  106  107  108  109  110  111  112
    3(3)  (no references identified)    3
    4(4)   16M 281
    5(5)  283
    6(6)   66N 167N 170  171  174  178  180N 190  192  193  194  197  199  200  201N
    7(7)  283
    8(8)  283
    9(9)  224  225  226  227
   10(A)  255U 342D
   11(B)  237  238  239N 240  241  242  243N 244  245N 271
   12(C)    8U
   13(D)  261  262  263  264  265  266
   14(E)  209  210  211  212  213  214  215  216
   15(F)   34  144
```

Figure 19. General Purpose Register cross reference

1 Lists the 16 general registers (0–15).

2 The statements within the program that reference the register. Additional indicators are suffixed to the statement numbers as follows:

(space)
Referenced

M Modified

B	Used as a branch address
U	Used in USING statement
D	Used in DROP statement
N	Used as an index register

3 The assembler indicates when it has not detected any references to a register.

Note: The implicit use of a register to resolve a symbol to a base and displacement does not create a reference in the General Purpose Register Cross Reference.

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.

The sample listing shown in Figure 20 contains a combination of z/OS and CMS data sets to show examples of the differences in data set information.

Note: For a complete list of the diagnostic messages issued by the assembler, see Appendix F, “High Level Assembler messages,” on page 297.

```
                                  Diagnostic Cross Reference and Assembler Summary                                   Page    9
                                                                                            &relname;  &reldate;
Statements Flagged
          1     1(P1,0), 3(P1,3), 4(P1,4), 5(P1,5), 6(P1,6), 7(P1,7), 8(P1,8), 170(L3:DCBD,2149)
    2  8 Statements Flagged in this Assembly       16 was Highest Severity Code High Level Assembler, 5696-234, RELEASE 6.0   3
SYSTEM: CMS 16                     JOBNAME: (NOJOB)     STEPNAME: (NOSTEP)   PROCSTEP: (NOPROC)  4
Datasets Allocated for this Assembly  5
 Con DDname   Data Set Name                               Volume  Member
   A1 ASMAOPT  XITDIS   OPTIONS  A1                        ADISK
   P1 SYSIN    XITDIS   ASSEMBLE A1                        ADISK
   L1 SYSLIB   TEST     MACLIB   A1                        ADISK
   L2          DSECT    MACLIB   A1                        ADISK
   L3          OSMACRO  MACLIB   S2                        MNT190
   L4          OSMACRO1 MACLIB   S2                        MNT190
 6    SYSLIN   XITDIS   TEXT     A1                        ADISK
      SYSPRINT XITDIS   LISTING  A1                        ADISK

External Function Statistics  7
 ----Calls----  Message   Highest  Function
  SETAF   SETCF    Count  Severity  Name
3       1        5      22    MSG
1       0        2       8    MSG1
1       0        1       0    MSG2
 8
Input/Output Exit Statistics
Exit Type  Name      Calls  ---Records---  Diagnostic
                             Added Deleted  Messages
LIBRARY    CTLXIT      258      0       0        2
LISTING    ASMAXPRT    195      0      52        0
 9
Suppressed Message Summary
Message     Count   Message     Count   Message     Count   Message     Count   Message     Count
   169        0       306       0       309       0       320       0
 10  622K allocated to Buffer Pool,
 11   16 Primary Input Records Read   13  3072 Library Records Read                0 Work File Reads
 12    1 ASMAOPT Records Read         14   141 Primary Print Records Written       0 Work File Writes
 15    2 Punch Records Written        16     0 ADATA Records Written
Assembly Start Time: 12.06.06 Stop Time: 12.06.07 Processor Time: 00.00.00.1771   17
Return Code 016
```

Figure 20. Diagnostic cross reference and assembler summary

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 statement number as shown in the source and object section of the listing.

dsnum is the value applied to the source or library data set, 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 data sets listed under the sub-heading "Datasets Allocated for this Assembly" **5** .

member
is the name of the macro from which the statement was read. On z/OS, this can 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 data set 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 Option Summary section of the listing (see Figure 1 on page 8).

For a complete discussion of how error messages and MNOTEs are handled, see Chapter 6, "Diagnosing assembly errors," on page 143.

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. This information is:

- The name and level of the operating system used to run the assembly.
- The job name for the assembly job. If the job name is not available, "(NOJOB)" is printed.
- The step name for the assembly job. If the step name is not available, "(NOSTEP)" is printed.
- The procedure name for the assembly job. If the procedure name is not available, "(NOPROC)" is printed.

5 On z/OS and CMS, 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. On z/OS, the data set information can also include the name of a member of a partitioned data set (PDS) or library (PDSE).

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 "Exit-Specific Information Block" on page 92.

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.

z/OS: On z/OS, the data set name for all data sets is extracted from the z/OS 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: On CMS, the data set name is assigned one of the values shown in Table 5 on page 31.

Table 5. Data set names on CMS

File Allocated To:	Data Set Name
CMS file	The 8-character file name, the 8-character file type, and the 2-character file mode of the file, each separated by a space. 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`
Labeled tape file	The data set name of the tape file
Unlabeled tape file	`TAP`*n*, where *n* is a value from 0 to 9, or from A to F.
Terminal	`TERMINAL`

z/VSE: On z/VSE, the data set name is assigned one of the values shown in Table 6.

Table 6. Data set names on z/VSE

File Allocated To:	Data Set Name
Disk	The file-id
Job stream (SYSIPT)	None
Library (Disk). The ddname is shown as `*LIB*`.	The file-id
Printer	None
Punch	None
Labeled tape file	The file ID of the tape file
Unlabeled tape file	None
Terminal (TERM)	None

6 Output data sets do not have a concatenation value.

7 The usage statistics of external functions for the assembly. The following statistics are reported:

SETAF function calls
The number of times the function was called from a SETAF assembler instruction.

SETCF function calls
The number of times the function was called from a SETCF assembler instruction.

Messages issued
The number of times the function requested that a message be issued.

Messages severity
The maximum severity for the messages issued by this function.

Function Name
The name of the external function module.

8 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-aligned and leading zeros are suppressed, unless the count is zero.

9 The message number of each message specified for suppression, and the count of the number of times it was suppressed during the assembly.

10 The minimum storage required for an in-storage assembly.

11 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.

12 The number of records read from the ASMAOPT file (z/OS and CMS) or the Librarian member (z/VSE) by the assembler.

13 The number of records read from the libraries allocated to SYSLIB on z/OS and CMS, or assigned to the Librarian on z/VSE. This count does not include any records read or discarded by the LIBRARY user exit.

14 The count of the actual number of records generated by the assembler. If you have used the SPACE *n* assembler instruction, the count might be less than the total number of printed and blank lines appearing in the listing. For a SPACE *n* that does not cause an eject, the assembler inserts *n* blank lines in the listing by generating *n*/3 triple-spaced blank records, rounded to the next lower integer if a fraction results. 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.

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

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

17 On z/VSE, the assembly start and stop times in hours, minutes, and seconds.

On z/OS and CMS, 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 time 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 time does not include the time used during assembly termination, which deallocates main storage and data sets.

On z/OS and CMS, High Level Assembler provides a sample listing exit which allows you to suppress the Diagnostic Cross Reference and Assembler Summary. See Appendix I, “Sample LISTING user exit (z/OS and CMS),” on page 361.

Terminal output

On z/OS and CMS, the TERM option lets you receive a summary of the assembly at your terminal. You can direct the terminal output to a disk data set.

On z/VSE, the TERM option lets you send a summary of the assembly to SYSLOG.

The output from the assembly includes all error diagnostic messages and the source statement in error. It also shows the number of flagged statements and the highest severity code.

The terminal output can be shown in two formats. Figure 22, the wide format, shows the source statements in the same columns as they were in the input data set. Figure 21, the narrow format, shows the source statements which have been compressed by replacing multiple consecutive blanks with a single blank. Use the TERM assembler option to control the format.

```
 1 &abc setc l'f 00000100
ASMA137S Invalid character expression - l'f
000000 3 dc c'' 00000300
ASMA068S Length error - '
Assembler Done      2 Statements Flagged / 12 was Highest Severity Code
```

Figure 21. Sample terminal output in the NARROW format

```
                                      1 &abc   setc  l'f
                                00000100
ASMA137S Invalid character expression - l'f
000000                                3    dc   c''
                                00000300
ASMA068S Length error - '
Assembler Done      2 Statements Flagged / 12 was Highest Severity Code
```

Figure 22. Sample terminal output in the WIDE format

You can replace or modify the terminal output using a TERM user exit. See Chapter 4, “Providing user exits,” on page 79.

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) requests a facility, and an alternate negative form (such as NOOBJECT) excludes the facility.
- Keywords, such as LINECOUNT(50), that permit you to assign a value to a function.

High Level Assembler accepts options from five different sources:
- External file (z/OS and CMS)
- Library member (z/VSE)
- Invocation parameter
- JCL Option (z/VSE)
- *PROCESS statement

This chapter describes the different sources for assembler options, each assembler option, and when you can use the options. Each option has a default value that the assembler uses if you do not specify an alternative value. The way in which the default values are arrived at is explained under "Default options" on page 37.

The sources of assembler options

This section describes in detail the different sources of assembler options, and the general rules for specifying the options.

Before describing the sources, the section sets out the rules of precedence by which the assembler works out what to do when an option is specified two or more times with differing values.

Precedence of assembler options

Assembler options are recognized in this order of precedence (highest to lowest):
1. Fixed installation default options
2. Options on *PROCESS OVERRIDE statements
3. Options in the ASMAOPT file (CMS or z/OS) or library member (z/VSE)
4. Invocation options
   a. Options on the JCL PARM parameter of the EXEC statement on z/OS and z/VSE, or the ASMAHL command on CMS
   b. Options on the JCL OPTION statement (z/VSE)
   c. Options specified using the STDOPT (Standard JCL Options) command (z/VSE)
5. Options on *PROCESS statements
6. Non-fixed installation default options

You can specify an option as often as you want, and in as many sources as you want. If you specify the positive and negative form of an option (for keyword pairs), or two or more different values (for a value keyword), these option specifications are "conflicting" options. If all specifications of a particular option are at the same level of precedence, the last specification takes effect, and a warning message is issued. If some conflicting specifications are at different levels of precedence, then the specification at the higher order of precedence takes effect, and (in general) the assembler issues a warning message for each option that is not accepted. A warning message is not issued if the higher level of precedence is a *PROCESS OVERRIDE statement, unless the option cannot be set by a *PROCESS statement.

© Copyright IBM Corp. 1992, 2013

Fixed installation default 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.

*PROCESS OVERRIDE Statement Options

If the keyword OVERRIDE is added to a process (*PROCESS) statement, then the nominated assembler option is not overridden by specifications at a lower level of precedence. Furthermore, if there is a conflicting specification at a lower level of precedence, the assembler does *not* issue a warning message.

If an option specified on the process statement is an option listed in "*PROCESS statement options" on page 37, and a different value is supplied as an ASMAOPT or invocation option, then the ASMAOPT or invocation value is accepted, and the assembler issues a warning message.

ASMAOPT options

High Level Assembler accepts options from an external file (z/OS and CMS) with the DDname ASMAOPT or library member (z/VSE) with the name and type ASMAOPT.USER. The file or library member can contain multiple records. All records are read from the file or library member.

The contents of each record are used to build an internal buffer containing the complete option list. Each record is processed until the first space is encountered, unless the option being processed is a SYSPARM option which is enclosed in quotes. The assembler appends the options string from each record, separating the strings with commas. For example, these records:

```
ADATA,MXREF
SYSPARM(TESTPARM)
XREF(FULL)
```

generate this option list:

```
ADATA,MXREF,SYSPARM(TESTPARM),XREF(FULL)
```

Invocation options

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

z/OS In z/OS batch, you select the options by specifying them on 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 cataloged 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'
```

For more examples on how to specify options in a cataloged procedure, see "Overriding statements in cataloged procedures" on page 178.

On 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'
```

z/VM 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[)]
```

z/VSE

In batch, you select the options by specifying them in the PARM field of the EXEC JCL statement that invokes the assembler. You can also specify some of the options on the JCL OPTION statement. For example:

```
// OPTION TERM
// EXEC ASMA90,SIZE=ASMA90,PARM='LIST,DBCS'
```

z/VSE ICCF: On ICCF, you select the options by specifying them in the PARM field of the job entry statement /LOAD that invokes the assembler. For example:

```
/LOAD ASMA90
PARM='LIST,DBCS'
```

*PROCESS statement options

Process (*PROCESS) statements let you specify 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.

If you add the OVERRIDE keyword, then the option has a higher precedence than values set as default and invocation options (see "*PROCESS OVERRIDE Statement Options" on page 36).

Some options cannot be set by a process statement. These options are:

```
ADATA                       LANGUAGE                    PRINT (CMS)
ASA (z/OS and CMS)          LINECOUNT                   SIZE
DECK                        LIST                        SYSPARM
DISK (CMS)                  MACHINE                     TERM
EXIT                        NOPRINT (CMS)               TRANSLATE
GOFF (z/OS and CMS)         OBJECT                      XOBJECT (z/OS and CMS)
```

If you specify one of these options on a normal process statement, the assembler issues a warning message.

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

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. During an assembly you cannot override default options that were fixed during installation (see "Fixed installation default options" on page 36). The assembler issues a message if you try to override a fixed option.

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.

Invoking the assembler dynamically

Assembler options can be passed in a parameter list when the assembler is invoked dynamically from an executing program. For further information, refer to "Invoking the assembler dynamically" on page 152 (for the z/OS platform) or "Invoking the assembler dynamically" on page 203 (for the z/VSE platform).

Coding rules

The rules for coding the assembler options are:

- You can specify the options in any order.
- 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.

z/VM

- If you specify two or more options, the options can be separated by spaces or commas.

z/OS

- If you specify two or more options, the list of options must be enclosed within apostrophes 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 apostrophes or parentheses can be omitted.
- All options that have suboptions must be within apostrophes because they contain special characters.
- 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 apostrophes must not be continued on another record.

z/VSE

- You must enclose the options in apostrophes and separate each option with a comma.
- If you need to continue the PARM field onto another record, place any character in column 72 of the record you want to continue, and continue in column 16 on the following record.
- The operating system passes to the assembler any spaces you code in the PARM=JCL parameter, including those implied in a continuation. For example:

  ```
  // EXEC ASMA90,SIZE=(ASMA90,50K),PARM='RENT,SIZE(MAX-500K),EXIT(LIBEXIT/
                 (EDECKXIT))'
  ```

  is not equivalent to:

  ```
  // EXEC ASMA90,SIZE=(ASMA90,50K),PARM='RENT,SIZE(MAX-500K),            /
                 EXIT(LIBEXIT(EDECKXIT))'
  ```

  The second example results in this diagnostic message:

  ```
  ** ASMA400W ERROR IN INVOCATION PARAMETER - EXIT (LIBEXIT(EDECKXIT))
  ```

Additional rules for coding the assembler options in the external file or library member are:

- Trailing spaces are ignored.
- If two or more options are specified in a record they must be separated by a comma.
- Only a SYSPARM option enclosed in quotes can be continued on another record.
- If a SYSPARM option contains embedded spaces it must be enclosed in quotes.
- The maximum length of the options list including the delimiting commas inserted by the assembler is 32766.

Assembler options

Here are descriptions of the assembler options. The IBM-supplied default value for each option is shown above the main path in the syntax diagram. Your installation might have a different default (see “Default options” on page 37). There are minor variations for Linux on zSeries. See “Assembler options” on page 387 for details.

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 on z/OS and CMS, or with the SYSADAT file name on z/VSE. Appendix C, “Associated data file output,” on page 227 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.

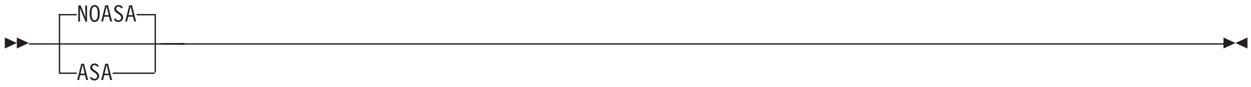
NOALIGN
Instructs the assembler not to check alignment of machine instruction data references not always requiring operand alignment, but still to check instruction references and machine instruction data references always requiring operand alignment. DC, DS, and DXD are to be aligned on the correct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

Notes:

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

2. If your program is assembled with data areas or DSECT for interfacing with IBM products, use the default (ALIGN) unless directed otherwise.
3. On z/VSE, you can specify the ALIGN option on the JCL OPTION statement.

ASA (z/OS and CMS)



Default
NOASA

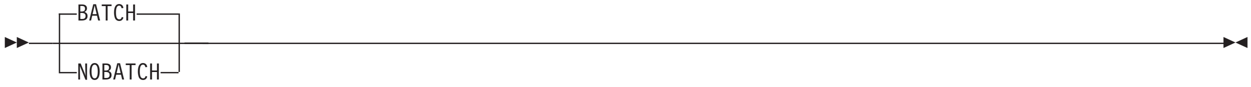
Abbreviations
None

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



Default
BATCH

Abbreviations
None

BATCH
Instructs the assembler that multiple assembler source programs might be in the input data set. The first statement of the second and subsequent source programs must immediately follow the assembled 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.

CODEPAGE

```
     ┌─CODEPAGE(1148|X'47C')───┐
►►───┤                         ├──────────────────────────────────────────────────────►◄
     └─CODEPAGE(nnnnn|X'xxxx')─┘
```

Default
CODEPAGE(1148|X'47C')

Abbreviation
CP

1148|X'47C'
Specifies that characters contained in the Unicode character (CU-type) data constants (DCs) are to be converted using the ECECP: International 1 Unicode-3 mappings contained in module ASMA047C.

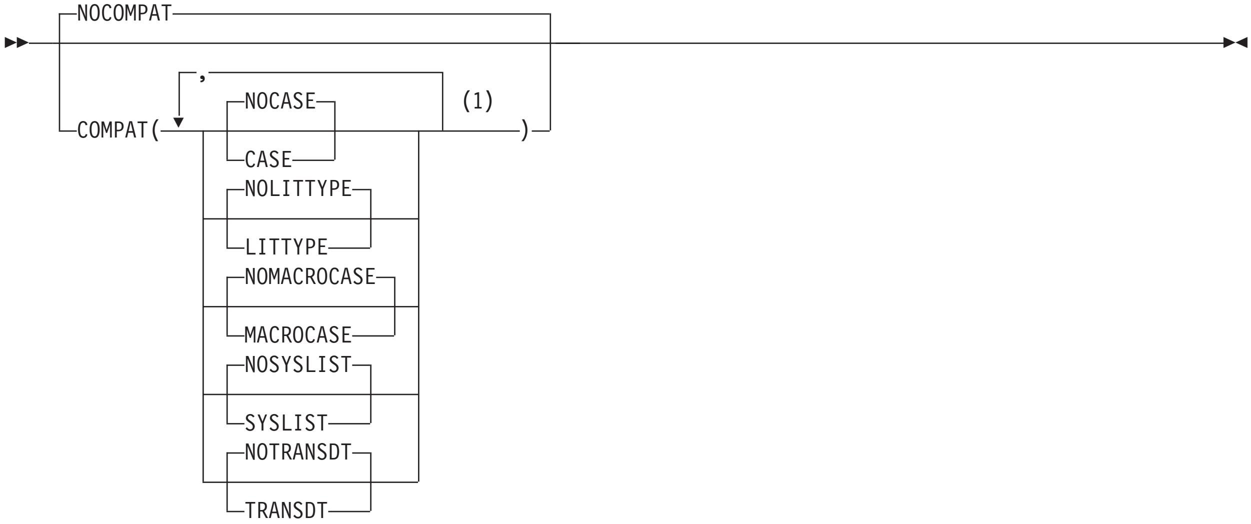
nnnnn|X'xxxx'
Specifies that characters contained in the Unicode character (CU-type) data constants (DCs) are to be converted using the Unicode mapping table module ASMA*xxxx* where *xxxx* is the hexadecimal value of the number *nnnnn* which is the number of the code page contained in the module. The number must be in the range 1 through 64k-1.

The following Unicode-3 code pages are supported by the assembler:

Table 7. Unicode-3 SBCS mapping code pages

Code Page	Module Name	Description
1140	ASMA0474	ECECP: USA, Canada, Netherlands, Portugal, Brazil, Australia, New Zealand
1141	ASMA0475	ECECP: Austria, Germany
1142	ASMA0476	ECECP: Denmark, Norway
1143	ASMA0477	ECECP: Finland, Sweden
1144	ASMA0478	ECECP: Italy
1145	ASMA0479	ECECP: Spain, Latin America (Spanish)
1146	ASMA047A	ECECP: United Kingdom
1147	ASMA047B	ECECP: France
1148	ASMA047C	ECECP: International 1

COMPAT



Notes:

1 Choose at least one option.

Default
NOCOMPAT

Abbreviations
CPAT(CASE, NOCASE, LIT, NOLIT, MC, NOMC, SYSL, NOSYSL,TRS,NOTRS) / NOCPAT

MC as a COMPAT abbreviation means “MACROCASE”, but for PCONTROL, means “MCALL”.

Parameter of ACONTROL statement
You can specify the COMPAT (or NOCOMPAT) option as a parameter of the ACONTROL statement. For further details, refer to the “ACONTROL instruction” in the *HLASM Language Reference*.

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.

NOCASE
Instructs the assembler to allow a mixed-case alphabetic character set.

LITTYPE
Instructs the assembler to return “U” as the type attribute for all literals.

NOLITTYPE
Instructs the assembler to provide the correct type attribute for literals once they have been defined.

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

Note: A quoted macro operand begins and ends with an apostrophe. Operands containing an apostrophe that do not begin and end with apostrophes are considered unquoted.

NOMACROCASE
Instructs the assembler not to convert (internally) lowercase alphabetic characters (*a* through *z*) in unquoted macro operands.

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.

NOSYSLIST
Instructs the assembler not to treat sublists in SETC symbols as character strings, when passed to a macro definition in an operand of a macro instruction.

TRANSDT
Instructs the assembler to extend use of the translation table, as specified by the TRANSLATE assembler option, to any C-type character Self-Defining Terms.

NOTRANSDT
Instructs the assembler not to translate any C-type character Self-Defining Terms.

NOCOMPAT
Instructs the assembler to allow lowercase alphabetic characters *a* through *z* in all language elements, to treat sublists in SETC symbols as sublists when passed to a macro definition in the operand of a macro instruction, and to provide the correct type attribute for literals once they have been defined.

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 apostrophes, 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 to the object data set. You define the object data set with the SYSPUNCH ddname on z/OS and CMS, or with the IJSYSPH file name and by assigning SYSPCH on z/VSE.

NODECK
Instructs the assembler not to write the object code to SYSPUNCH on z/OS and CMS, or SYSPCH on z/VSE.

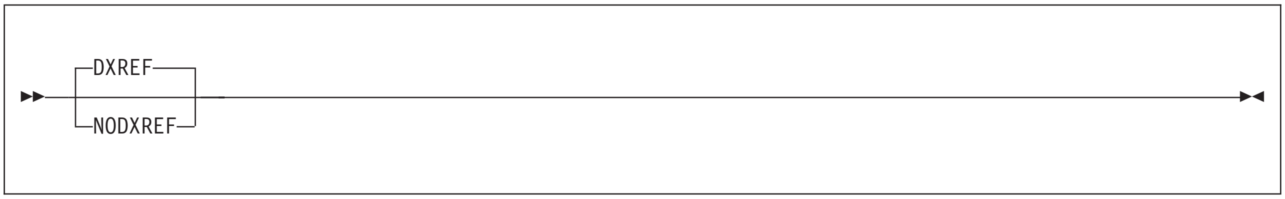
If you specify NODECK and NOOBJECT, the assembler ignores the EXIT(OBJEXIT)) option.

On z/VSE, you can only specify the DECK option on the JCL OPTION statement. If you specify it on the PARM operand of the JCL EXEC statement, the assembler issues message `ASMA400W`, and ignores the option.

DISK (CMS)

See "PRINT (CMS)" on page 63.

DXREF



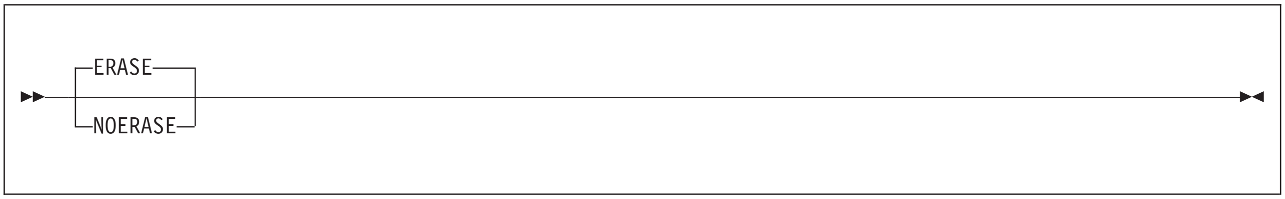
Default
DXREF

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 (CMS)



Default
ERASE

Abbreviations
None

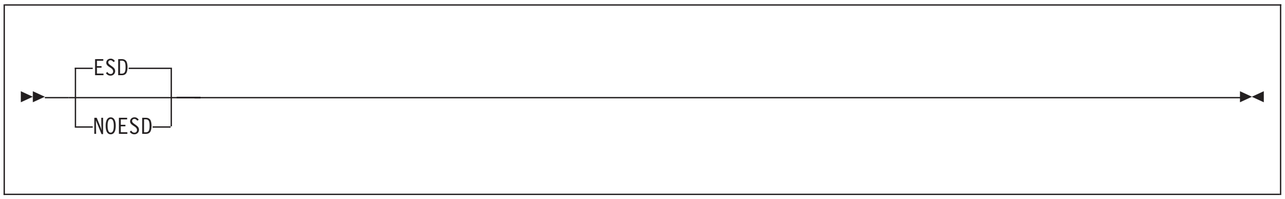
Restrictions
This option is not allowed on *PROCESS statements.

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

ERASE
Specifies that the existing files with a file name the same as the file name on the ASMAHL command, and a file type 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



Default
ESD

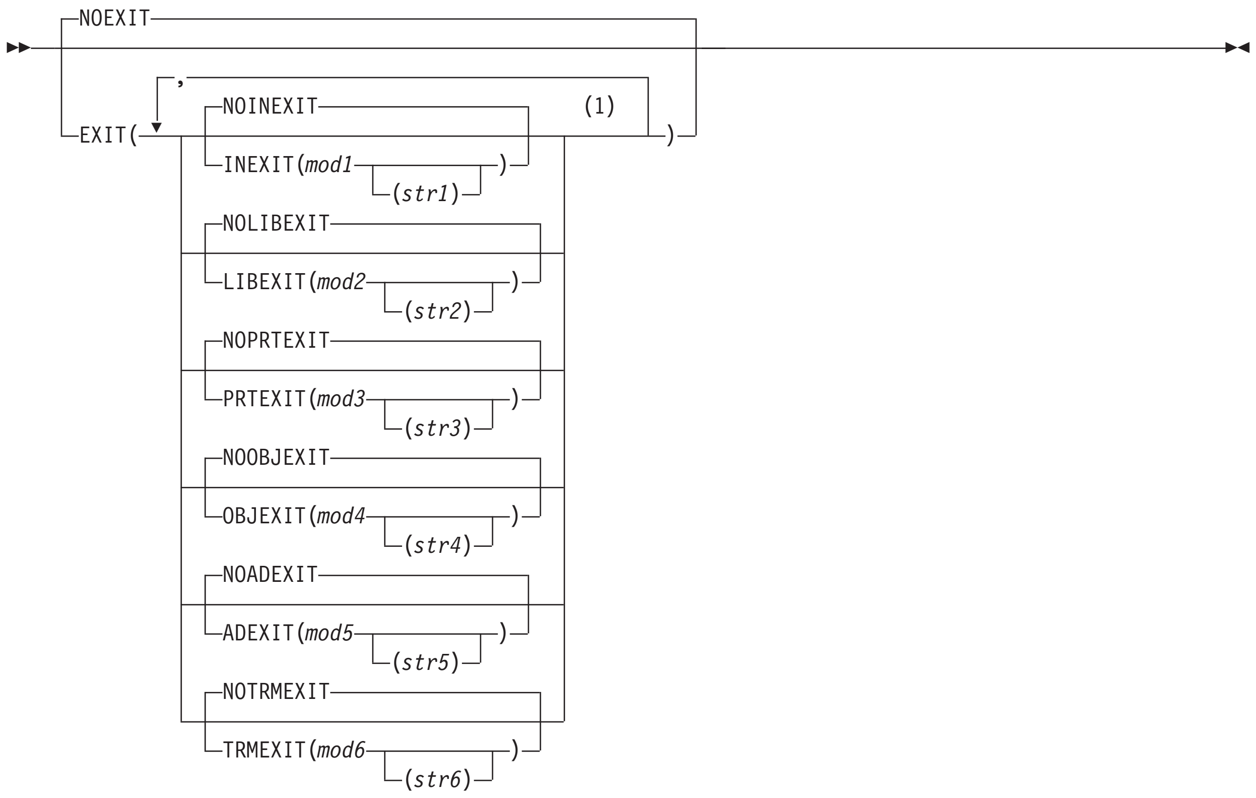
Abbreviations
None

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 z/OS binder, in the object module.

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

On z/VSE, you can specify this option on the JCL OPTION statement.

EXIT



Notes:

1 Choose at least one option.

Default
NOEXIT

Abbreviations
EX(INX, NOINX, LBX, NOLBX, PRX, NOPRX, OBX, NOOBX, ADX, NOADX, TRX, NOTRX) / 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, and provides a pointer to *str1* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79.

You can use a SOURCE exit, for example, to read variable-length source input records. See also Appendix J, “Sample SOURCE user exit (z/OS and CMS),” on page 363.

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, and provides a pointer to *str2* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79.

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

On z/VSE, you can use this exit to handle edited macros from the library sublibraries.

Refer to *z/VSE: Guide to System Functions* for a description of a LIBRARY exit to read edited macros.

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, and provides a pointer to *str3* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79.

You can use the LISTING exit, for example, to suppress parts of the assembly listing, or provide additional listing lines. See also Appendix I, “Sample LISTING user exit (z/OS and CMS),” on page 361.

NOPRTEXIT
Specifies that there is no LISTING exit.

OBJEXIT
On z/OS and CMS, 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 GOFF option, and provides a pointer to *str4* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79. 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 and NOOBJECT.

On z/VSE, specifies that the assembler use a PUNCH exit for the assembly. The name of the load module for the exit is *mod4*. The assembler passes control to the load module for PUNCH type exit processing when you specify the DECK option. You can use the PUNCH exit, for example, to catalog object modules directly into a library sublibrary.

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, and provides a pointer to *str5* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79. See also Appendix H, “Sample ADATA user exits (z/OS and CMS),” on page 351.

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, and provides a pointer to *str6* in a parameter list when the exit is first called. For a full description, see Chapter 4, “Providing user exits,” on page 79.

NOTRMEXIT
Specifies that there is no TERM exit.

NOEXIT
Specifies that there are no exits for the assembly.

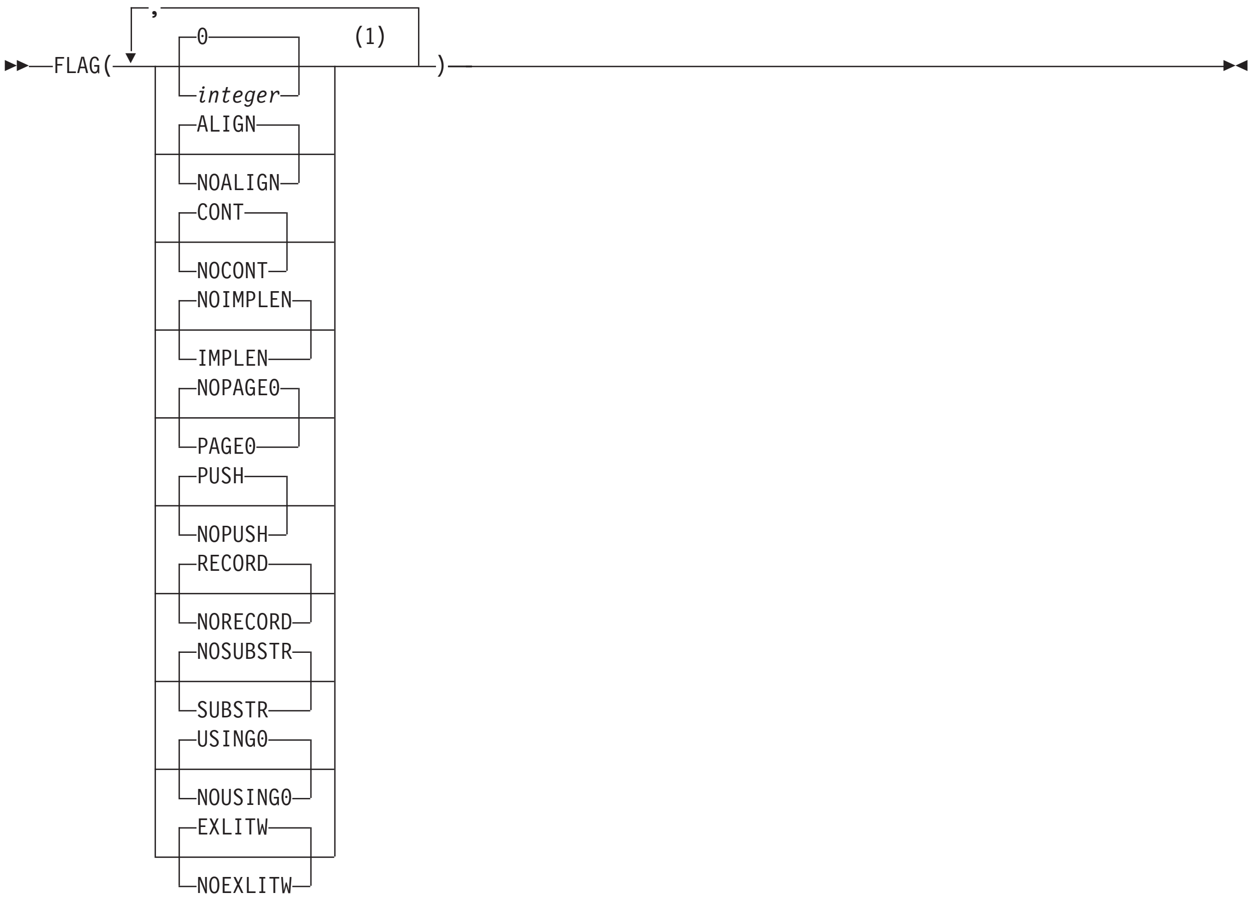
The module names *mod1*, *mod2*, *mod3*, *mod4*, *mod5*, and *mod6* can refer to the same load module.

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 can include any character in a string, but you must pair parentheses. JCL restrictions require that you specify two apostrophes to represent a single apostrophe, and two ampersands to represent a single ampersand.

For more information about the EXIT option, see Chapter 4, “Providing user exits,” on page 79.

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

FLAG



Notes:

1 Choose at least one option.

Default
FLAG(0, ALIGN, CONT, NOIMPLEN, NOPAGE0, PUSH, RECORD, NOSUBSTR, USING0, EXLITW)

Abbreviations
AL, NOAL, IMP, NOIMP, PG0, NOPG0, PU, NOPU, RC, NORC, SUB, NOSUB, US0, NOUS0

Parameter of ACONTROL statement
You can specify the FLAG option as a parameter of the ACONTROL statement. For further details, refer to the "ACONTROL instruction" in the *HLASM Language Reference*.

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 can 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.

ALIGN
Instructs the assembler to issue diagnostic message `ASMA033I` 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.

NOALIGN
Instructs the assembler not to issue diagnostic messages `ASMA033I`, `ASMA212W`, and `ASMA213W` when an inconsistency is detected between the operation code type and the alignment of addresses in machine instructions.

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 (normally column 16).
- A list of one or more operands ends with a comma, but the continuation indicator field (normally column 72) is a space.
- The continuation record starts in the continue column (normally 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 (normally column 16).

Note: FLAG(CONT) checks only apply to statements that appear in the output listing.

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

IMPLEN
Instructs the assembler to issue diagnostic message `ASMA169I` when an explicit length subfield is omitted from an SS-format machine instruction.

NOIMPLEN
Instructs the assembler not to issue diagnostic message `ASMA169I` when an explicit length subfield is omitted from an SS-format machine instruction.

PAGE0
Instructs the assembler to issue diagnostic message `ASMA309W` when an operand is resolved to a baseless address and a base and displacement is expected. This message is only issued for instructions that reference storage. For example, a LOAD instruction generates the message but a LOAD ADDRESS instruction does not generate the message.

NOPAGE0

Instructs the assembler not to issue diagnostic message `ASMA309W` when an operand is resolved to a baseless address and a base and displacement is expected.

The FLAG suboptions PAGE0 and IMPLEN are specified in the installation default options as PAGE0WARN and IMPLENWARN.

PUSH

Instructs the assembler to issue diagnostic warning message `ASMA138W` when a PUSH/POP stack is not empty at the completion of a compile.

NOPUSH

Instructs the assembler to suppress diagnostic warning message `ASMA138W` when a PUSH/POP stack is not empty at the completion of a compile.

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.

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.

SUBSTR

Instructs the assembler to issue warning diagnostic message `ASMA094I` when the second subscript value of the substring notation indexes past the end of the character expression.

NOSUBSTR

Instructs the assembler not to issue warning diagnostic message `ASMA094I` when the second subscript value of the substring notation indexes past the end of the character expression.

USING0

Instructs the assembler to issue diagnostic warning message `ASMA306W` for a USING that is coincident with or overlaps an implied USING 0,0, when the USING(WARN) suboption includes the condition numbers 1 and 4.

Note: Message ASMA302W is issued when R0 is specified as a base register with a non-zero base address, and message ASMA306W is issued when any register other than R0 is specified as a base register with an absolute base address whose range overlaps the assembler's default (0,4095).

NOUSING0

Instructs the assembler to suppress diagnostic warning message `ASMA306W`.

Note: Message ASMA302W is issued when R0 is specified as a base register with a non-zero base address, and message ASMA306W is issued when any register other than R0 is specified as a base register with an absolute base address whose range overlaps the assembler's default (0,4095).

EXLITW

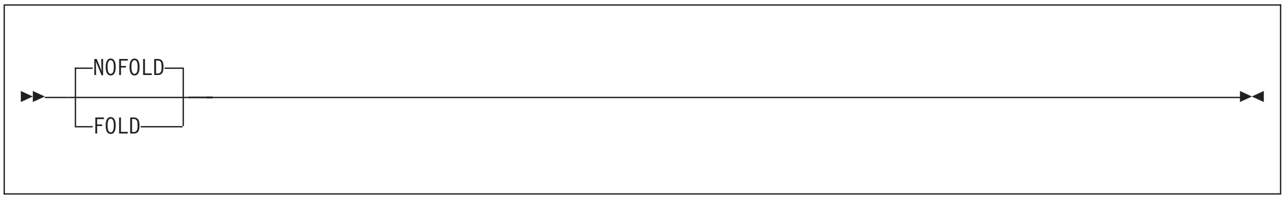
Instructs the assembler to issue diagnostic warning ASMA016W when a literal is used as the object of an EX instruction.

NOEXLITW
Instructs the assembler to suppress diagnostic warning message ASMA016W when a literal is used as the object of an EX instruction.

The FLAG suboptions ALIGN, CONT, IMPLEN, PAGE0, PUSH, RECORD, SUBSTR, USING0, and EXLITW are specified in the installation default options as ALIGNWARN, CONTWARN, IMPLENWARN, PAGE0WARN, PUSHWARN, RECORDINFO, SUBSTRWARN, USING0WARN, and EXLITW.

For information about installation default options, please refer to the *HLASM Installation and Customization Guide*.

FOLD



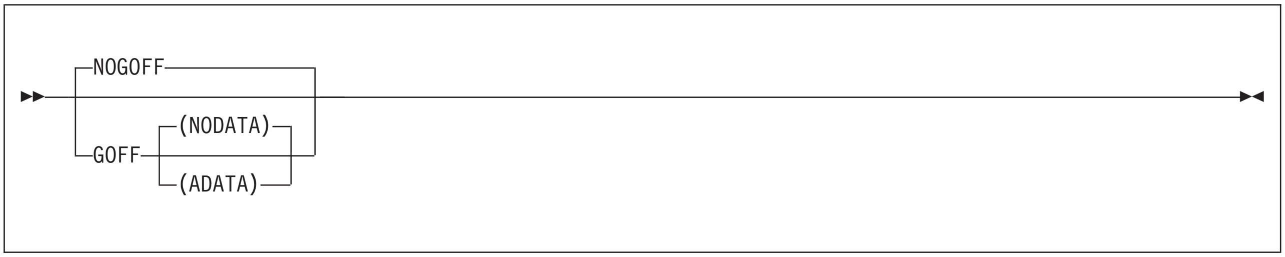
Default
NOFOLD

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, listing headings, 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. If the LANGUAGE option specifies a language other than English, it is possible that headings are not converted to uppercase.

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

GOFF (z/OS and CMS)



Default
NOGOFF

Abbreviations
None

Restrictions
You cannot specify this option on *PROCESS statements.

GOFF
Instructs the assembler to produce a Generalized Object File format (GOFF) data set. You define this data set with the SYSLIN or SYSPUNCH ddname.

Note: For more information about the GOFF format, refer to *z/OS DFSMS Program Management*.

NOADATA
The same as GOFF without a suboption.

ADATA
Instructs the assembler to produce a Generalized Object File format data set, and include ADATA text records.

NOGOFF
Instructs the assembler not to produce a Generalized Object File format data set.

Notes:

1. Specify the LIST(133) option when you specify the GOFF option. If the logical record length of the listing data set is less than 133, the assembler truncates the listing lines.
2. The generalized object format does not support TEST (SYM) records. If you specify the TEST option with the GOFF option, the assembler issues a diagnostic error message.
3. The assembler option XOBJECT is treated as a synonym for the GOFF option and accepts the same subparameters as GOFF.

For more information on the 133-character format, see “Source and object” on page 12.

When the GOFF option is not specified a control section is initiated or resumed by the CSECT, RSECT, and COM statements. Any machine language text created by statements that follow such control section declarations belongs to the control section, and is manipulated during program linking and binding as an indivisible unit.

When the GOFF option is specified, the behavior of statements like CSECT is different. By default, the assembler creates a definition of a text class named B_TEXT, to which subsequent machine language text belongs if no other classes are declared. If you specify other class names using the CATTR statement, machine language text following such CATTR statements belongs to that class.

The combination of a section name and a class name defines an *element*, which is the indivisible unit manipulated during linking and binding. All elements with the same section name are “owned” by that section, and binding actions (such as section replacement) act on all elements owned by a section.

When the GOFF option is specified, and if no CATTR statements are present, then all machine language text is placed in the default class B_TEXT, and the behavior of the elements in the bound module is essentially the same as the behavior of control sections when the OBJECT option is specified. However, if additional classes are declared, a section name can best be thought of as a "handle" by which elements within declared classes are owned.

INFO

```
       ┌─NOINFO───────────┐
►►─────┼──────────────────┼──────────────────────────────────────────────►◄
       ├─INFO(yyyymmdd)───┤
       └─INFO─────────────┘
```

Default
NOINFO

Abbreviations
None

INFO
Instructs the assembler to copy to the list data set all product information.

The Product Information Page (see Figure 23) follows the Option Summary,

INFO(*yyyymmdd*)
Instructs the assembler not to copy to the list data set any product information which is dated before *yyyymmdd*.

NOINFO
Instructs the assembler not to copy any product information to the list data set.

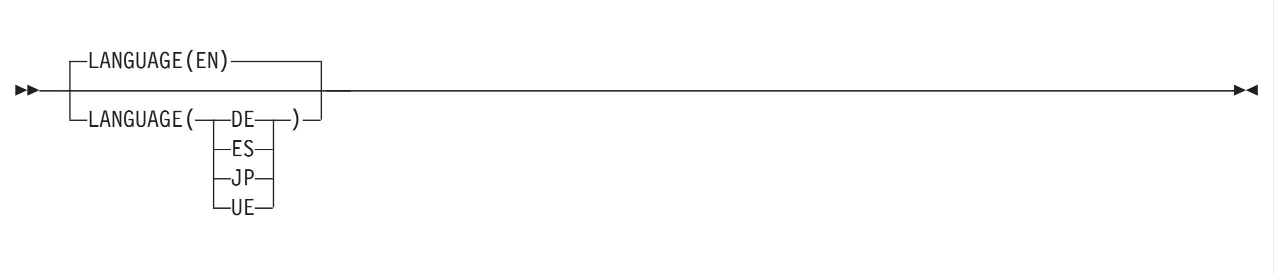
```
        High Level Assembler Product Information                                                    Page    2
                                                                                        HLASM R6.0  2008/07/11 17.48
          The following information describes enhancements and changes to the
          High Level Assembler Product.

          The information displayed can be managed by using the following options:
          INFO           - prints all available information for this release.
          INFO(yyyymmdd) - suppresses items dated prior to "yyyymmdd".
          NOINFO         - suppresses the product information entirely.

19981104  APAR PQ21028 Fixed
          Some machine opcodes incorrectly no longer accept literal operands.
```

Figure 23. High Level Assembler product information page

LANGUAGE



Default
LANGUAGE(EN)

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

Restrictions
This option is not allowed on *PROCESS statements.

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

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

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

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

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

Parameter of ACONTROL statement
You can specify the LIBMAC (or NOLIBMAC) option as a parameter of the ACONTROL statement. For further details, refer to the “ACONTROL instruction” in the *HLASM Language Reference*.

Format errors within a particular library macro definition are listed directly following the first call to that macro. Subsequent calls to the library macro do not result in this type of diagnostic. You can bring the macro definition into the source program with a COPY statement or by using the LIBMAC assembler option. The format errors then follow immediately after the statements in error.

LIBMAC
Specifies that, for each macro, macro definition statements read from a macro library are to be embedded 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

```
     ┌─LINECOUNT(60)──────┐
►►───┤                    ├────────────────────────►◄
     └─LINECOUNT(integer)─┘
```

Default
LINECOUNT(60)

Abbreviations
LC(*integer*)

CMS Only:

The LINECOUNT option can be abbreviated to LINECOUN.

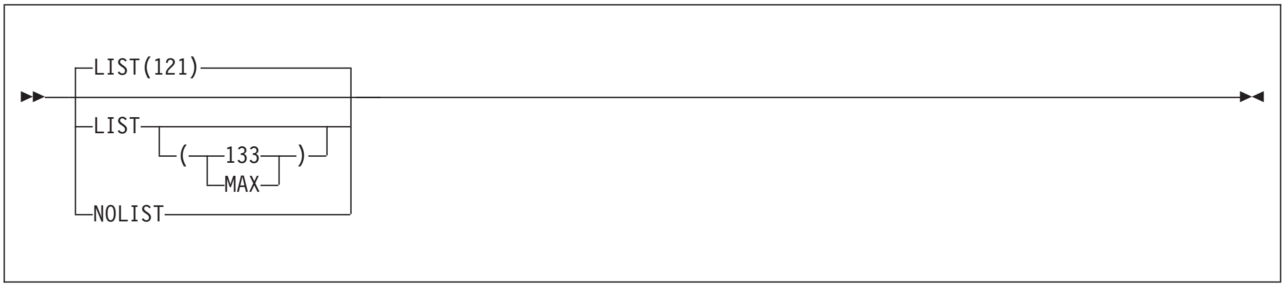
Restrictions
This option is not allowed on *PROCESS statements.

LINECOUNT(*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; there is only one eject character in the listing and that is at the top of the listing at the first line. From there on there are no headings, no ejects, and no titles. No eject characters are inserted at the start of sections of the listing (such as the ESD and RLD).

Up to seven lines on each page can be used for heading lines.

LIST



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 suboption is equivalent to specifying LIST(121).

121 (z/OS and CMS)
Instructs the assembler to produce a listing, and print the Source and Object section in the 121-character format.

133 (z/OS and CMS)
Instructs the assembler to produce a listing, and print the Source and Object section in the 133-character format. Use this option when you specify the GOFF option.

MAX (z/OS and CMS)
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             PCONTROL
ESD               RLD
EXIT(PRTEXIT)     RXREF
INFO              USING(MAP)
MXREF(MAP)        XREF
```

MACHINE

```
                       ┌─,──NOLIST─┐
►►──MACHINE(──┬─S370───────────┬─┼───────────┼─)───────────────────────────────►◄
              ├─S370XA─────────┤ └─,──LIST───┘
              ├─S370ESA────────┤
              ├─S390───────────┤
              ├─S390E──────────┤
              ├─ZSERIES────────┤
              ├─ZS─────────────┤
              ├─ZSERIES-2──────┤
              ├─ZS-2───────────┤
              ├─ZSERIES-3──────┤
              ├─ZS-3───────────┤
              ├─ZSERIES-4──────┤
              ├─ZS-4───────────┤
              ├─ZSERIES-5──────┤
              ├─ZS-5───────────┤
              ├─ZSERIES-6──────┤
              └─ZS-6───────────┘
```

Abbreviations
MAC

Restrictions
None

S370
Instructs the assembler to load and use the operation code table that contains the System/370 machine instructions, including those with a vector facility. Equivalent to OPTABLE(370).

S370XA
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. Equivalent to OPTABLE(XA).

S370ESA
Synonym for MACHINE(S390E).

S390
Synonym for MACHINE(S390E).

S390E
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. Equivalent to OPTABLE(ESA).

ZSERIES
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to OPTABLE(ZOP).

ZS Synonym for MACHINE(ZSERIES).

ZSERIES-2
Same as MACHINE(ZSERIES) but with the addition of the long displacement facility. Equivalent to OPTABLE(YOP).

ZS-2
Synonym for MACHINE(ZSERIES-2).

ZSERIES-3
Same as MACHINE(ZSERIES-2) but with the addition of support for the z9-109 instructions. Equivalent to OPTABLE(ZS3).

ZS-3
Synonym for MACHINE(ZSERIES-3).

ZSERIES-4
Same as MACHINE(ZSERIES-3) but with the addition of support for the z10 instructions. Equivalent to OPTABLE(ZS4).

ZS-4
Synonym for MACHINE(ZSERIES-4).

| **ZSERIES-5**
| Same as MACHINE(ZSERIES-4) but with the addition of support for the z196 instructions. Equivalent to OPTABLE(ZS5).

| **ZS-5**
| Synonym for MACHINE(ZSERIES-5).

| **ZSERIES-6**
| Same as MACHINE(ZSERIES-5) but with the addition of support for the zEC12 instructions.. Equivalent to OPTABLE(ZS6).

| **ZS-6**
| Synonym for MACHINE(ZSERIES-6).

LIST
Instructs the assembler to produce the Operation Code Table Contents section in the listing. Equivalent to OPTABLE(LIST).

NOLIST
Instructs the assembler not to produce the Operation Code Table Contents section in the listing. Equivalent to OPTABLE(NOLIST).

Notes:

1. The operation codes supported by High Level Assembler are described in the documents listed under "Bibliography" on page 399.
2. Table 8 shows the equivalent suboptions for the MACHINE and OPTABLE options.

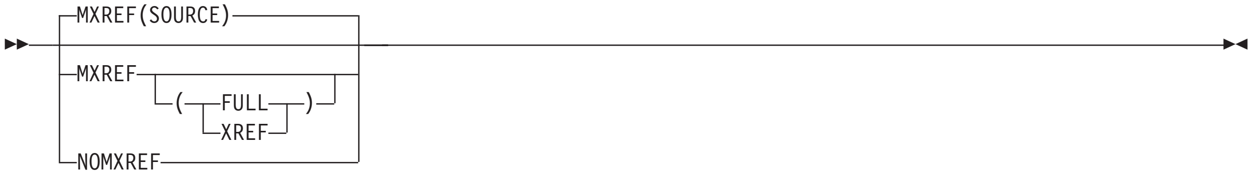
Table 8. Equivalent suboptions for MACHINE and OPTABLE options

MACHINE suboption	OPTABLE suboption	Output in assembler listing
	UNI	MACHINE(,NOLIST)
	DOS	MACHINE(,NOLIST)
S370	370	MACHINE(S370,NOLIST)
S370XA	XA	MACHINE(S370XA,NOLIST)
S370ESA	ESA	MACHINE(S390,NOLIST)
S390	ESA	MACHINE(S390,NOLIST)
S390E	ESA	MACHINE(S390,NOLIST)
ZSERIES	ZOP	MACHINE(ZSERIES,NOLIST)

Table 8. Equivalent suboptions for MACHINE and OPTABLE options (continued)

MACHINE suboption	OPTABLE suboption	Output in assembler listing
ZS	ZOP	MACHINE(ZSERIES,NOLIST)
ZSERIES-2	YOP	MACHINE(ZSERIES-2,NOLIST)
ZS-2	YOP	MACHINE(ZSERIES-2,NOLIST)
ZSERIES-3	ZS3	MACHINE(ZSERIES-3,NOLIST)
ZS-3	ZS3	MACHINE(ZSERIES-3,NOLIST)
ZSERIES-4	ZS4	MACHINE(ZSERIES-4,NOLIST)
ZS-4	ZS4	MACHINE(ZSERIES-4,NOLIST)
ZSERIES-5	ZS5	MACHINE(ZSERIES-5,NOLIST)
ZS-5	ZS5	MACHINE(ZSERIES-5,NOLIST)
ZSERIES-6	ZS6	MACHINE(ZSERIES-6,NOLIST)
ZS-6	ZS6	MACHINE(ZSERIES-6,NOLIST)

MXREF



Default
MXREF(SOURCE)

Abbreviations
MX / NOMX

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

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.

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.

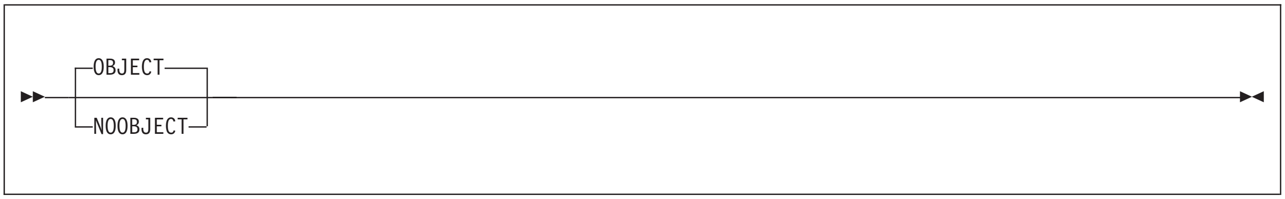
Note: See note following MXREF(XREF).

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.

Note: If you specify MXREF(FULL) or MXREF(XREF), you might need to review the value of the SIZE option (as both of these assembler options use considerable amounts of storage).

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

OBJECT



Default
OBJECT

Abbreviations
OBJ / NOOBJ

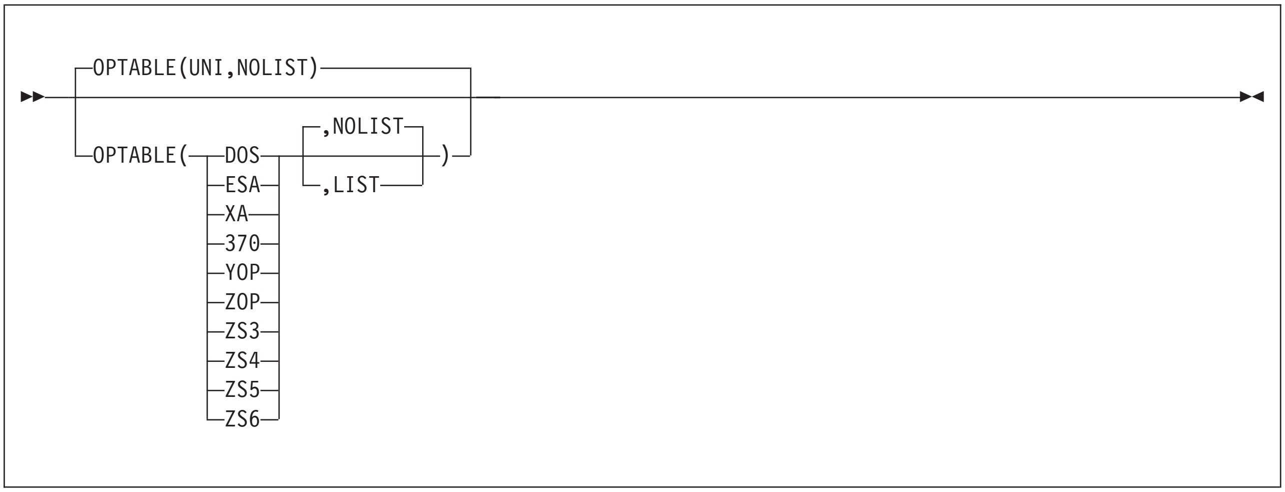
Restrictions
You cannot specify this option on *PROCESS statements.

OBJECT
Instructs the assembler to generate object code and write it to the object data set. You define the object data set with the SYSLIN ddname, on z/OS and CMS, or with the IJSYSLN file name and by assigning SYSLNK on z/VSE.

NOOBJECT
Instructs the assembler not to write the object code to SYSLIN, on z/OS and CMS, or SYSLNK on z/VSE.

On z/VSE, you can only specify the OBJECT option by using the LINK or CATAL option on the JCL OPTION statement. If you specify OBJECT on the PARM operand of the JCL EXEC statement, the assembler issues message `ASMA400W`, and ignores the option.

OPTABLE



Default
OPTABLE(UNI,NOLIST)

Abbreviation
OP

Restrictions
None

DOS
Instructs the assembler to load and use the DOS operation code table. The DOS operation code is designed for assembling programs previously assembled using the DOS/VSE assembler. The operation code table contains the System/370 machine instructions, excluding those with a vector facility.

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. Equivalent to MACHINE(S390E).

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, and Z/Architecture machine instructions.

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. Equivalent to MACHINE(S370XA).

370
Instructs the assembler to load and use the operation code table that contains the System/370 machine instructions, including those with a vector facility. Equivalent to MACHINE(S370).

YOP
Same as OPTABLE(ZOP) but with the addition of the long displacement facility. Equivalent to MACHINE(ZSERIES-2).

ZOP
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to MACHINE(ZSERIES).

ZS3
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to MACHINE(ZSERIES-3).

ZS4
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to MACHINE(ZSERIES-4).

ZS5
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to MACHINE(ZSERIES-5).

ZS6
Instructs the assembler to load and use the operation code table that contains the symbolic operation codes for the machine instructions specific to z/Architecture systems. Equivalent to MACHINE(ZSERIES-6).

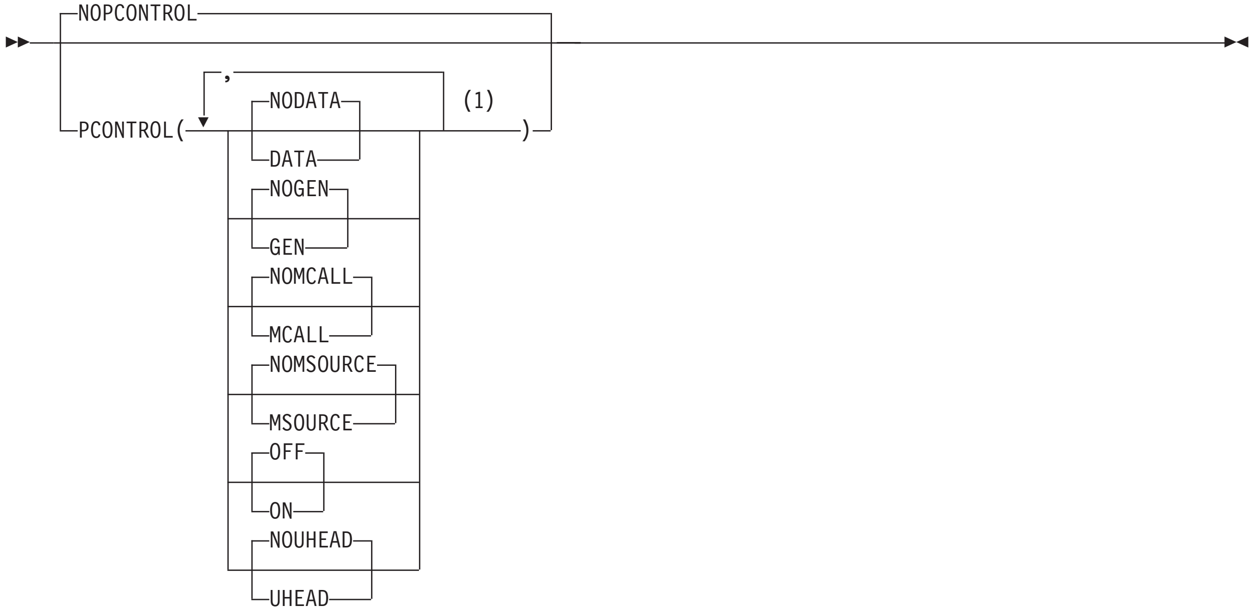
LIST
Instructs the assembler to produce the Operation Code Table Contents section in the listing. Equivalent to MACHINE(LIST).

NOLIST
Instructs the assembler not to produce the Operation Code Table Contents section in the listing. Equivalent to MACHINE(NOLIST).

Note:

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 documents listed under "Bibliography" on page 399.
3. Table 8 on page 57 shows the equivalent suboptions for the OPTABLE and MACHINE options.

PCONTROL



Notes:

1 Choose at least one option.

Default
NOPCONTROL

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

MC as a PCONTROL abbreviation means "MCALL", but for COMPAT, means "MACROCASE".

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.

NOADATA
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.

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.

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.

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.

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.

MSOURCE
Specifies that the assembler is to print the source statements generated during conditional assembly substitution or 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. PRINT GEN statements or the PCONTROL(GEN) option must be active for this option to have effect.

NOMSOURCE
Instructs the assembler to suppress the printing of source statements generated during conditional assembly substitution or 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. PRINT GEN statements or the PCONTROL(GEN) option must be active for this option to have effect.

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.

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.

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.

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.

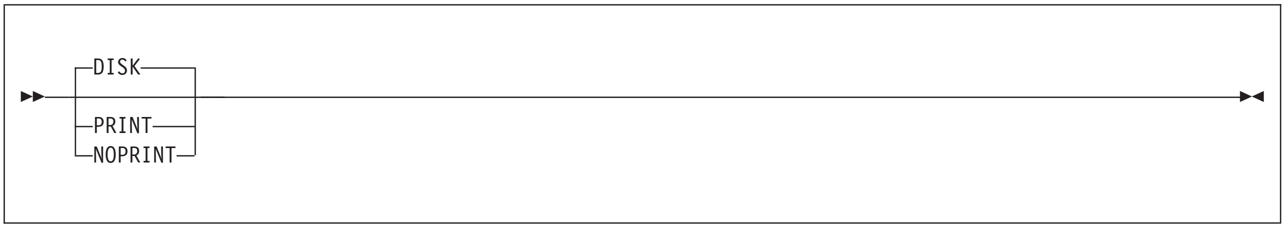
NOPCONTROL
Specifies that the assembler honor all PRINT statements in the source program. The standard PRINT operands active at the beginning of an assembly are ON, GEN, NODATA, NOMCALL, MSOURCE, and UHEAD.

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 “PESTOP” in the *HLASM Installation and Customization Guide* for instructions how to specify this option.

PRINT (CMS)



Default
DISK

Abbreviations
PR / NOPR / DI

Restrictions
This option is not allowed on *PROCESS statements.

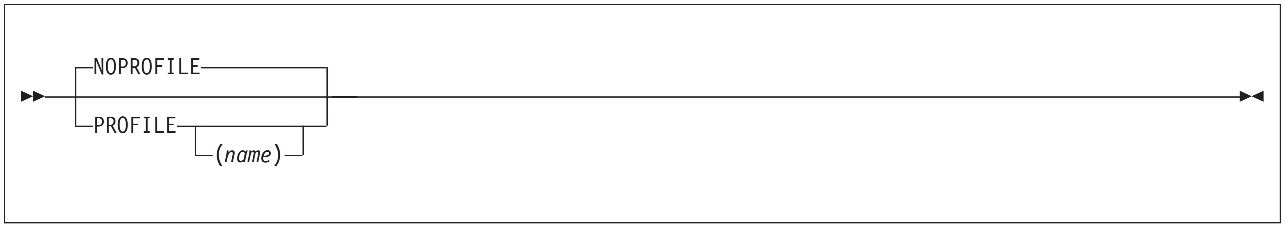
This option can only be specified when you use the ASMAHL command on CMS.

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.

PROFILE



Default
NOPROFILE

Abbreviations
PROF / NOPROF

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

The assembler generates a COPY instruction for the specified member after it has processed any ICTL and *PROCESS instructions at the beginning of the program. Because this COPY instruction ends the scan for these special instructions, the profile cannot contain ICTL or *PROCESS statements.

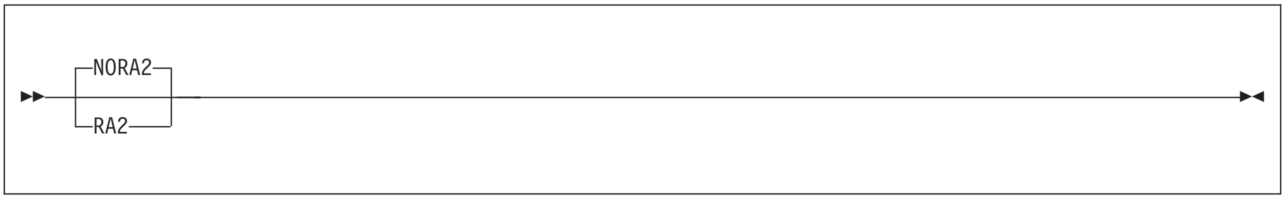
name
Instructs the assembler to copy the member *name* into the source program, as if the source program contained a COPY instruction (see “COPY instruction” in the *HLASM Language Reference*).

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 the last *PROCESS statement.
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 “ASMAOPT” in the *HLASM Installation and Customization Guide* for instructions how to use the ASMAOPT macro.
3. On z/OS and CMS, the assembler searches for the member in the macro and copy code libraries defined in the SYSLIB DD statement.
4. On z/VSE, the assembler searches for the member in the macro and copy code libraries defined in the LIBDEF job control statement.
5. 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 “COPY instruction” in the *HLASM Language Reference* for further information about the COPY instruction.

RA2



Default
NORA2

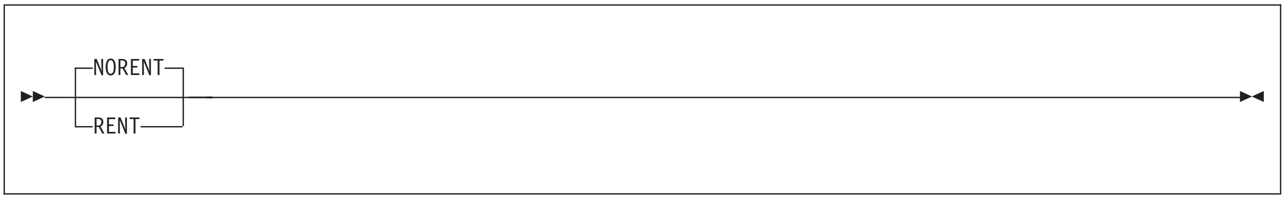
Abbreviations
None

Parameter of ACONTROL statement
You can specify the RA2 (or NORA2) option as a parameter of the ACONTROL statement. For further details, refer to “ACONTROL instruction” in the *HLASM Language Reference*.

RA2
Instructs the assembler to suppress error diagnostic message `ASMA066W` 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 `ASMA066W` when 2-byte relocatable address constants, such as AL2(*) and Y(*), are defined in the source program.

RENT



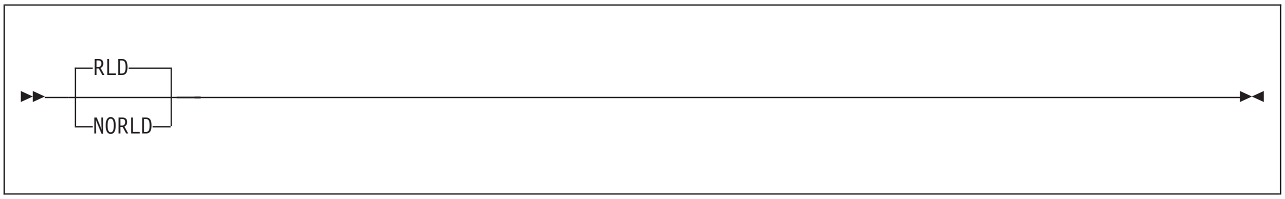
Default
NORENT

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



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 z/OS binder, in the object module.

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

On z/VSE, you can specify the RLD option on the JCL OPTION statement.

RXREF

```
   ┌─RXREF───┐
►►─┤         ├──────────────────────────────────────────────────────────────►◄
   └─NORXREF─┘
```

Default

RXREF

Abbreviations

RX / NORX

RXREF

Instructs the assembler to produce the General Purpose Register Cross Reference section of the assembler listing. The General Purpose Register Cross Reference includes:
- The register number
- An indicator showing the context in which the register was used.

NORXREF

Instructs the assembler not to produce the General Purpose Register Cross Reference section of the assembler listing.

SECTALGN

```
    ┌─SECTALGN(8)─────────┐
►►──┴─SECTALGN(alignment)─┴─────────────────────────────────────────────────►◄
```

Default

8 (doubleword alignment)

Abbreviations

None

Restrictions

The GOFF option must be specified if the *alignment* value is greater than 8.

SECTALGN(*alignment*)

Specifies the alignment for all sections. The alignment must be a power of 2 between 8 (doubleword) and 4096 (page).

SEG (CMS)

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

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

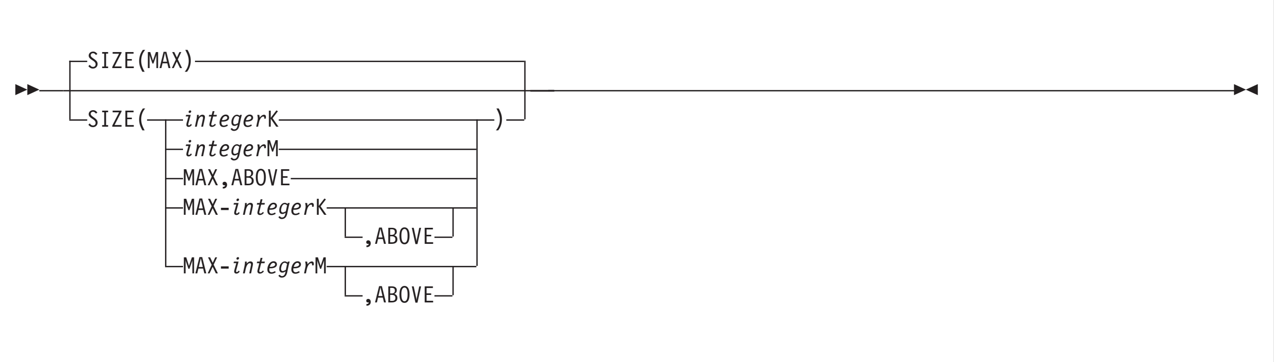
Restrictions
You cannot specify this option on *PROCESS statements.

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

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

NOSEG
Specifies that the assembler modules can be loaded from disk.

SIZE



You use the SIZE option to specify the amount of virtual storage available to the assembler to perform an in-storage assembly.

Default
SIZE(MAX)

Abbreviation
SZ

Restrictions
You cannot specify this option on *PROCESS statements.

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

The minimum acceptable value is 200 KB (refer to Note 2 on page 68).

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

The minimum acceptable value is 1 MB.

MAX
Specifies that the assembler requests all the available space (refer to Note 3 on page 68) in the user region (z/OS), or virtual machine (CMS) or in the partition GETVIS (z/VSE).

MAX-*integer*K

Specifies that the assembler requests all the available space (refer to Note 3) in the user region (z/OS), virtual machine (CMS) or partition GETVIS (z/VSE) less the amount of *integer*K of storage (1 KB equals 1024 bytes).

The minimum acceptable integerK value is 1 KB.

MAX-*integer*M

Specifies that the assembler requests all the available space (refer to Note 3) in the user region (z/OS), or virtual machine (CMS) less the amount of *integer*K of storage (1 MB equals 1048756 bytes).

The minimum acceptable integerM value is 1 MB.

Note:

1. The maximum storage value you can specify might not be available in the user region (z/OS), virtual machine (CMS), or in the partition GETVIS (z/VSE), after storage has been allocated by the operating system for I/O buffers, and so on.
2. The minimum amount of working storage required by the assembler is 200 KB or 10 times the work data set block size, whichever is greater.
3. When you specify the MAX suboption, the assembler releases 128 KB back to the user region (z/OS), virtual machine (CMS), or the partition GETVIS (z/VSE), for system usage. When you specify the MAX suboption, there might not be enough storage remaining in the user region (z/OS), virtual machine (CMS), or the partition GETVIS (z/VSE), to load any exits you specify, or any external functions you use in your assembly or for operating system functions that might be required during the assembly. If the assembly does fail due to a lack of storage, then use MAX-*integer*K or MAX-*integer*M to ensure that there is sufficient free storage during the assembly for any of these functions.
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.
6. With High Level Assembler R6, the ABOVE operand is obsolete. It is documented here for compatibility. The assembler ignores this operand and always attempts to obtain the storage specified by the SIZE option from above the 16 MB line. If storage above the 16 MB line is not available, the assembler attempts to obtain the storage from below the 16 MB line.

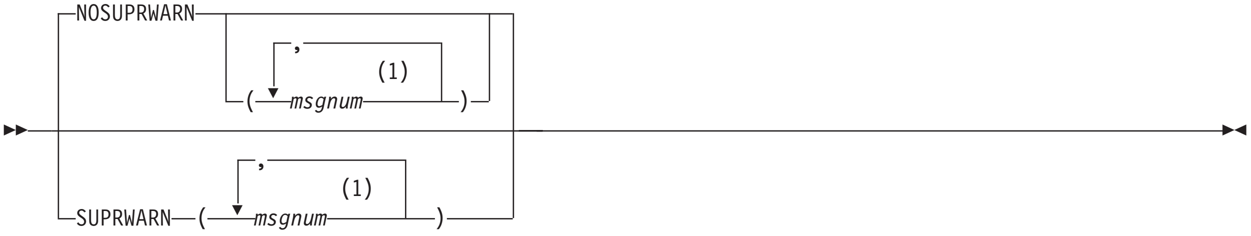
High Level Assembler acquires the amount of storage you specify in the SIZE option from the user region (z/OS), virtual machine (CMS), or partition GETVIS (z/VSE). The assembler only requires a work data set when it has insufficient virtual storage to perform an in-storage assembly. An in-storage assembly normally 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 allocated and the amount of storage the assembler used. Specify a large enough value on the SIZE option to allow the assembler to perform an in-storage assembly, otherwise you must provide a work data set.

To reduce the amount of storage required by the assembly you can disable some of the cross-reference sections of the assembler listing, for example you can specify NOMXREF, NORXREF, or NOXREF. Additionally if ADATA is not required then it should be disabled by specifying NOADATA. Enabling of ADATA processing causes most of the cross-reference processing to be done, even though the sections are not output to the assembler listing.

SUPRWARN

SUPRWARN



Notes:

1 Specify at least one message number.

Default
NOSUPRWARN

Abbreviations
SUP, NOSUP

Restrictions
You can only suppress messages of severity 4 or less.

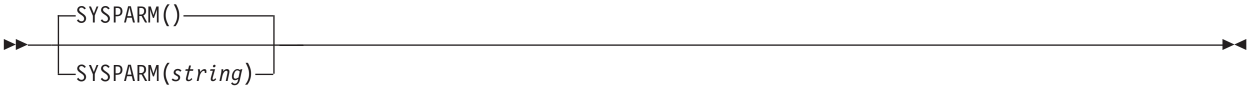
SUPRWARN
Suppresses the specified message numbers.

NOSUPRWARN
Reverses the action of SUPRWARN and allows the specified message numbers to be displayed.

msgnum
1–4 digit message number.

If you specify an invalid message number, message ASMA318W is issued. If you specify a message number with a severity higher than 4, message ASMA319W is issued.

SYSPARM



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 can be included in the string, subject to the rules for building character strings defined in the *HLASM Language Reference*.

On a z/OS or z/VSE invocation parameter, if the string includes spaces, commas, or parentheses, it must be enclosed in apostrophes. Any parentheses inside the string must be balanced. You must use two apostrophes to represent a single apostrophe, and two ampersands to represent a single ampersand. For example:

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

assigns the value `(&AM,'EO).FY` to &SYSPARM.

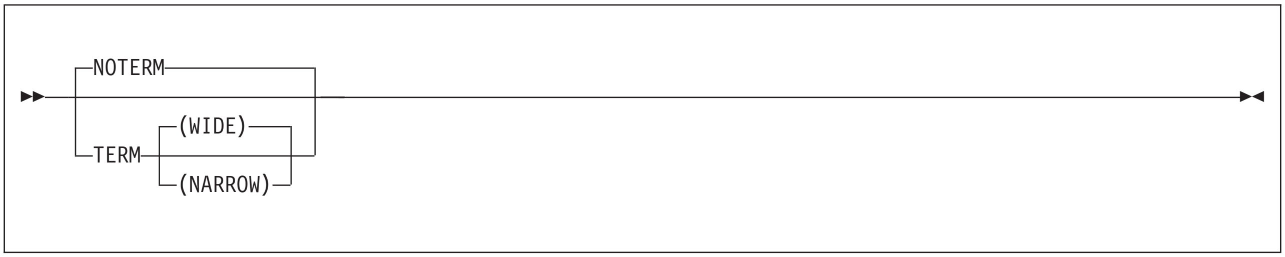
on z/OS and z/VSE, JCL restrictions limit the length of the SYSPARM value. When you call the assembler from a problem program (dynamic invocation), you can specify a SYSPARM value up to 255 characters long. The ASMAOPT file ("ASMAOPT options" on page 36) also supports a SYSPARM value up to 255 characters long.

On 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.

TERM



Default
NOTERM

Abbreviations
None

Restrictions
This option is not allowed on *PROCESS statements.

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

WIDE
On z/OS and CMS, instructs the assembler to write error messages to the terminal data set. You define the terminal data set with the SYSTERM ddname.

On z/VSE, instructs the assembler to write error messages to SYSLOG. SYSLOG is normally assigned at system initialization time and is assigned permanently to the system log (console).

NARROW
The NARROW suboption instructs the assembler to compress multiple consecutive spaces into a single space.

On z/OS and CMS, instructs the assembler to write error messages to the terminal data set. You define the terminal data set with the SYSTERM ddname.

On z/VSE, instructs the assembler to write error messages to SYSLOG. SYSLOG is normally assigned at system initialization time and is assigned permanently to the system log (console).

NOTERM
Instructs the assembler not to write error messages to SYSTERM (z/OS and CMS) or SYSLOG (z/VSE).

You can specify the TERM option on the JCL OPTION statement.

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.

On z/OS and CMS, if you specify the TEST option with the GOFF option, the assembler ignores the TEST option.

THREAD

```
     ┌─THREAD───┐
▸▸───┼──────────┼────────────────────────────────────────────────────────────▸◂
     └─NOTHREAD─┘
```

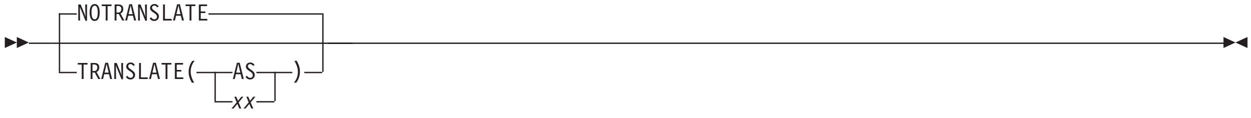
Default
THREAD

Abbreviations
THR / NOTHR

THREAD
Specifies that the assembler not reset the location counter to zero at the beginning of each CSECT.

NOTHREAD
Specifies that the assembler reset the location counter to zero at the beginning of each CSECT, except for the first CSECT when it is initiated by the START instruction having a nonzero operand.

TRANSLATE



Default
NOTRANSLATE

Abbreviations
TR / NOTR

Restrictions
This option is not allowed on *PROCESS statements.

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.

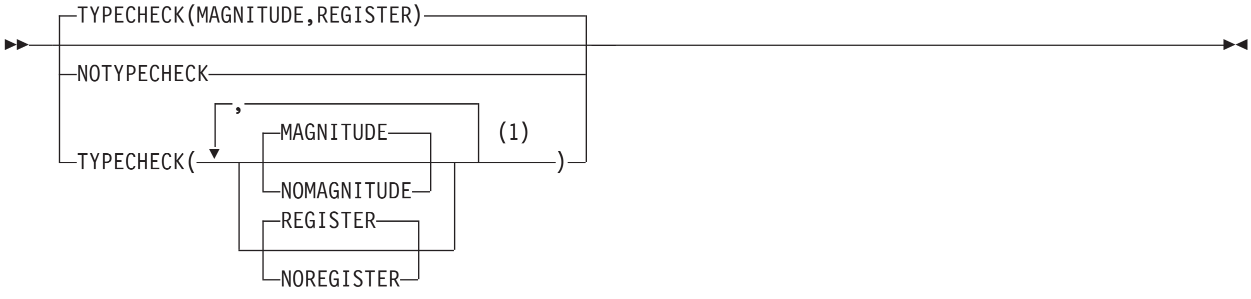
xx Specifies that characters contained in character (C-type) data constants (DCs) and literals are converted using a user-supplied translation table. The translation table must be named ASMALT*xx*.

Notes:

1. Using the TRANSLATE option when you have also specified the DBCS option can cause erroneous translation of double-byte data in C-type constants. G-type constant data is not affected.
2. The assembler searches for the user-supplied translation table load module in the standard load module search order. See also Appendix K, "How to generate a translation table," on page 365. The assembler does not convert UNICODE character strings when the TRANSLATE option is specified (see "CODEPAGE" on page 41.)
3. The assembler does not convert UNICODE character strings.

TYPECHECK

For a detailed description of the TYPECHECK assembler option, see Appendix M, "TYPECHECK assembler option," on page 373.



Notes:

1 Choose at least one option.

Default
TYPECHECK(MAGNITUDE,REGISTER)

Abbreviations
TC(MAG, NOMAG, REG, NOREG) / NOTC

Parameter of ACONTROL statement
You can specify the TYPECHECK (or NOTYPECHECK) option as a parameter of the ACONTROL statement. For further details, refer to “ACONTROL statement” in the *HLASM Language Reference*.

MAGNITUDE
Specifies that the assembler performs magnitude validation of signed immediate-data fields of machine instruction operands.

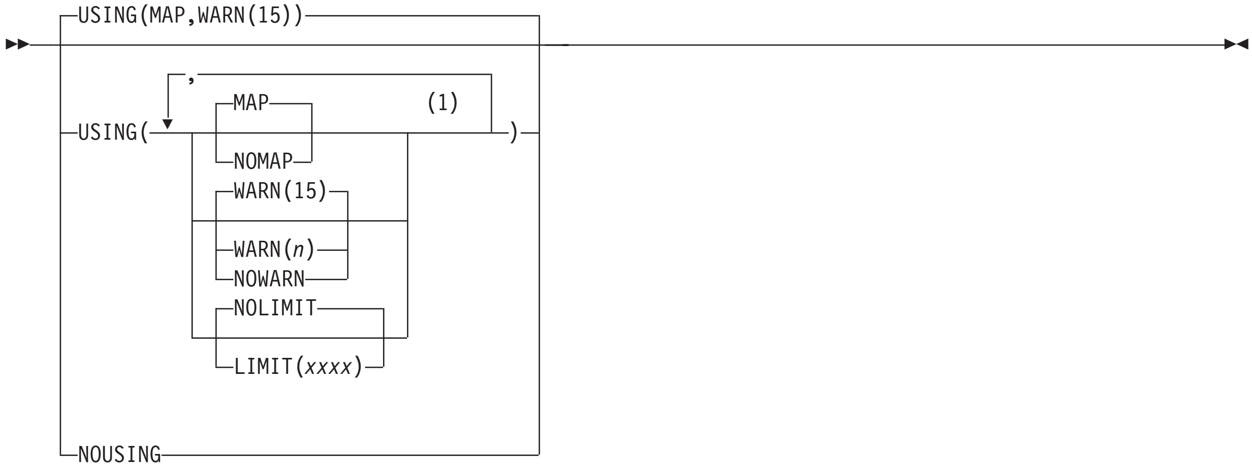
NOMAGNITUDE
Specifies that the assembler not perform magnitude validation of signed immediate-data fields of machine instruction operands.

REGISTER
Specifies that the assembler performs type checking of register fields of machine instruction operands.

NOREGISTER
Specifies that the assembler not perform type checking of register fields of machine instruction operands.

NOTYPECHECK
Specifies that the assembler not perform any type checking of machine instruction operands.

USING



Notes:

1 Choose at least one option.

Default
USING(MAP,WARN(15))

Abbreviations
US / NOUS

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' can 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)) is specified at invocation, the messages is issued as in Figure 24.

```
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
000000                00000 01F82     1 EXAMPLE  CSECT                                                          00001000
                R:AB  00000           2          USING EXAMPLE,10,11                                            00002000

⋮
00040E 47F0 AB8C            00B8C   176          B     LABEL111                  1                              00176000

⋮
                      00B8C         496 LABEL111 EQU   *                                                        00496000

⋮
00152E 47F0 BF80            01F80   908          B     LABEL999                  2                              00908000
** ASMA304W Displacement exceeds LIMIT value specified

⋮
                      01F80        1436 LABEL999 EQU   *                                                        01436000
001F80 07FE                        1437          BR    14                                                       01437000
                                   1438          END                                                            01438000
```

Figure 24. Effect of the LIMIT suboption

Although the resolved displacement of the instruction at **1** is greater than the specified limit, error diagnostic message `ASMA304W` 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 27.

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:

0 No USING warning messages are issued.

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

- `ASMA300W` when a previous active ordinary (unlabeled) USING's range coincides with and supersedes that of the USING being processed.
- `ASMA301W` when the range of the USING being processed coincides with and supersedes that of a previous active ordinary (unlabeled) USING.
- `ASMA306W` when the range of the USING being processed coincides with the implicit USING 0,0 (for example USING 0,2).

Note: Message ASMA302W is issued when R0 is specified as a base register with a non-zero base address, and message ASMA306W is issued when any register other than R0 is specified as a base register with an absolute base address whose range overlaps the assembler's default (0,4095).

2 **R0 based USINGs**: The assembler issues message `ASMA302W` 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:

- `ASMA303W` when multiple resolutions are possible for an implicit address.
- `ASMA306W` when the range of the USING being processed overlaps the range of the implicit USING 0,0 (for example USING 16,2).

Note: Message ASMA302W is issued when R0 is specified as a base register with a non-zero base address, and message ASMA306W is issued when any register other than R0 is specified as a base register with an absolute base address whose range overlaps the assembler's default (0,4095).

8 **LIMIT**: The assembler issues message `ASMA304W` 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.

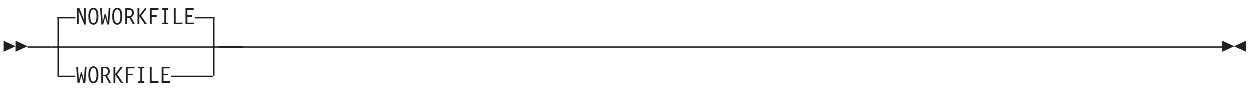
Several conditions can be combined by adding the associated condition numbers. For example, specifying WARN(12) tells the assembler to issue warning diagnostic messages for the conditions with condition numbers 4 and 8.

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

NOUSING
NOUSING specifies that all USING suboptions are off. It is equivalent to specifying USING(NOLIMIT,NOMAP,NOWARN), or specifying in the ASMADOPT default options: LIMIT=NO,MAP=NO,WARN=NO.

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

WORKFILE



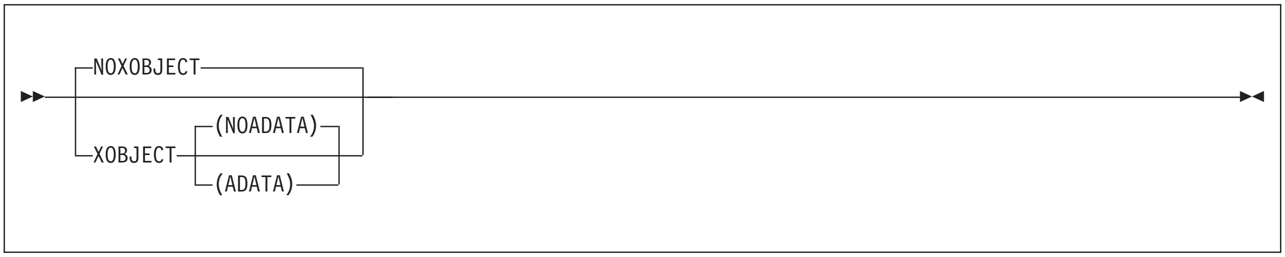
Default
NOWORKFILE

Abbreviations
None

WORKFILE
If the WORKFILE option is specified, the assembler attempts to assemble the program in central storage. If extra storage is needed, the assembler uses the utility file for temporary storage.

NOWORKFILE
If the NOWORKFILE option is specified, the assembly uses only central storage.

XOBJECT (z/OS and CMS)

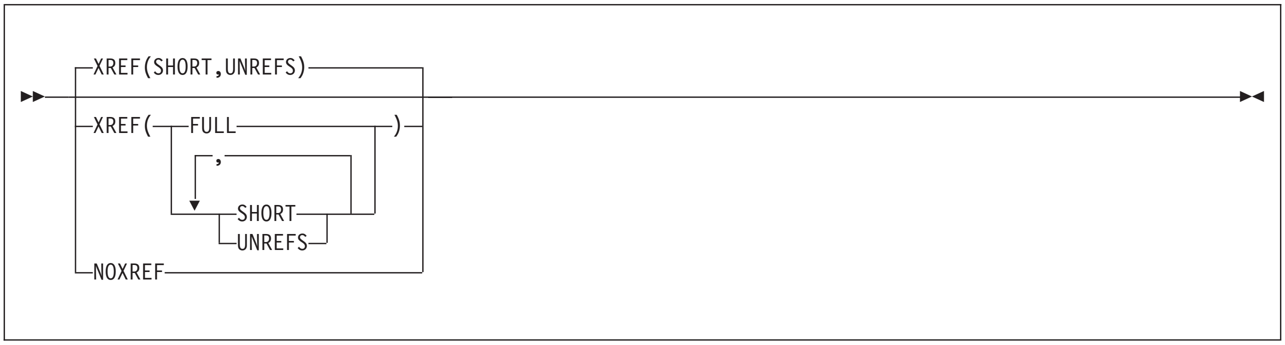


Default
NOXOBJECT

Abbreviations
XOBJ / NOXOBJ

Note: XOBJECT is treated as a synonym for the GOFF option and accepts the same subparameters as GOFF.

XREF



Default
XREF(SHORT,UNREFS)

Abbreviations
None

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.

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 can be specified with the UNREFS suboption to produce a list of unreferenced symbols. The SHORT suboption cannot be specified with the FULL suboption.

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 can be specified with the SHORT suboption to produce a cross reference list of referenced symbols. The UNREFS suboption cannot 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)`.

On z/VSE, you can use the XREF option of the JCL OPTION statement to specify the XREF(FULL) assembler option, and the SXREF option of the JCL OPTION statement to specify the XREF(SHORT) assembler option.

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 these 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 (z/OS and CMS) 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 replace or complement the assembler's associated data output processing. You can use it to write the associated data records the assembler supplies, or monitor and modify the records before the assembler writes them to the associated data set. The exit can write all the associated data records, or supply additional records for the assembler to write during the assembly. The exit can also discard records.

TERM Exit

To process Terminal Output records.

© Copyright IBM Corp. 1992, 2013

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. 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.

z/VSE The assembler writes terminal output to SYSLOG; however, you can use the exit to write the output to a disk data set.

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. See “EXITCTL*n*” on page 86.

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 46 for the syntax of the EXIT assembler option.

These tables lists the exit type, the EXIT suboption, the default data set *ddname* (z/OS and CMS), or *filename* (z/VSE), that the exit corresponds to, and a Page number reference to the section that describes how the assembler processes the exit:

Table 9. z/OS and CMS exit types

Exit Type	Exit Suboption	*ddname*	Page Number
SOURCE	INEXIT	SYSIN	“SOURCE exit processing” on page 95
LIBRARY	LIBEXIT	SYSLIB	“LIBRARY exit processing” on page 98
LISTING	PRTEXIT	SYSPRINT	“LISTING exit processing” on page 103
PUNCH	OBJEXIT	SYSPUNCH	“OBJECT (z/OS and CMS) and PUNCH exit processing” on page 107
OBJECT	OBJEXIT	SYSLIN	“OBJECT (z/OS and CMS) and PUNCH exit processing” on page 107
ADATA	ADEXIT	SYSADATA	“ADATA exit processing” on page 110
TERM	TRMEXIT	SYSTERM	“TERM exit processing” on page 112

Table 10. z/VSE exit types

Exit Type	Exit Suboption	*filename*	Page Number
SOURCE	INEXIT	IJSYSIN	“SOURCE exit processing” on page 95
LIBRARY	LIBEXIT	Library sublibraries	“LIBRARY exit processing” on page 98
LISTING	PRTEXIT	IJSYSLS	“LISTING exit processing” on page 103

Table 10. z/VSE exit types (continued)

Exit Type	**Exit Suboption**	*filename*	**Page Number**
PUNCH	OBJEXIT	IJSYSPH	"OBJECT (z/OS and CMS) and PUNCH exit processing" on page 107
ADATA	ADEXIT	SYSADAT	"ADATA exit processing" on page 110
TERM	TRMEXIT	SYSLOG	"TERM exit processing" on page 112

Loading user exits

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

z/OS

The user exit must be a load module or program object 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).

An exit module can contain more than one entry name if the initially loaded module executes an IDENTIFY macro to make additional names available to the assembler as exit names.

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.

z/VM

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.

z/VSE

The user exit must be a relocatable phase that is in a sublibrary you specify in your JCL LIBDEF phase search chain. The user exit can also be located in the Shared Virtual Area (SVA).

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

The user exits can be created 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 these constraints:
- Can be called many times, using the exit module entry point
- Retains storage for private variables across invocations

High Level Assembler provides an "anchor word" in the Exit Parameter List to allow you to maintain information across calls to the exit.

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 25 on page 83.

Registers 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 block that contains information for each exit type. High Level Assembler provides macro ASMAXITP to map the Exit Parameter List and the Exit-Specific Information block. Figure 25 on page 83 describes the format of the Exit Parameter List, Figure 26 on page 93 describes the format of the Exit-Specific Information block for the LISTING exit, and Figure 27 on page 93 describes the format of the Exit-Specific Information block for the other exit types.

The high-order bit of the last pointer in the I/O exit parameter list is set to 1.

The HLASM Services Interface for I/O exits is described Appendix N, “HLASM Services Interface,” on page 383.

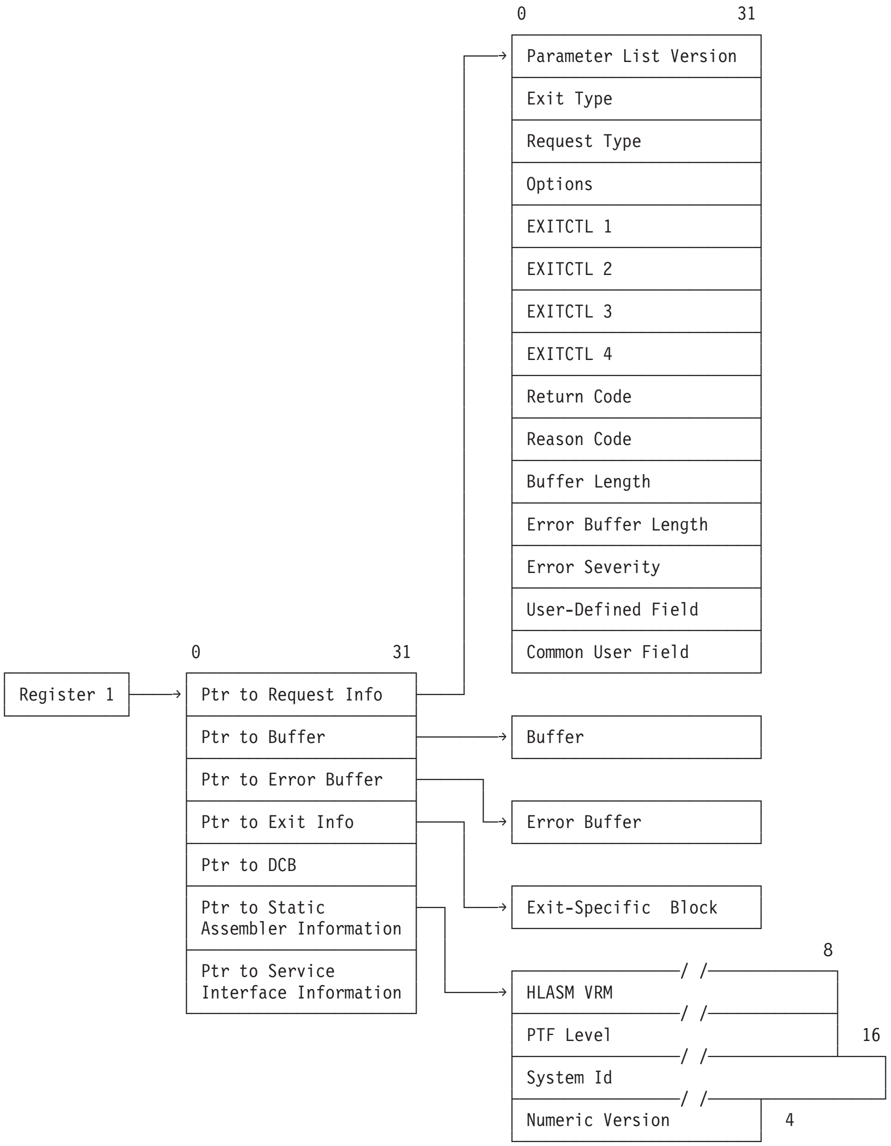


Figure 25. Exit parameter list format

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 6 this field contains a value of 3.

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 these values:

1 OPEN—exit receives control before any input or output processing.

The exits are opened in this order:
1. SOURCE
2. LIBRARY
3. LISTING
4. TERM
5. OBJECT/PUNCH
6. ADATA

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

The exits are closed in this order:
1. SOURCE
2. LIBRARY
3. OBJECT/PUNCH
4. ADATA
5. TERM
6. LISTING

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 (LIBRARY exit type)—exit receives control to inspect or manipulate the macro definition record provided by the assembler.

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

7 FIND MACRO (LIBRARY exit type)—exit receives control to locate the specified library macro.

8 FIND COPY MEMBER (LIBRARY exit type)—exit receives control to locate the specified copy member.

9 END OF MEMBER (LIBRARY exit type)—exit receives control after the reading of a macro or copy member is completed.

10 REINIT—exit receives control during the assembler's reinitialization processing between assemblies in a batch. This call only occurs if the exit requested the REINIT by setting reason code 16 in response to the OPEN request.

The assembler always sets this field.

Options

A fullword that provides additional information to the exit.

For the SOURCE and LIBRARY exits: These 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, the copy code or a macro should resume at the saved record position.
3	For the LIBRARY exit, when the request type is FIND MACRO or FIND COPY, copy code or a 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 101.

For the LISTING exit: These 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
17	High Level Assembler Product Information heading line
18	High Level Assembler Product Information detail line
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
85	General Purpose Register Cross Reference heading line
86	General Purpose Register Cross Reference detail line
90	Diagnostic Cross Reference and Assembler Summary heading line
91	Diagnostic Cross Reference and Assembler Summary detail line

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

The assembler sets this field.

EXITCTL*n*

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 after 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 passing to the exit 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. Table 11 summarizes the return codes.

Return code 20 is described at "Error handling" on page 92.

Table 11. User-exit return codes

Exit	Request	RC=0	4	8	16	20[7]
SOURCE	OPEN	Assembler to open the primary input data set [1]	Exit provides records [2]		Disable [6]	Operation failed
	CLOSE	Operation successful				Operation failed
	READ	Exit has provided record			End-of-file indicator	Operation failed
	PROCESS	Accept record	Discard record		Disable [6]	Operation failed
	REINIT	Operation successful			Disable [6]	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	Disable [6]	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		Disable [5, 6]	Operation failed

Table 11. User-exit return codes (continued)

Exit	Request	RC=0	4	8	16	20[7]
	FIND (macro or copy member)	Operation successful	Member not found; search assembler library if available			Operation failed
	END OF MEMBER	Operation successful			Disable [5, 6]	Operation failed
	REINIT	Operation successful			Disable [6]	Operation failed
LISTING PUNCH OBJECT(z/OS and CMS) TERM ADATA	OPEN	Assembler opens the output data set [1]	Exit has opened its output data set [4]		Disable [6]	Operation failed
	CLOSE	Operation successful				Operation failed
	WRITE	Exit has written record				Operation failed
	PROCESS	Accept record	Discard record		Disable [6]	Operation failed
	REINIT	Operation successful			Disable [6]	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. This return is valid from all PROCESS and END OF MEMBER requests, with these exceptions:
   a. PROCESS MACRO requests when the LIBRARY exit set the return code of 8 for the OPEN request.
   b. PROCESS COPY requests when the LIBRARY exit set the return code of 8 for the OPEN request.
   c. END OF MEMBER requests when the LIBRARY exit set the return code of 4 or 8 for the OPEN request.
6. If an exit sets the disable return code, then the assembler does not call that exit again, so do any any resource clean-up before returning control to the assembler.
7. See “Error handling” on page 92.

Reason code

A fullword, set by the exit, to qualify the return code. Table 12 shows reason codes for each exit type, and which request they are checked after.

Table 12. User exit reason codes

Exit	Request	RSN=0	4	8	16
SOURCE	OPEN	No additional information	Input source information available		REINIT call required
	READ	No additional information	Input source information available	Redrive requested [2]	

Table 12. User exit reason codes (continued)

Exit	Request	RSN=0	4	8	16
LIBRARY	OPEN	No additional information	End of member call required		REINIT call required
	FIND (macro or copy member)	No additional information	Input source information available	Redrive requested [2]	
	READ	No additional information	Input source information available	Redrive requested [2]	
LISTING TERM	OPEN	No additional information	When return code is 0, reason code 4 indicates the exit has provided a line length in the buffer length field. When return code is 4, reason code 4 indicates the exit has provided the data set information.		REINIT call required
SOURCE LIBRARY LISTING PUNCH OBJECT(z/OS and CMS) TERM ADATA	PROCESS	No additional information	Return to exit with empty buffer	Redrive requested [2]	
LISTING PUNCH OBJECT(z/OS and CMS) TERM ADATA	WRITE	No additional information		Redrive requested [2]	
PUNCH OBJECT(z/OS and CMS)	OPEN	No additional information	Exit has provided the output data set information		REINIT call required
ADATA	OPEN	No additional information	Exit has provided the output data set information	Exit intends to discard type X'0002' and X'0090' records	REINIT call required

Notes:

1. Multiple reason codes can be specified by OR-ing them together.
2. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

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 optional 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 can 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.

On z/OS and CMS, if you specify the GOFF assembler option, and the OBJECT exit is invoked, the buffer length might be fixed-length 80, or variable-length, depending on the JCL (z/OS) 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.

z/VM and z/OS
The line length must be in the range 121 to 255. If it is any other value, the assembler issues message `ASMA402W` and does not call the exit to process listing records.

z/VSE If the assembler opens the listing data set and the LISTING exit provides a print line length, the line length must be 121 if SYSLST is assigned to disk, otherwise it can be any value from 121 to 133. If it is any other value, the assembler issues message `ASMA402W` 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 indication is 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, or negative, and must not be greater than 255 on z/OS and CMS, or 125 on z/VSE. If the value is not correct, the assembler issues message `ASMA404W` 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 `ASMA700I` to `ASMA704C`. The assembler selects which message to issue by checking the value of the error severity field.

The contents of the error buffer are ignored on CLOSE calls to the exit, unless the exit requests immediate termination. If immediate termination is requested, the assembler generates an ASMA940U message with the text provided by the exit; all remaining open exits are CLOSEd, but might not provide any additional message text.

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 is meant to have 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 permitted 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 the five diagnostic messages, `ASMA700I` through `ASMA704C`. For example, severity code of 4 causes the assembler to issue message `ASMA701W`. Table 13 summarizes the return code values and the associated diagnostic message.

Table 13. Error severity and associated diagnostic message

Error Severity Code Specified	Error Severity Code Used	Associated Message
0	0	ASMA700I
1–4	4	ASMA701W
5–8	8	ASMA702E
9–12	12	ASMA703S
> 12	16	ASMA704C

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.

Common user field

A fullword, set to zero by the assembler. Any exit can use this to store information (such as the address of acquired storage) between calls. This field is common for all exit types and so provides a mechanism for different exits to share information. The assembler does not modify or interpret this field.

Buffer pointer

The buffer pointer points to 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 spaces 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.

Note: The buffer pointer might not be the same for all calls to the exit, even if the exit is the same type for all calls.

Error buffer pointer

The error buffer pointer points to 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. For more details, see "Exit-Specific Information Block" on page 92.

DCB pointer

z/VSE This field is a fullword and always contains zeros. It is included to maintain compatibility with the Exit Parameter List in the z/VM and z/OS environments.

z/VM and z/OS
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 might use the DCB to obtain additional information about the data set or member being used. For example, on z/OS, the exit can obtain user information from a PDS or PDSE directory by using the BLDL system macro.

Static assembler information pointer

The Static Assembler Information Pointer is a fullword that contains the address of the Static Assembler Information block. The block is not aligned on any particular boundary.

HLASM VRM

This is an 8 byte field which contains the version, release, and modification level of the assembler being used to assemble the source module. For example, when HLASM Release 5.0 is being used, this field has the value "1.5.0". The value of the field is set to the value of the system variable &SYSVER.

PTF level

This is an 8 byte field which contains the current PTF level of the assembler being used to assemble the source module. For example, when the last PTF applied to the assembler was UQ12345 this field has the value "UQ12345".

System ID

This is a 16 byte field which contains the name and release of the operating system under which your source module is being assembled. For example, on an MVS™ system, the field might contain the value "MVS/ESA SP 5.1.0". The value of the field is set to the value of the system variable &SYSTEM_ID.

Numeric version

A fullword containing a numeric value indicating the version and release of the assembler being used. For example, for High Level Assembler Release 5, this field is set to 5.

HLASM Services Interface pointer

HLASM Services Interface pointer is a fullword that contains the address of the HLASM Services Interface block. For more information see Appendix N, “HLASM Services Interface,” on page 383.

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 `ASMA940U`, 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 Table 11 on page 86, the assembler issues message `ASMA940U` and stops the assembly.

Note: For CLOSE requests, the assembler only checks for a return code of 20; it treats all other values as a successful completion.

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.

On entry to the exit, the Member Name field contains spaces (unless the data set is a SYSLIB data set), the Data Set Name field contains the path name, and the Volume Serial field contains spaces.

The Exit-Specific Information block passed to all exits, except the LISTING exit, is shown in Figure 27 on page 93. The Exit-Specific Information block passed to the LISTING exit has additional parameters as shown in Figure 26 on page 93.

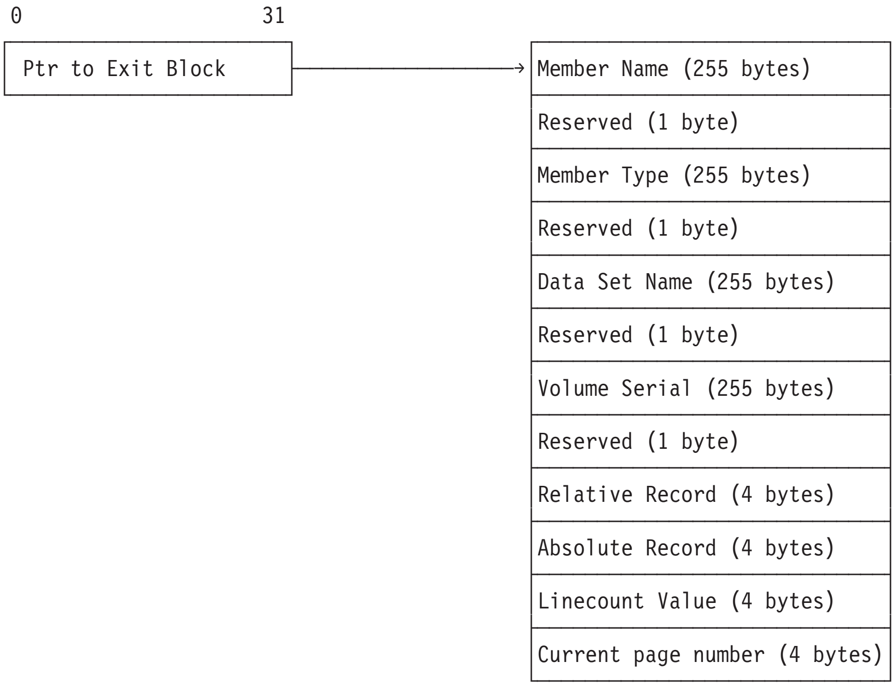

0
31
Ptr to Exit Block
Member Name (255 bytes)
Reserved (1 byte)
Member Type (255 bytes)
Reserved (1 byte)
Data Set Name (255 bytes)
Reserved (1 byte)
Volume Serial (255 bytes)
Reserved (1 byte)
Relative Record (4 bytes)
Absolute Record (4 bytes)
Linecount Value (4 bytes)
Current page number (4 bytes)

Figure 26. Exit-specific information block—LISTING exit

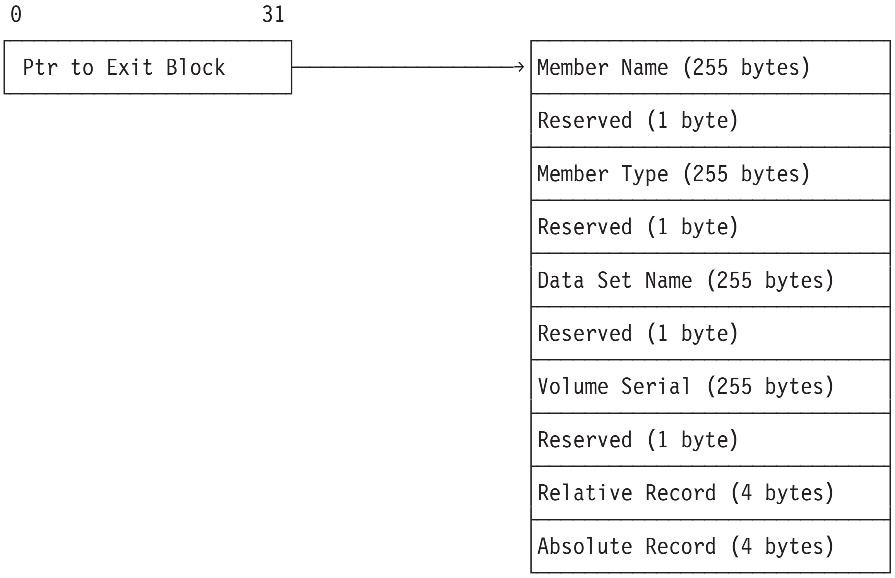


Figure 27. Exit-specific information block—other exit types

The Exit-Specific Information block consists of these fields:

Member Name

Member name within the data set. It is always provided for library members and is also provided for data set members used sequentially on z/OS 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-aligned and padded with spaces.

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 Table 14 (z/OS and CMS) and Table 15 (z/VSE).

Member Type

z/VM and z/OS
Always blank. This field is present to maintain compatibility with High Level Assembler running on z/VSE.

z/VSE The file type of the member. This field is also set by the assembler as a parameter for the FIND operation. It is left-aligned and padded with spaces.

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-aligned and padded with spaces.

z/VSE For library data sets, the name includes the library and sublibrary name.

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 Table 14 (z/OS and CMS) and Table 15 (z/VSE).

Volume Serial
Volume serial where the data set is located. It is left-aligned and padded with spaces.

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 Table 14 (z/OS and CMS) and Table 15 (z/VSE).

Table 14. z/OS and CMS 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

Table 15. z/VSE system variable symbols

Data Set	Member Name	Data Set Name	Volume Serial
SYSIPT (IJSYSIN)	&SYSIN_MEMBER	&SYSIN_DSN	&SYSIN_VOLUME
Library	&SYSLIB_MEMBER	&SYSLIB_DSN	&SYSLIB_VOLUME
SYSLST (IJSYSLS)	&SYSPRINT_MEMBER	&SYSPRINT_DSN	&SYSPRINT_VOLUME
SYSLOG	&SYSTERM_MEMBER	&SYSTERM_DSN	&SYSTERM_VOLUME
SYSPCH (IJSYSPH)	&SYSPUNCH_MEMBER	&SYSPUNCH_DSN	&SYSPUNCH_VOLUME
SYSADAT	&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 before 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 might 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 these 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 sets the return code in the Exit Parameter List to one of these values:

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

z/VM and z/OS
The buffer length field can be set to any value from 121 to 255. If the listing data set has a variable-length record format, the LRECL assigned is 4 bytes greater than the value the exit returns. If the value is less than 121 or greater than 255, the assembler issues message ASMA402W and does not call the exit for any further processing.

z/VSE The buffer length field can be set to any value from 121 to 133. If the value is less than 121 or greater than 133, the assembler issues message ASMA402W and does not call the exit for any further processing.

If you assign SYSLST to a disk data set in your JCL, the record length must be 121.

Note: A reason code of 16 indicates a REINIT call is required.

4 Indicates that the exit supplies the primary input records to the assembler in later READ calls. If you want 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.

Note: A reason code of 16 indicates a REINIT call is required.

16 Instructs the assembler to open the primary input data set, and make no further calls to the exit.

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 sets the return code in the Exit Parameter List to one of these values:

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 want 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 section of the listing, nor are they included in the ADATA Job Identification record.

16 Indicates to the assembler that there are no more records. This indication 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.

z/VM and z/OS
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 assembler 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 in “For the LISTING exit” on page 85.

The exit sets the return code in the Exit Parameter List to one of these values:

0 Indicates that the record has been accepted, and the assembler is to process it. The exit might modify the record before it returns control to the assembler. The user exit might 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.

Note: A reason code of 8 requests redrive of the exit.

z/VSE z/OS
The exit must also ensure that a valid printer control character is placed in the first character of the record. The printer control character is 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.

z/VSE The exit must also ensure that a valid American National Standard printer control character is placed in the first character of the record.

4 Instructs the assembler to discard the current record.

16 Instructs the assembler to make no further calls to the exit.

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 (see “Options” on page 85), 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.

Table 16 on page 98 summarizes the SOURCE exit processing.

Table 16. 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. If reason code=16, REINIT call required.
	16	Assembler opens primary input, and makes no further calls to the exit.
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. If reason code=8, redrive requested. [1]
	16	Exit indicates end-of-file.
5=PROCESS	0	Record accepted. Exit might modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.
10=REINIT	0	Operation successful.
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

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 call is the first call to the exit.

The exit sets the return code in the Exit Parameter List to one of these values:

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.

Note: A reason code of 4 indicates that the exit expects to receive END OF MEMBER calls.

A reason code of 16 indicates a REINIT call is required.

4 Indicates that the exit supplies the macro and copy code library records to the assembler in later READ calls. If you want 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.

Note: A reason code of 4 indicates that the exit expects to receive END OF MEMBER calls.

A reason code of 16 indicates a REINIT call is required.

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.

Note: A reason code of 4 indicates that the exit expects to receive END OF MEMBER calls.

A reason code of 16 indicates a REINIT call is required.

16 Instructs the assembler to open the library data set, and make no further calls to the exit.

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 sets the return code in the Exit Parameter List to one of these values:

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.

Note: A reason code of 8 requests redrive of the exit.

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

Note: A reason code of 8 requests redrive of the exit.

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 sets the return code in the Exit Parameter List to one of these values:

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.

Note: A reason code of 8 requests redrive of the exit.

4 Instructs the assembler to discard the current record.

Note: A reason code of 8 requests redrive of the exit.

16 Instructs the assembler to make no further calls to the exit. This is disregarded by the assembler if the exit return code from the OPEN was 8.

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 and the exit issued a return code on OPEN of either 4 or 8. 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 and the exit issued a return code on OPEN of either 4 or 8. The member name field contains the name of the copy code member.

If the user exit is supplying the library records, the exit sets the return code in the Exit Parameter List to one of these values:

0 Indicates that the exit supplies the library records. The assembler calls the user exit with later READ calls to retrieve each record.

Note: A reason code of 8 requests redrive of the exit.

4 Indicates that the exit is not supplying the macro or copy member, and is equivalent to not finding the member in the library.

Note: A reason code of 8 requests redrive of the exit.

If both the assembler and the user exit are supplying the library records, the exit sets the return code in the Exit Parameter List to one of these values:

0 Indicates that the exit supplies the library records. The assembler calls the user exit with later READ calls to retrieve each record.

Note: A reason code of 8 requests redrive of the exit.

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.

Note: A reason code of 8 requests redrive of the exit.

System variables

If you want 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 section of the listing, 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. 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.

END OF MEMBER

The assembler calls the exit with a request type of 9 (END OF MEMBER) whenever the reading of a macro or copy member is completed. For a macro, processing of a MEND statement indicates completion; for a copy member, an end of file condition indicates completion.

The END OF MEMBER call simplifies stack management required in coding a LIBRARY exit which contains READs and FINDs. The exit might use the information provided by this call in the handling of nested FINDs where there is typically a corresponding resume FIND (options=2) for every nested FIND (options=3). For an example of how you can use END OF MEMBER calls to perform stack management, see the code example `Use End of Member calls to perform stack management` in "TERM exit—TRMEXIT" on page 116.

Note: A reason code of 8 requests redrive of the exit.

Table 17 summarizes the LIBRARY exit processing.

Table 17. LIBRARY exit processing summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens its library for input. If reason code=4, the assembler makes END OF MEMBER calls to the exit. If reason code=16, REINIT call required.
	4	Exit supplies library records. If reason code=4, the assembler makes END OF MEMBER calls to the exit. If reason code=16, REINIT call required.
	8	Both the assembler and the exit supply library records. The assembler opens its library. If reason code=4, the assembler makes END OF MEMBER calls to the exit. If reason code=16, REINIT call required.
	16	Assembler opens the library data set, and makes no further calls to the EXIT.
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. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the EXIT (disregarded if the EXIT return code from the OPEN is 8).

Table 17. LIBRARY exit processing summary (continued)

Request Value=Type	Exit Return Code	Action
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. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the EXIT (disregarded if the EXIT return code from the OPEN is 8).
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=2, the exit should restore the previous member position. If reason code=4, exit supplies data set information. If reason code=8, redrive requested. [1]
	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=2, the exit should restore the previous member position. If reason code=4, exit supplies data set information. If reason code=8, redrive requested. [1]
	4	Copy code member not found by exit; the exit does not supply the records.
9=END OF MEMBER		Exit might use the information to perform stack management. If reason code=8, redrive requested. [1]
10=REINIT	0	Operation successful. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

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 normally writes them.

The LISTING exit is not called if you specify the NOLIST assembler option. If you want 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 these actions:

- Instruct the assembler to discard the listing records by setting the exit return code.
- Suppress the listing output by doing this action:
  z/OS Provide a `//SYSPRINT DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSPRINT DUMMY` command.
  z/VSE Assign the `SYSLST` to `IGN`.
- Instruct the exit to issue an OPEN return code of 4.

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.

z/VM and z/OS

Although the assembler can write to a listing data set with a record format of variable-length, the exit is always presented with fixed-length records.

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 sets the return code in the Exit Parameter List to one of these values:

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.

Note: A reason code of 16 indicates a REINIT call is required.

z/VM and z/OS

The buffer length field can be set to any value from 121 to 255. If the listing data set has a variable-length record format, the LRECL assigned is 4 bytes greater than the value the exit returns. If the value is less than 121 or greater than 255, the assembler issues message ASMA402W and does not call the exit for any further processing.

z/VSE

The buffer length field can be set to any value from 121 to 133. If the value is less than 121 or greater than 133, the assembler issues message ASMA402W and does not call the exit for any further processing.

If you assign SYSLST to a disk data set in your JCL, the record length must be 121.

The buffer length field can be set to any value from 121 to 255. If the listing data set has a variable-length record format, the LRECL assigned is 4 bytes greater than the value the exit returns. If the value is less than 121 or greater than 255, the assembler issues message `ASMA402W` and does not call the exit for any further processing.

The buffer length field can be set to any value from 121 to 133. If the value is less than 121 or greater than 133, the assembler issues message `ASMA402W` and does not call the exit for any further processing.

If you assign SYSLST to a disk data set in your JCL, the record length must be 121.

4

Indicates that the exit writes the listing records in later WRITE calls. If you want 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.

Note: A reason code of 16 indicates a REINIT call is required.

16

Instructs the assembler to open the listing data set, and make no further calls to the exit.

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 - 32767. “LINECOUNT” on page 54 describes the LINECOUNT assembler option.

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.

Note: A reason code of 8 requests redrive of the exit.

z/VM and z/OS

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 in "For the LISTING exit" on page 85.

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.

z/VM and z/OS

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 assembler 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 in "For the LISTING exit" on page 85.

The exit sets the return code in the Exit Parameter List to one of these values:

0

Indicates that the record has been accepted, and the assembler is to write it to the listing 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 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.

Note: A reason code of 8 requests redrive of the exit.

z/VM and z/OS

The exit must also ensure that a valid printer control character is placed in the first character of the record. The printer control character is 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.

z/VSE The exit must also ensure that a valid American National Standard printer control character is placed in the first character of the record.

4 Instructs the assembler to discard the listing record.

Note: A reason code of 8 requests redrive of the exit.

16 Instructs the assembler to make no further calls to the exit.

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

Table 18 summarizes the LISTING exit processing.

Table 18. 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. If reason code=16, REINIT call required.
	4	Exit writes listing records. If reason code=4, exit supplies data set information. If reason code=16, REINIT call required.
	16	Assembler opens listing data set, and makes no further calls to the exit.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record. If reason code=8, redrive requested. 1
5=PROCESS	0	Record accepted. Exit might modify record. If reason code=4, the assembler, after processing the current record, provides an empty buffer for the exit to provide additional record. If reason code=8, redrive requested. 1
	4	Requests assembler to discard record. If reason code=8, redrive requested. 1
	16	Assembler makes no further calls to the exit.

Table 18. LISTING exit processing summary (continued)

Request Value=Type	Exit Return Code	Action
10=REINIT	0	Operation successful. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

OBJECT (z/OS and CMS) and PUNCH exit processing

When you specify the OBJEXIT suboption of the EXIT assembler option, the assembler calls either the PUNCH user exit or the OBJECT user exit, or both, as follows:

z/VM and z/ OS

- If you specify the OBJECT assembler option, the assembler calls the OBJECT user exit.
- If you specify the OBJECT and the DECK assembler options, the assembler calls the user exit as a PUNCH exit, and then as an OBJECT exit.
- If you specify the DECK assembler option, the assembler calls the PUNCH user exit.

You can use the exit to override the effect of the DECK or OBJECT 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 normally writes them.

z/VM and z/ OS
The exit is not called if you specify the NODECK and NOOBJECT assembler options.

z/VSE The exit is not called if you specify the NODECK assembler option.

If you want 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 these actions:

- Instruct the assembler to discard the records by setting the exit return code.
- Suppress the object output by doing this action:
  - **z/OS** 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.
  - **z/VSE** Assign `SYSPCH` to `IGN`.
- Instruct the exit to issue an OPEN return code of 4.

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 sets the return code in the Exit Parameter List to one of these values:

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

Note: A reason code of 16 indicates a REINIT call is required.

4 Indicates that the exit writes the object records in later WRITE calls. If you want 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 want 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.

Note: A reason code of 16 indicates a REINIT call is required.

16 Instructs the assembler to open the object data set and make no further calls to the exit.

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 the buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

Note: If both OBJECT and DECK are specified (so that the exit is called twice), the optional character string specified in the PARM string is passed each time.

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.

Note: A reason code of 8 requests redrive of the exit.

z/VM and z/OS
The record length is always 80 bytes when you specify the NOGOFF assembler option. If you specify the GOFF 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.

z/VSE The record length is always 80 bytes.

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.

z/VM and z/OS
The record length is always 80 bytes when you specify the NOXOBJECT 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.

z/VSE The record length is always 80 bytes.

The exit sets the return code in the Exit Parameter List to one of these values:

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.

Note: A reason code of 8 requests redrive of the exit.

4 Instructs the assembler to discard the record.

Note: A reason code of 8 requests redrive of the exit.

16 Instructs the assembler to make no further calls to the exit.

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

Table 19 summarizes the OBJECT and PUNCH exit processing.

Table 19. OBJECT and PUNCH exit processing summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens object data set. If reason code=16, REINIT call required.
	4	Exit writes object records. If reason code=4, exit supplies data set information. If reason code=16, REINIT call required.
	16	Assembler opens object data set, and makes no further calls to the exit.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record. If reason code=8, redrive requested. [1]
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. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.
10=REINIT	0	Operation successful. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

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 want 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 these actions:

- Instruct the assembler to discard the associated data records by setting the exit return code.
- Suppress the associated data output by doing this action:
  z/OS Provide a `//SYSADATA DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSADATA DUMMY` command.
  z/VSE Assign `SYSADAT` to `IGN`.

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 the buffer length contains the length of the character string. The buffer length is set to zero if there is no character string.

The exit sets the return code in the Exit Parameter List to one of these:

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

Note: A reason code of 8 indicates that the exit intends to discard both type X'0002' and X'0090' records presented to it during the assembly.

This reason code is used by the assembler when building the assembler summary section of the listing. To calculate accurate associated data record counts the assembler must be aware of any future discards of the final two records to be written; the statistics record and the end record.

A reason code of 16 indicates a REINIT call is required.

4 Indicates that the exit writes the associated data records in later WRITE calls. If you want to provide the assembler with the values for the system variables &SYSADATA_DSN, &SYSADATA_MEMBER, and &SYSADATA_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.

Note: A reason code of 8 indicates that the exit intends to discard both type X'0002' and X'0090' records presented to it during the assembly.

This reason code is used by the assembler when building the assembler summary section of the listing. To calculate accurate associated data record counts the assembler must be aware of any future discards of the final two records to be written; the statistics record and the end record.

A reason code of 16 indicates a REINIT call is required.

16 Instructs the assembler to open the ADATA data set and make no further calls to the exit.

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 associated data records. The buffer pointer field points to the associated data record, and the buffer length contains the length of the record. 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 associated data header, not the RDW.

Note: A reason code of 8 requests redrive of the exit.

z/VM and z/OS

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 in "For the LISTING exit" on page 85.

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 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 associated data header, not the RDW.

The exit sets the return code in the Exit Parameter List to one of these:

0 Indicates that the record has been accepted, and the assembler is to write it to the associated data 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 associated data 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 associated data 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.

Note: A reason code of 8 requests redrive of the exit.

4 Instructs the assembler to discard the record.

Note: A reason code of 8 requests redrive of the exit.

16 Instructs the assembler to make no further calls to the exit.

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

Table 20 on page 112 summarizes the ADATA exit processing.

Table 20. ADATA exit processing summary

Request Value=Type	Exit Return Code	Action
1=OPEN	0	Assembler opens associated data data set. If reason code=8, exit intends to discard both X'0002' and X'0090' record types. If reason code=16, REINIT call required.
	4	Exit writes associated data records. If reason code=4, exit supplies data set information. If reason code=8, exit intends to discard both X'0002' and X'0090' record types. If reason code=16, REINIT call required.
	16	Assembler opens associated data data set, and makes no further calls to the exit.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record. If reason code=8, redrive requested. [1]
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. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.
10=REINIT	0	Operation successful. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

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 normally writes them.

The TERMINAL exit is not called if you specify the NOTERM assembler option. If you want 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 these:

- Instruct the assembler to discard the terminal records by setting the exit return code.
- Suppress the terminal output by doing this:
  z/OS Provide a `//SYSTERM DD DUMMY` JCL statement.
  CMS Issue a `FILEDEF SYSTERM DUMMY` command.
  z/VSE Assign `SYSTERM` to `IGN`.
- Instruct the exit to issue an OPEN return code of 4.

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 sets the return code in the Exit Parameter List to one of these:

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 to 255 on z/OS and CMS, or from 1 to 125 on z/VSE. If the value is zero or greater than 255 on z/OS and CMS, or zero or greater than 125 on z/VSE, the assembler issues message `ASMA404W` and does not call the exit for any further processing.

Note: A reason code of 16 indicates a REINIT call is required.

4 Indicates that the exit writes the terminal records in later WRITE calls. If you want 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.

Note: A reason code of 16 indicates a REINIT call is required.

16 Instructs the assembler to open the terminal data set and make no further calls to the exit.

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 the 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.

Note: A reason code of 8 requests redrive of the exit.

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 sets the return code in the Exit Parameter List to one of these:

0 Indicates that the record has been accepted, and the assembler is to write it to the terminal 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 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.

Note: A reason code of 8 requests redrive of the exit.

4 Instructs the assembler to discard the terminal record.

Note: A reason code of 8 requests redrive of the exit.

16 Instructs the assembler to make no further calls to the exit.

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

Table 21 summarizes the TERM exit processing.

Table 21. 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. If reason code=16, REINIT call required.
	4	Exit writes terminal records. If reason code=4, exit supplies system variable symbols. If reason code=16, REINIT call required.
	16	Assembler opens terminal data set, and makes no further calls to the exit.
2=CLOSE	n/a	Exit should close any data sets it opened, and release any storage it acquired.
4=WRITE	0	Exit writes record. If reason code=8, redrive requested. [1]
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. If reason code=8, redrive requested. [1]
	4	Requests assembler to discard record. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.
10=REINIT	0	Operation successful. If reason code=8, redrive requested. [1]
	16	Assembler makes no further calls to the exit.

Notes:

1. When redrive is requested, the assembler issues any required error message and then recalls the exit with the same parameter list as before.

Sample user exits

Three sample exits are provided with High Level Assembler. They are described under:

- Appendix H, “Sample ADATA user exits (z/OS and CMS),” on page 351
- Appendix I, “Sample LISTING user exit (z/OS and CMS),” on page 361
- Appendix J, “Sample SOURCE user exit (z/OS and CMS),” on page 363

User exit coding example

Figure 28 on page 116 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.

This user exit is not supplied with High Level Assembler. This exit is written for z/OS and CMS only.

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 its 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 macros 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 macros 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 allows the assembler to open the associated data data set. The exit issues a WTO for the first 80 characters of each associated data record.

If you specify EXIT(ADEXIT(MYEXIT(EXIT))), the exit opens the associated data data set. The exit issues a WTO for the first 80 characters of each associated data record passed to the exit.

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 (z/OS/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 28. Example of a user exit (part 1 of 17)

```
***********************************************************************
* MYEXIT   Entry                                                      *
* - Save the registers.                                               *
* - Acquire the dynamic storage on the first entry and save the       *
*   address in AXPUSER.                                               *
* - Chain the save areas using the forward and backward pointers.     *
* - Address the data areas passed.                                    *
* - Process the required exit according to the 'Exit type' passed.    *
***********************************************************************
MYEXIT   CSECT
         STM   R14,R12,12(R13)         Save registers
         LR    R12,R15                 Set up first base register
         USING MYEXIT,R12,R11
         LA    R11,2048(,R12)
         LA    R11,2048(,R11)          Set up second base register
         LR    PARMREG,R1              Save parameter list address
         USING AXPXITP,PARMREG
         L     REQREG,AXPRIP           Get address of exit parm list
         USING AXPRIL,REQREG
         ICM   R1,B'1111',AXPUSER      Get address of user area
         BNZ   CHAIN                   Yes, use area
         LA    0,WORKLEN               Otherwise, get length
         GETMAIN R,LV=(0)              and getmain storage
         ST    R1,AXPUSER              Save it for later
         XC    0(WORKLEN,R1),0(R1)     Clear area
CHAIN    DS    0H
         ST    R13,4(R1)               Save previous pointer
         ST    R1,8(R13)               Save next pointer
         LR    R13,R1                  Set savearea/workarea address
         USING WORKAREA,R13
         SPACE 1
         L     BUFREG,AXPBUFP          Get address of buffer
         USING BUFF,BUFREG
         L     ERRREG,AXPERRP          Get address of error buffer
         USING ERRBUFF,ERRREG
         L     SRCREG,AXPSIP           Get address of source info
         USING AXPSIL,SRCREG
         L     DCBREG,AXPDCBP          Get address of DCB
         USING IHADCB,DCBREG
         SPACE 1
         XC    AXPRETC,AXPRETC         Zero the return code
         L     R15,AXPTYPE             Load the exit type value (1-7)
         BCTR  R15,0                   Decrement by 1
         SLL   R15,1                   Multiply by 2
         LH    R15,EXITADDR(R15)       Index into address list
         AR    R15,R12                 Calculate the address
         BR    R15                     Branch to applicable routine
         SPACE 1
EXITADDR DC    Y(SOURCE-MYEXIT)
         DC    Y(LIBRARY-MYEXIT)
         DC    Y(LISTING-MYEXIT)
         DC    Y(PUNCH-MYEXIT)
         DC    Y(OBJECT-MYEXIT)
         DC    Y(ADATA-MYEXIT)
         DC    Y(TERM-MYEXIT)
         DC    Y(*-*)
         EJECT
```

Figure 29. Example of a user exit (part 2 of 17)

```
***********************************************************************
* MYEXIT   Exit1                                                      *
* - Restore the callers register 13                                   *
* - Restore the registers and set the register 15 to zero.            *
* - Return to the caller.                                             *
***********************************************************************
EXIT1    DS    0H
         MVC   LASTOP,AXPRTYP+3       Save last operation code
         L     R13,4(,R13)            Unchain save areas
EXIT2    DS    0H
         LM    R14,R12,12(R13)        Restore callers registers
         LA    R15,0                  Set the return code
         BSM   R0,R14                 Return to caller
         SPACE 1
***********************************************************************
* MYEXIT   - Free storage                                             *
* - Called on a CLOSE request.                                        *
* - Free the storage acquired and zero AXPUSER.                       *
* - Go to EXIT (after R13 is restored)                                *
***********************************************************************
FREESTOR DS    0H
         XC    AXPUSER,AXPUSER        Zero User field
         LA    0,WORKLEN              Length of area to free
         LR    R1,R13                 Address of area to free
         L     R13,4(,R13)            Restore callers register 13
         FREEMAIN R,A=(1),LV=(0)      Free the storage acquired
         B     EXIT2
         SPACE 1
***********************************************************************
* MYEXIT   - Logic error                                              *
* - If an error occurred, set up the error message in the buffer      *
*   and length in AXPERRL.  Set the severity code.                    *
* - Set the return code to 20.                                        *
* - Return to the caller.                                             *
***********************************************************************
LOGICERR DS    0H
         MVC   AXPRETC,=A(AXPCBAD)    Severe error occurred
         MVC   ERRBUFF(ERRMSGL),ERRMSG Set up error message
         MVC   AXPERRL,=A(ERRMSGL)    Set up error message length
         MVC   AXPSEVC,=A(20)         Set up error message severity
         B     EXIT1
         EJECT
***********************************************************************
* SOURCE EXIT                                                         *
* - Process required request type                                     *
***********************************************************************
SOURCE   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,SOURCE_ADDR(R15)   Index into Address list
         AR    R15,R12                Calculate the address
         BR    R15                    Branch to applicable routine
SOURCE_ADDR DC Y(SOURCE_OPEN-MYEXIT)
         DC    Y(SOURCE_CLOSE-MYEXIT)
         DC    Y(SOURCE_READ-MYEXIT)
         DC    Y(SOURCE_WRITE-MYEXIT)
         DC    Y(SOURCE_PROCESS-MYEXIT)
         DC    Y(*-*)
         SPACE 1
```

Figure 30. Example of a user exit (part 3 of 17)

```
***********************************************************************
* SOURCE EXIT - Process OPEN request                                  *
* - Pick up character string if it is supplied.                       *
* - Set return code indicating whether the assembler or user exit     *
*   will open the primary input data set.                             *
* - Open data set if required.                                        *
***********************************************************************
SOURCE_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    SOURCE_NOSTR            Yes, no string passed
         BCTR  R1,0                    Decrement for execute
         EX    R1,UPPERSTR             Move and uppercase string
SOURCE_NOSTR    DS    0H
         CLC   OPENPARM(8),=CL8'EXIT'  Will user exit read input?
         BE    SOURCE_OPEN_EXIT        Yes
         MVC   AXPRETC,=A(0)           assembler to read primary input
         B     EXIT1                   Return
SOURCE_OPEN_EXIT DS   0H
         OI    OPENFLAG,EXIT           Set flag
         MVC   AXPRETC,=A(AXPCOPN)     User exit to read primary input
         LA    R1,SRC1                 Address first source record
         ST    R1,CURR_PTR             Set up pointer
         B     EXIT1                   Return
         SPACE 1
***********************************************************************
* SOURCE EXIT - Process CLOSE request                                 *
* - Close data set if required.                                       *
* - Free storage and return.                                          *
***********************************************************************
SOURCE_CLOSE    DS    0H
         B     FREESTOR
         SPACE 1
```

Figure 31. Example of a user exit (part 4 of 17)

```
***********************************************************************
* SOURCE EXIT - Process READ request                                  *
* - Provide source information about 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     EXIT1
SOURCE_EOF      DS    0H
         MVC   AXPRETC,=A(AXPCEOD)     End of file on input
         B     EXIT1
         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     EXIT1
         EJECT
```

Figure 32. Example of a user exit (part 5 of 17)

```
***********************************************************************
* LIBRARY EXIT                                                        *
* - Process required request type                                     *
***********************************************************************
LIBRARY  DS    0H
         L     R15,AXPRTYP             Get the request type value (1-8)
         BCTR  R15,0                   Decrement by 1
         SLL   R15,1                   Multiply by 2
         LH    R15,LIBRARY_ADDR(R15)   Index into Address list
         AR    R15,R12                 Calculate the address
         BR    R15                     Branch to applicable routine
LIBRARY_ADDR DC Y(LIBRARY_OPEN-MYEXIT)
         DC    Y(LIBRARY_CLOSE-MYEXIT)
         DC    Y(LIBRARY_READ-MYEXIT)
         DC    Y(LIBRARY_WRITE-MYEXIT)
         DC    Y(LIBRARY_PR_MAC-MYEXIT)
         DC    Y(LIBRARY_PR_CPY-MYEXIT)
         DC    Y(LIBRARY_FIND_MAC-MYEXIT)
         DC    Y(LIBRARY_FIND_CPY-MYEXIT)
         DC    Y(LIBRARY_EOM-MYEXIT)
         DC    Y(*-*)
         SPACE 1
***********************************************************************
* LIBRARY EXIT - Process OPEN request                                 *
* - Pick up character string if it is supplied.                       *
* - Set return code indicating whether the assembler, user exit or    *
*   both will process the library.                                    *
* - Open data set if required.                                        *
***********************************************************************
LIBRARY_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    LIBRARY_NOSTR           Yes, no string passed
         BCTR  R1,0                    Decrement for execute
         EX    R1,UPPERSTR             Move and uppercase string
LIBRARY_NOSTR   DS    0H
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process library
         BE    LIBRARY_OPEN_EXIT       Yes
         CLC   OPENPARM(4),=CL8'BOTH'  Will Both process library
         BE    LIBRARY_OPEN_BOTH       Yes
         MVC   AXPRETC,=A(0)           assembler to process library
         B     EXIT1                   Return
LIBRARY_OPEN_EXIT DS   0H
         OI    OPENFLAG,EXIT           Set flag
         MVC   AXPRETC,=A(AXPCOPN)     User exit to process library
         MVC   AXPREAC,=A(AXPEEOM)     EXIT to get End of member calls
         B     EXIT1                   Return
LIBRARY_OPEN_BOTH DS   0H
         OI    OPENFLAG,BOTH           Set flag
         MVC   AXPRETC,=A(AXPCOPL)     Both to process library
         MVC   AXPREAC,=A(AXPEEOM)     EXIT to get End of member calls
         B     EXIT1                   Return
         SPACE 1
```

Figure 33. Example of a user exit (part 6 of 17)

```
***********************************************************************
* LIBRARY EXIT - Process CLOSE request                                *
* - Close data set if required.                                       *
* - Free storage and return.                                          *
***********************************************************************
LIBRARY_CLOSE    DS    0H
         USING LIBSTACK,R2             Map stack entries
         ICM   R2,B'1111',STACKPTR     Check that stack is empty
         BZ    FREESTOR                It should be!
LIBRARY_FREE_LOOP DS   0H
         LTR   R1,R2                   Load address for FREEMAIN
         BZ    FREESTOR                Finished here
         L     R2,NEXT_MEM             Prepare for next loop
         LA    R0,LIBSTACK_LEN         Load length for FREEMAIN
         FREEMAIN R,A=(1),LV=(0)       Free the storage acquired
         B     LIBRARY_FREE_LOOP
         SPACE 1
***********************************************************************
* LIBRARY EXIT - Process READ request                                 *
* - Read copy/macro source and place in buffer.                       *
* - Set return code to 16 at end of member.                           *
***********************************************************************
LIBRARY_READ     DS    0H
         ICM   R2,B'1111',STACKPTR     Is the stack empty?
         BZ    LIBRARY_STACK_ERR       It shouldn't be!
         L     R1,MEM_PTR              Get record address
         CLI   0(R1),X'FF'             Is it EOF?
         BE    LIBRARY_EOF             Yes, set return code
         MVC   0(80,BUFREG),0(R1)
         LA    R1,80(,R1)              Point to next record address
         ST    R1,MEM_PTR               and save in stack entry
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for library record
         L     R1,AXPRELREC            Update
         LA    R1,1(R1)                 Relative Record
         ST    R1,AXPRELREC              Number
         ST    R1,MEM_RELREC            and save in stack entry
         L     R1,AXPABSREC            Update
         LA    R1,1(R1)                 Absolute Record
         ST    R1,AXPABSREC              Number
         B     EXIT1
LIBRARY_EOF      DS    0H
         MVC   AXPRETC,=A(AXPCEOD)     End of file on input
         B     EXIT1
         SPACE 1
***********************************************************************
* LIBRARY EXIT - Process WRITE request                                *
* - Not valid for LIBRARY exit.                                       *
* - Set return code to 20 and set up error message.                   *
***********************************************************************
LIBRARY_WRITE    DS    0H
         B     LOGICERR
         SPACE 1
***********************************************************************
* LIBRARY EXIT - Process PROCESS MACRO/COPY 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.                                               *
***********************************************************************
LIBRARY_PR_MAC  DS    0H
LIBRARY_PR_CPY  DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for library record
         B     EXIT1
         SPACE 1
```

Figure 34. Example of a user exit (part 7 of 17)

```
***********************************************************************
* LIBRARY EXIT - Process FIND MACRO/COPY request                      *
* - Search for the member.  Set the return code to indicate           *
*   whether the member was found.                                     *
* - If the member is found, the source information is returned.       *
***********************************************************************
LIBRARY_FIND_MAC  DS   0H
LIBRARY_FIND_CPY  DS   0H
         CLC   AXPOPTS,=A(AXPORES)     Is it a resume request?
         BE    LIBRARY_RESUME          Yes, resume member
         LA    R3,MACA1
         CLC   AXPMEMN(8),=CL8'OUTER'
         BE    LIBRARY_FOUND
         LA    R3,MACB1
         CLC   AXPMEMN(8),=CL8'INNER'
         BE    LIBRARY_FOUND
         LA    R3,CPYA1
         CLC   AXPMEMN(8),=CL8'TINY'
         BE    LIBRARY_FOUND
         LA    R3,CPYB1
         CLC   AXPMEMN(8),=CL8'TINY1'
         BE    LIBRARY_FOUND
         MVC   AXPRETC,=A(AXPCMNF)     Indicate member not found
         B     EXIT1
LIBRARY_FOUND     DS    0H
         ICM   R2,B'1111',STACKPTR     Is the stack empty?
         BZ    LIBRARY_GET_STACK
         CLC   AXPOPTS,=A(AXPONEST)    Is it a nested COPY/MACRO?
         BNE   LIBRARY_STACK_ERR           NO - report an error
LIBRARY_GET_STACK     DS    0H
         LA    R0,LIBSTACK_LEN         Load reg with length
         GETMAIN R,LV=(0)              and getmain storage
         XC    0(LIBSTACK_LEN,R1),0(R1) Clear the storage
NEW_LIBSTACK USING LIBSTACK,R1         Map the new stack entry
         ST    R2,NEW_LIBSTACK.NEXT_MEM Add new link to top of stack
         DROP  NEW_LIBSTACK
         ST    R1,STACKPTR             Re-anchor the stack
         LR    R2,R1                   Make the new entry current
         ST    R3,MEM_PTR              Save current record pointer
         MVC   MEM_NAME,AXPMEMN        Save name in stack entry
         MVC   AXPREAC,=A(AXPEISA)     Indicate source info available
         MVC   AXPMEMT,=CL255'None'
         MVC   AXPDSN,=CL255'LIBRARY.data set.NAME'
         MVC   AXPVOL,=CL255'VOL002'
         XC    AXPRELREC,AXPRELREC     Set relative record No to zero
         B     EXIT1
***********************************************************************
* LIBRARY EXIT - Process FIND (resume) request                        *
* - Set the relative record number in the parameter list              *
* N.B. if the EXIT read the records from disk, at this point it would *
*      use the information saved in the stack to reposition itself    *
*      ready for the next read. (i.e. a FIND and POINT)               *
***********************************************************************
LIBRARY_RESUME    DS    0H             Stack Management now in EOM call
         MVC   AXPRETC,=A(AXPCMNF)     Assume member not found
         ICM   R2,B'1111',STACKPTR     Is the stack empty?
         BZ    LIBRARY_CHECK_BOTH      Yes - check open option
         CLC   MEM_NAME,AXPMEMN        Compare name with stack entry
         BNE   LIBRARY_CHECK_BOTH      Not equal - check open option
         MVC   AXPRETC,=A(0)           Correct our assumption
         L     R0,MEM_RELREC           Get saved rel rec no from stack
         ST    R0,AXPRELREC            Set relative record No
         B     EXIT1
         SPACE 1
```

Figure 35. Example of a user exit (part 8 of 17)

```
************************************************************************
* LIBRARY EXIT - Use End of Member calls to perform stack management   *
* - Compare member name, if equal unstack the top entry                *
************************************************************************
LIBRARY_EOM       DS   0H
         ICM   R2,B'1111',STACKPTR     Is the stack empty?
         BZ    LIBRARY_CHECK_BOTH      Yes - check open option
         CLC   MEM_NAME,AXPMEMN        Compare name with stack entry
         BNE   LIBRARY_CHECK_BOTH      Not equal - check open option
         LR    R1,R2                   Load address for FREEMAIN
         L     R2,NEXT_MEM             Get address of next entry
         ST    R2,STACKPTR              and save it.
         DROP  R2
         LA    R0,LIBSTACK_LEN         Load length for FREEMAIN
         FREEMAIN R,A=(1),LV=(0)       Free the storage acquired
LIBRARY_CHECK_BOTH DS  0H
         CLI   OPENFLAG,BOTH           Did EXIT open with BOTH option
         BE    EXIT1                   Yes - don't issue error msg
************************************************************************
* LIBRARY EXIT - Stack Error Routine                                   *
* - If an error occurred, set up the error message in the buffer       *
*   and length in AXPERRL.  Set the severity code.                     *
* - Set the return code to 20.                                         *
* - Return to the caller.                                              *
************************************************************************
LIBRARY_STACK_ERR DS   0H
         MVC   AXPRETC,=A(AXPCBAD)     Severe error occurred
         MVC   ERRBUFF(ERRMSGL),STKMSG Set up error message
         MVC   AXPERRL,=A(STKMSGL)     Set up error message length
         MVC   AXPSEVC,=A(20)          Set up error message severity
         B     EXIT1
         EJECT
************************************************************************
* 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
```

Figure 36. Example of a user exit (part 9 of 17)

```
***********************************************************************
* 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,UPPERSTR             Move and uppercase 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     EXIT1                   Return
LISTING_OPEN_EXIT DS   0H
         OI    OPENFLAG,EXIT           Set flag
         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     EXIT1                   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
***********************************************************************
* LISTING EXIT - Process WRITE request                                *
* - Write the listing record passed.                                  *
***********************************************************************
LISTING_WRITE     DS   0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for listing 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     EXIT1
         SPACE 1
```

Figure 37. Example of a user exit (part 10 of 17)

```
**********************************************************************
* LISTING 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.                                              *
**********************************************************************
LISTING_PROCESS DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for listing record
         B     EXIT1
         EJECT
**********************************************************************
* OBJECT EXIT                                                        *
* - Process required request type                                    *
**********************************************************************
PUNCH    DS    0H
OBJECT   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,OBJECT_ADDR(R15)    Index into Address list
         AR    R15,R12                 Calculate the address
         BR    R15                     Branch to applicable routine
OBJECT_ADDR DC Y(OBJECT_OPEN-MYEXIT)
         DC    Y(OBJECT_CLOSE-MYEXIT)
         DC    Y(OBJECT_READ-MYEXIT)
         DC    Y(OBJECT_WRITE-MYEXIT)
         DC    Y(OBJECT_PROCESS-MYEXIT)
         DC    Y(*-*)
         SPACE 1
**********************************************************************
* OBJECT 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 object/punch records.                             *
* - Open data set if required                                        *
**********************************************************************
OBJECT_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    OBJECT_NOSTR            Yes, no string passed
         BCTR  R1,0                    Decrement for execute
         EX    R1,UPPERSTR             Move and uppercase string
OBJECT_NOSTR   DS    0H
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process object
         BE    OBJECT_OPEN_EXIT        Yes
         MVC   AXPRETC,=A(0)           assembler to write object/punch
         B     EXIT1                   Return
OBJECT_OPEN_EXIT DS   0H
         OI    OPENFLAG,EXIT           Set flag
         MVC   AXPRETC,=A(AXPCOPN)     User exit to write object/punch
         MVC   AXPMEMN,=CL255'Member'
         MVC   AXPMEMT,=CL255' '
         MVC   AXPDSN,=CL255'OBJECT.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     EXIT1                   Return
         SPACE 1
```

Figure 38. Example of a user exit (part 11 of 17)

```
**********************************************************************
* OBJECT EXIT - Process CLOSE request                                *
* - Close data set if required.                                      *
* - Free storage and return.                                         *
**********************************************************************
OBJECT_CLOSE    DS    0H
         B     FREESTOR
         SPACE 1
**********************************************************************
* OBJECT EXIT - Process READ request                                 *
* - Not valid for OBJECT exit.                                       *
* - Set return code to 20 and set up error message.                  *
**********************************************************************
OBJECT_READ     DS    0H
         B     LOGICERR
**********************************************************************
* OBJECT EXIT - Process WRITE request                                *
* - Write the source record passed.                                  *
**********************************************************************
OBJECT_WRITE    DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)              Issue WTO for object 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     EXIT1
         SPACE 1
**********************************************************************
* OBJECT 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.                                              *
**********************************************************************
OBJECT_PROCESS DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)              Issue WTO for object record
         B     EXIT1
         EJECT
**********************************************************************
* 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
```

Figure 39. Example of a user exit (part 12 of 17)

```
***********************************************************************
* ADATA 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 associated data.                                   *
* - Open data set if required.                                        *
***********************************************************************
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,UPPERSTR             Move and uppercase string
ADATA_NOSTR   DS    0H
         CLC   OPENPARM(4),=CL8'EXIT'  Will user exit process adata
         BE    ADATA_OPEN_EXIT         Yes
         MVC   AXPRETC,=A(0)           assembler to write adata
         B     EXIT1                   Return
ADATA_OPEN_EXIT  DS   0H
         OI    OPENFLAG,EXIT           Set flag
         MVC   AXPRETC,=A(AXPCOPN)     User exit to write adata
         MVC   AXPMEMN,=CL255' '
         MVC   AXPMEMT,=CL255' '
         MVC   AXPDSN,=CL255'ADATA.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     EXIT1                   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                                  *
* - Write the adata record passed.                                    *
***********************************************************************
ADATA_WRITE    DS     0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)             Issue WTO for adata 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     EXIT1
         SPACE 1
```

Figure 40. Example of a user exit (part 13 of 17)

```
************************************************************************
* ADATA 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.                                                *
************************************************************************
ADATA_PROCESS DS    0H
         MVC   WTOL+4(80),0(BUFREG)
         WTO   MF=(E,WTOL)              Issue WTO for ADATA record
         B     EXIT1
         EJECT
************************************************************************
* 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,UPPERSTR              Move and uppercase 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     EXIT1                    Return
TERM_OPEN_EXIT DS      0H
         OI    OPENFLAG,EXIT            Set flag
         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     EXIT1                    Return
         SPACE 1
```

Figure 41. Example of a user exit (part 14 of 17)

```
***********************************************************************
* 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
***********************************************************************
* 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     EXIT1
         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     EXIT1
STKMSG   DC    C'LIBRARY EXIT encountered a stack error'
STKMSGL  EQU   *-ERRMSG
ERRMSG   DC    C'Invalid EXIT type or Request type passed to exit'
ERRMSGL  EQU   *-ERRMSG
WTOL     WTO   '1234567890123456789012345678901234567890123456789012345X
               6789012345678901234567890',MF=L
UPPERSTR OC    OPENPARM(*-*),0(BUFREG) Move and uppercase string
         SPACE 1
```

Figure 42. Example of a user exit (part 15 of 17)

```
SRC1     DC    CL80'SMALL    TITLE ''Test the assembler exits'''
SRC2     DC    CL80'         MACRO'
SRC3     DC    CL80'         LITTLE'
SRC4     DC    CL80'         BSM   0,14  Return'
SRC5     DC    CL80'         MEND'
SRC6     DC    CL80'         START'
SRC7     DC    CL80'         OUTER'
SRC8     DC    CL80'         LITTLE'
SRC9     DC    CL80'         REPRO'
SRC10    DC    CL80'This is to be written to the punch data set'
SRC11    DC    CL80'         COPY  TINY'
SRC12    DC    CL80'         END'
SRCEND   DC    X'FF'         END OF SOURCE STMTS
         SPACE 1
MACA1    DC    CL80'         MACRO'
MACA2    DC    CL80'         OUTER'
MACA3    DC    CL80'         XR    15,15'
MACA4    DC    CL80'         INNER'
MACA5    DC    CL80'         LTR   15,15'
MACA6    DC    CL80'         MEND'
MACAEND  DC    X'FF'         END OF MACRO STMTS
         SPACE 1
MACB1    DC    CL80'         MACRO'
MACB2    DC    CL80'         INNER'
MACB3    DC    CL80'         LR    12,15'
MACB4    DC    CL80'         MEND'
MACBEND  DC    X'FF'         END OF MACRO STMTS
         SPACE 1
CPYA1    DC    CL80'TINY     DSECT               LINE 1 TINY'
CPYA2    DC    CL80'         DS    C''TINY''       LINE 2 TINY'
CPYA3    DC    CL80'         COPY  TINY1         LINE 3 TINY'
CPYA4    DC    CL80'         DS    CL10''TINY''    LINE 4 TINY'
CPYA5    DC    CL80'         DS    CL80          LINE 5 TINY'
CPYEND   DC    X'FF'         END OF COPY STMTS
CPYB1    DC    CL80'TINY1    DSECT                 LINE 1 TINY1'
CPYB2    DC    CL80'         DS    C''TINY1''        LINE 2 TINY1'
CPYB3    DC    CL80'         DS    CL10''TINY1''     LINE 3 TINY1'
CPYBEND  DC    X'FF'         END OF COPY STMTS
         SPACE 1
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
DCBREG   EQU   5                  Address of DCB
SRCREG   EQU   6                  Address of Source Information
ERRREG   EQU   7                  Address of Error Buffer
BUFREG   EQU   8                  Address of buffer
REQREG   EQU   9                  Address of request information
PARMREG  EQU   10                 Address or parameter
```

Figure 43. Example of a user exit (part 16 of 17)

```
         LTORG ,
         SPACE 1
         DCBD  DSORG=PS,DEVD=DA
         SPACE 1
         ASMAXITP ,              Mapping for exit parameter list
         SPACE 1
BUFF     DSECT ,
         DS    CL255             Record buffer
         SPACE 1
ERRBUFF  DSECT ,
         DS    CL255             Error message buffer
         SPACE 1
WORKAREA DSECT
SAVEAREA DS    18F               Save area
OPENPARM DS    CL64              Character string passed at open time
OPENFLAG DS    X                 Type of Operation requested at OPEN
EXIT     EQU   X'80'
BOTH     EQU   X'C0'
LASTOP   DS    X                 Previous request type
CURR_PTR DS    A                 Current record pointer
STACKPTR DS    A                 Address of top of Lib status stack
WORKLEN  EQU   *-WORKAREA
LIBSTACK DSECT                   Library status stack entry
NEXT_MEM DS    A                 Address of entry next in stack
MEM_PTR  DS    A                 Current record pointer
MEM_RELREC DS  F                 Current relative record number
MEM_NAME DS    CL64              Stack of saved member names
LIBSTACK_LEN  EQU   *-LIBSTACK
         END   MYEXIT
```

Figure 44. Example of a user exit (part 16 of 17)

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 1024 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 that 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 expects the external function module to be generated in 31-bit addressing mode (AMODE 31). The external function must return to the assembler in the same addressing mode from which it was called after restoring the registers to the values they contained at the time of the call. Only one copy of the load module is loaded, so it must be serially reusable. The assembler must be able to locate the external function module as follows:

z/OS	The external function must be a link-edited load module in a partitioned data set, or a program object in a PDSE, that is in the standard search sequence. The external function can also be located in the Link Pack Area (LPA).
z/VM	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.
z/VSE	The external function must be a relocatable phase in a sublibrary that is specified in the LIBDEF phase search chain. The external function can also be located in the Shared Virtual Area (SVA).

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, allow enough virtual storage in the address space (z/OS and CMS) or the partition (z/VSE) in which the assembler runs, so that the external function modules can be loaded successfully, and for any storage that 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.

© Copyright IBM Corp. 1992, 2013

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

Registers 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 45 on page 137 shows the SETAF parameter list, and Figure 46 on page 138 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.

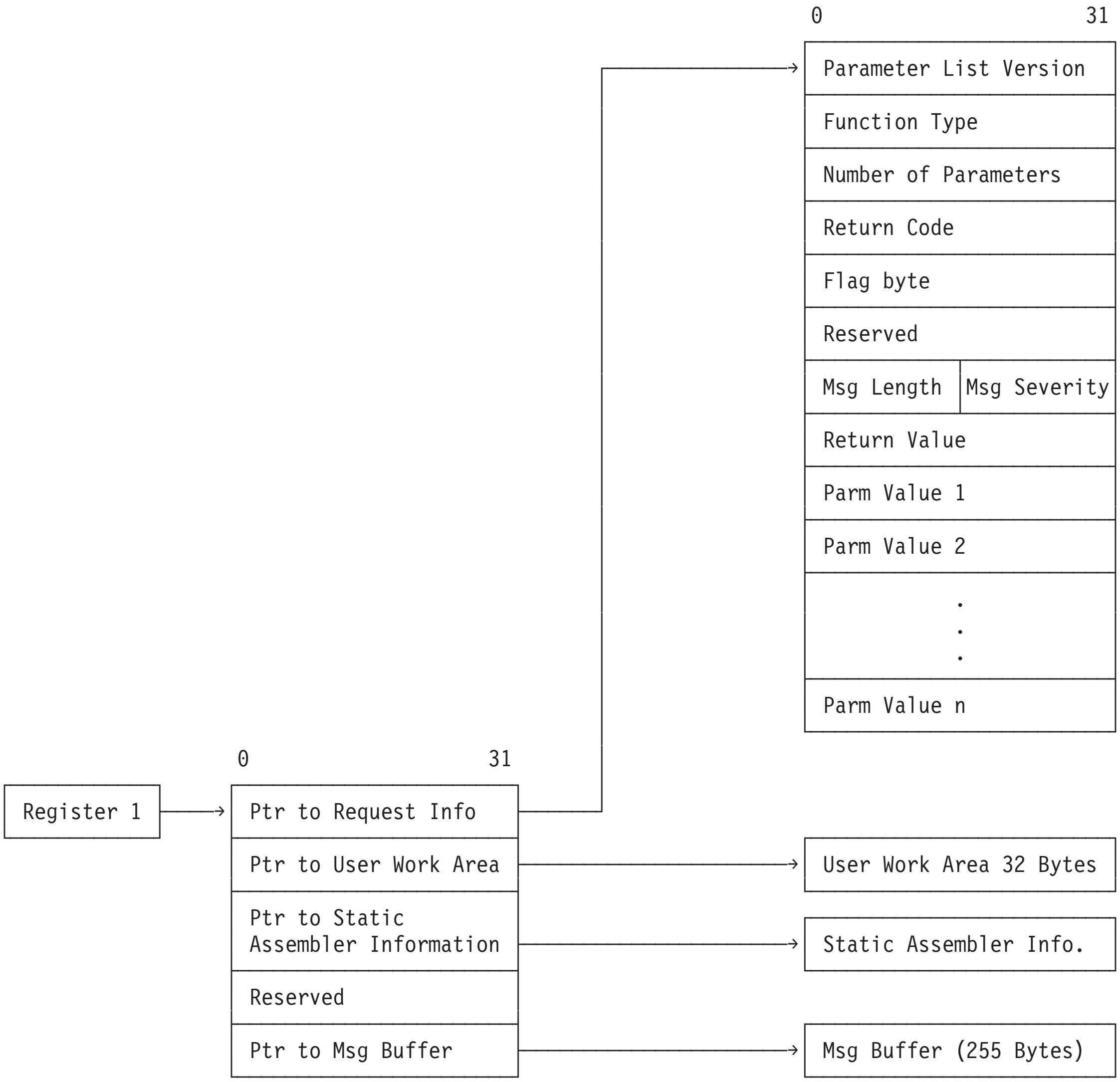


Figure 45. SETAF external function parameter list format

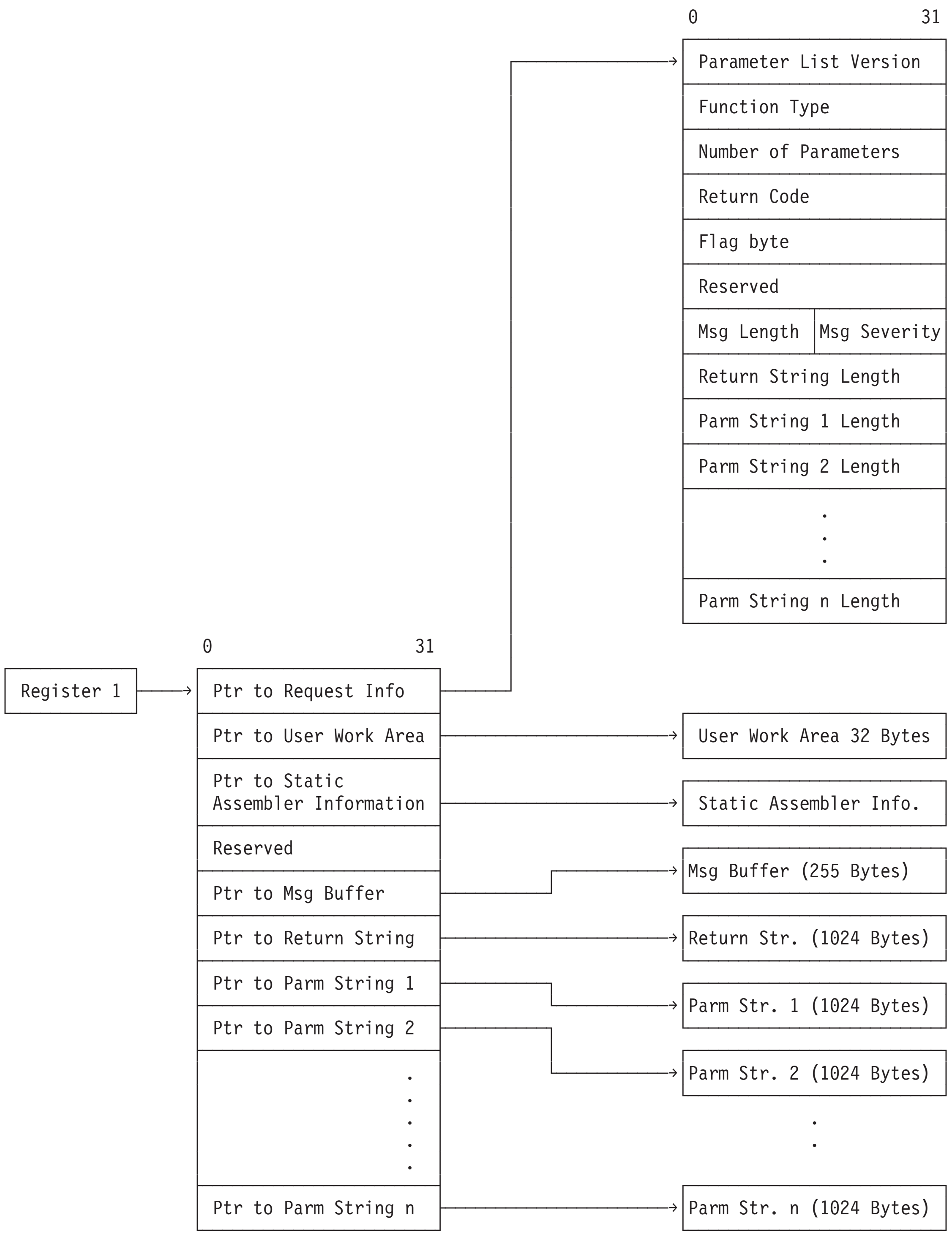


Figure 46. SETCF external function parameter list format

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 these fields:

Parameter list version

A fullword identifying which version of the parameter list is provided to the external function. For High Level Assembler Release 6 this field contains a value of 3.

Function type

A fullword, set by the assembler to indicate the function type:

0	CLOSE call
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:

0	Operation successful. Value or string returned.
>0	Operation failed. Request assembler to terminate immediately.

When the return code is greater than 0 the assembler issues diagnostic error message `ASMA941U`.

Flag byte

X'80' Function requests a CLOSE call.

The CLOSE call is not enabled by default. Each time the external function is called, it is able to set (or reset) this flag to indicate that it needs to perform some extra processing (releasing storage, for example) before being deleted. The external function might therefore set the flag on one call and reset it on another.

If the flag is set at the end of the assembly, HLASM calls the function with a CLOSE code to allow it to release resources.

The assembler maintains the Flag Byte and provides it to the external function on all calls.

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 (which are liable to change without notice).

External functions can request that a message be issued on their behalf. The function provides the text of the message, and the assembler inserts the function's name and the supplied text into one of five messages. The relevant information is contained in two fields, **Message Length** and **Message Severity**:

Msg length

A halfword, set by the external function, indicating the length of the message to be issued.

Msg severity

A halfword, set by the external function, from which the assembler determines the associated message number. The severity code returned by the function is rounded up to a multiple of four as shown in Table 22 on page 140.

Table 22. Message severity and associated messages

Severity Code Specified	Severity Code Used	Associated Message
0	0	ASMA710I
1–4	4	ASMA711W
5–8	8	ASMA712E
9–12	12	ASMA713S
>12	16	ASMA714C

Return value (SETAF)

A fullword, set by the external function. This field is set to zero by the assembler before the external function call.

Parm value *n* (SETAF)

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 *n*** fields in the Request Information List.

Return string length (SETCF)

An unsigned fullword, set by the external function, containing the length of the string pointed to by the **Pointer to Parm String** field.

The assembler uses this field as the length of the returned string.

If the length is greater than 1024, it is reset to 1024 by the assembler. The consequence of returning a string longer than 1024 bytes is unpredictable.

Parm string *n* length (SETCF)

An unsigned fullword, set by the assembler, containing the length of the string pointed to by the **Ptr to Parm String *n*** 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 1024 inclusive.

The **Number of Parameters** field indicates the number of **Parm String *n* Length** fields in the Request Information List.

Pointer to user work area

Pointer to the User Work Area.

The assembler provides four doublewords of storage for use by the external function. This storage is doubleword 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 are preserved across all call types (SETAF, SETCF, and CLOSE) until the assembly completes. The assembler does not use or modify the work area.

Pointer to static assembler information

Pointer to the Static Assembler Information.

This is pointed to by the Static Assembler Information Pointer. See "Static assembler information pointer" on page 91.

Pointer to msg buffer

Pointer to the "function-supplied message" area.

The assembler always sets this pointer before invoking an external function. The external function can put up to 255 bytes of message text into the area addressed by this field.

Pointer to return string (SETCF)

Pointer to 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 1024 bytes of character string data into the area addressed by this field.

Pointer to parm string *n* (SETCF)

Pointer to **Parm String *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 **Pointer to Parm String *n*** fields 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 F, “High Level Assembler messages,” on page 297.

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 can 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 UNDEFINED OPERATION CODE — xxxxxxxx
```

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, system variable, macro parameter, or value string associated with the error.

Macro Parameters: Messages reference three types of macro parameter: the *name field* parameter, *keyword* parameters, and *positional* parameters. A reference to the name field parameter is indicated by the word “NAME” appended to the message. References to keyword and positional parameters (for which there might be multiple occurrences) are in the form “KPARM*nnnn*” and “PPARM*nnnn*”, where *nnnn* is the relative number of the parameter within the macro definition.

© Copyright IBM Corp. 1992, 2013

Figure 47 shows an example of a macro with messages referencing each type of variable or parameter.

```
Active Usings: None
Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                    1     MACRO                                                                00001000
                                        3          4                           4                  5
                                    2 &z  parms &kw1=a,&kw2=b,&kw3=c,&kw4=d,&kw5=e,&kw6=f,&pp1,&pp2            00002000
                                    3 &c  SETC   'just a string'                                               00003000
                                    4 &ss SETA   &c                                                            00004000
                                    5 &sv SETA   &sysasm   2                                                   00005000
                                    6 &z1 SETA   &z    3                                                       00006000
                                    7 &k1 SETA   &kw1 ←┬→ 4                                                    00007000
                                    8 &k5 SETA   &kw5 ←┘                                                       00008000
                                    9 &n  SETA   n'&syslist                                                    00009000
                                   10 &pn     SETA   &syslist(&n) ←┬→ 5                                        00010000
                                   11 &p2     SETA   &pp2         ←┘         ┌→ 5                            00011000
                                   12     MEND                           ┌───┴──┐                              00012000
 000000            00000 00000     13 default CSECT                        ↓      ↓                              00013000
                                   14 n   parms pp1,pp2,kw5=z,pp3,kw1=y,pp4,pp5,pp6                            00014000
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00004/C   1
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00005/SYSASM  2
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00006/Z   3
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00007/KPARM00001 ←┬→ 4
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00008/KPARM00005 ←┘
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00010/PPARM00006 ←┬→ 5
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00011/PPARM00002 ←┘
                                   15     END                                                                  00015000
```

Figure 47. Sample macro parameter messages

Notes to Figure 47:

1	SET symbol, and related message
2	System variable symbol, and related message
3	The name field parameter, and related message
4	Keyword parameters, and related messages
5	Positional parameters, and related messages

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 normally 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 the errors in the statement are flagged. Occasionally, duplicate error messages occur. This is a normal result of the error detection process.

Figure 48 on page 145 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***  statement number, severity code, message`

Unless the severity code is shown as an asterisk (*), 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 R6.0  2008/07/11 17.48
                                      1 *********************************************************************** DIA00010
                                      2 *             SAMPLE ERROR DIAGNOSTIC MESSAGES                        * DIA00020
                                      3 *          IN SOURCE PROGRAM (OPEN CODE) AND GENERATED BY MACRO CALLS * DIA00030
                                      4 *********************************************************************** DIA00040
000000                00000 0003C     5 A        CSECT                                                          DIA00050
000000 0000 0000            00000     6          STM   14,U2,12(13(                                             DIA00060
** ASMA044E Undefined symbol - U2
** ASMA029E Incorrect register specification - U2
** ASMA179S Delimiter error, expected right parenthesis - (
** ASMA435I Record 6 in DIAGMSG  ASSEMBLE A1 on volume: ADISK
000004 05C0                           7          BALR  12,0                                                     DIA00070
                 R:C  00006           8          USING *,12                                                     DIA00080
000006 0000 0000            00000     9          ST    13,SAVE+4                                                DIA00090
** ASMA044E Undefined symbol - SAVE
** ASMA435I Record 9 in DIAGMSG  ASSEMBLE A1 on volume: ADISK
                                     10          OPEN   (CRDIN,(INPUT),CRDOUT,(OUTPUT)                          DIA00100
** ASMA088E Unbalanced parentheses in macro call operand - OPEN /(CRDIN,(INPUT),CRDOUT,(OUTPUT)
** ASMA435I Record 323 in OSMACRO  MACLIB   S2(OPEN) on volume: MNT190
00000A 0700                          11+         CNOP  0,4                          ALIGN LIST TO FULLWORD      01-OPEN
00000C 4110 C00E            00014    12+         LA    1,*+8                        LOAD R1 W/LIST ADR @V6PXJRU 01-OPEN
000010 47F0 C00E            00014    13+         B     *+4                          BRANCH AROUND LIST @V6PXJRU 01-OPEN
** ASMA254I *** MNOTE ***            14+     12,***  IHB001  DCB OPERAND REQ'D-NOT SPECIFIED                    02-IHBER
                                     15          DROP  11                                                       DIA00110
** ASMA045W Register or label not previously used - 11
** ASMA435I Record 11 in DIAGMSG  ASSEMBLE A1 on volume: ADISK

                                     16 *********************************************************************** DIA00120
                                     17 *     EDITING AND GENERATION ERRORS AND MNOTES FROM A LIBRARY MACRO   * DIA00130
                                     18 *********************************************************************** DIA00140
                                     19          LOADR REG1=10,REG2=8,WOOSHA,SUMA                               DIA00150
000014 58A0 C02E            00034    20+         L     10,WOOSHA                                                01-LOADR
000018 5880 C032            00038    21+         L     8,SUMA                                                   01-LOADR
                                     22          LOADR REG1=25,REG2=8,WOOSHA,MAINY                              DIA00160
00001C 0000 0000            00000    23+         L     25,WOOSHA                                                01-LOADR
** ASMA029E Incorrect register specification - 25
** ASMA435I Record 5 in TEST     MACLIB   A1(LOADR) on volume: ADISK
000020 0000 0000            00000    24+         L     8,MAINY                                                  01-LOADR
** ASMA044E Undefined symbol - MAINY
** ASMA435I Record 6 in TEST     MACLIB   A1(LOADR) on volume: ADISK
                                     25          LOADR REG2=10,SUMA,MAINY                                       DIA00170
** ASMA254I *** MNOTE ***            26+     36,YOU LEFT OUT THE FIRST REGISTER                                 01-LOADR

                                     27 *********************************************************************** DIA00180
                                     28 *     SAMPLE IN-LINE MACRO DEFINITION                                 * DIA00190
                                     29 *********************************************************************** DIA00200
                                     30          MACRO                                                          DIA00210
                                     31 &NAME    LOADR &REG1=,&REG2=,&OP1,&OP2                                  DIA00220
                                     32 &R(1)    SETA  &REG1,&REG2                                              DIA00230
                                     33          AIF   (T'&REG1 EQ 'O').ERR                                     DIA00240
                                     34          L     &R(1),&OP1                                               DIA00250
                                     35          L     &R(2),&OP2                                               DIA00260
                                     36          MEXIT                                                          DIA00270
                                     37 .ERR     MNOTE 36,'YOU LEFT OUT THE FIRST REGISTER'                     DIA00280
                                     38          MEND                                                           DIA00290

                                     39 *********************************************************************** DIA00300
                                     40 *     SAMPLE MACRO CALLS WITH GENERATION ERRORS AND MNOTES            * DIA00310
                                     41 *********************************************************************** DIA00320
                                                                                                               Page    4
  Active Usings: A+X'6',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                     42          LOADR REG1=10,REG2=8,WOOSHA,SUMA                               DIA00330
000024 58A0 C02E            00034    43+         L     10,WOOSHA                                                01-00034
000028 5880 C032            00038    44+         L     8,SUMA                                                   01-00035
                                     45          LOADR REG1=25,REG2=8,WOOSHA,&MAINY                             DIA00340
** ASMA003E Undeclared variable symbol; default=0, null, or type=U - OPENC/MAINY
** ASMA435I Record 34 in DIAGMSG  ASSEMBLE A1 on volume: ADISK
00002C 0000 0000            00000    46+         L     25,WOOSHA                                                01-00034
** ASMA029E Incorrect register specification - 25
** ASMA435I Record 25 in DIAGMSG  ASSEMBLE A1 on volume: ADISK
000030 0000 0000            00000    47+         L     8,                                                       01-00035
** ASMA074E Illegal syntax in expression -
** ASMA435I Record 26 in DIAGMSG  ASSEMBLE A1 on volume: ADISK
                                     48          LOADR REG2=8,SUMA,MAINY                                        DIA00350
** ASMA254I *** MNOTE ***            49+     36,YOU LEFT OUT THE FIRST REGISTER                                 01-00037
000034                               50 WOOSHA   DS    F                                                        DIA00360
000038                               51 SUMA     DS    F                                                        DIA00370
                                     52          END                                                            DIA00380
```

Figure 48. 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 data set 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 F, “High Level Assembler messages,” on page 297 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, (the sum of binary or decimal options)
```

The options 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 E, “MHELP sample macro trace and dump,” on page 289 for complete details about this facility.

Chapter 7. Assembling your program on z/OS

This chapter describes how to invoke the assembler on z/OS. It describes:
- The input to the assembler.
- The output from the assembler.
- How to invoke the assembler on z/OS 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 *HLASM 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 79 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 GOFF assembler option, the assembler produces a generalized 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 z/OS binder. See Appendix B, “Object deck output,” on page 219 for the format of the object module. The format of the generalized object format module is described in *z/OS MVS Program Management: Advanced Facilities*.

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 5.

Invoking the assembler on z/OS

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 on z/OS is to code JCL that uses the cataloged procedure shown in Figure 49 on page 150.

© Copyright IBM Corp. 1992, 2013

```
//jobname  JOB accountno,progrname,MSGLEVEL=1    1
//stepname EXEC ASMAC                            2
//SYSIN    DD   *                                3
⋮
Assembler source statements
⋮
/*
```

Figure 49. 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 172; 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 50. 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 35.

Using your own JCL: The cataloged procedures might not comply with your data processing requirements. Figure 51 on page 151 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
//SYSPRINT DD   SYSOUT=A
//SYSTERM  DD   SYSOUT=A
//ASMAOPT  DD   DSNAME=PROG.OPTIONS,DISP=OLD
//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 51. JCL for assembly

Refer to "Bibliography" on page 399 for a list of JCL documents that describe additional techniques for specifying job control statements and overriding cataloged procedures.

Invoking the assembler on TSO

On TSO, you can use TSO commands, command lists (CLISTs), REXX EXECs, or ISPF to assemble your program. Figure 52 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(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(32760) LRECL(32756) RECFM(V B)
    REUSE CATALOG
READY
ALLOCATE FILE(SYSIN) DATASET(PROG.ASSEMBLE) SHR REUSE
READY
ALLOCATE FILE(ASMAOPT) DATASET(PROG.OPTIONS) SHR REUSE
READY
CALL *(ASMA90) 'ADATA,LIST(133),OBJECT,TERM'
:
Assembler listing and messages
:
READY
FREE FILE(SYSADATA,SYSPRINT,SYSTERM,SYSLIN,SYSIN)
READY
```

Figure 52. Assembling on TSO

You can enter ALLOCATE commands in any order; however, you must enter all of them before you start the assembly. Table 23 shows the data sets you must allocate when you specify particular assembler options.

Table 23. Assembler options and data sets required

Option Specified	Data Sets Required
Any	SYSUT1 and SYSIN
LIST	SYSPRINT
TERM	SYSTERM
OBJECT	SYSLIN
DECK	SYSPUNCH

Table 23. Assembler options and data sets required (continued)

Option Specified	**Data Sets Required**
ADATA	SYSADATA, WORKFILE, and SYSUT1

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

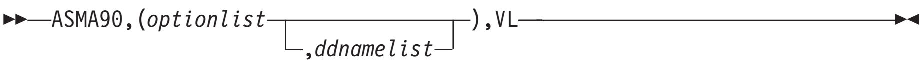
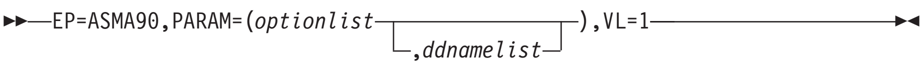
You can invoke High Level Assembler from a running program using the CALL, LINK, XCTL, or ATTACH system macro instructions.

Note: If the program that invokes the High Level Assembler is APF-authorized, then ensure that the High Level Assembler product resides in an Authorized Program Facility (APF) authorized library.

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. Table 24 shows how to invoke the assembler dynamically.

Table 24. Invoking the assembler dynamically

Name	**Operation**	**Operand**
symbol	`CALL`	
	`LINK` or `ATTACH`	

`ASMA90`
The load module name and entry point to invoke the assembler. ASMA90 must be invoked in 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 (*optionlist*) contains the address of the option list. The second word (*ddnamelist*) 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 spaces should appear in the list, except within the string specified for the EXIT or SYSPARM options, providing the string is enclosed within apostrophes.

`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 eight bytes must be left-aligned and padded to eight bytes with spaces. If an alternative ddname is omitted, the standard name is assumed. If the name is omitted within the list, the eight-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the eight-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
17	Not applicable
18	Not applicable
19	Not applicable
20	ASMAOPT

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. However, 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 53. Sample program to call the assembler dynamically

Batch assembling

A sequence of separate assembler programs can be assembled with a single invocation of the assembler when the BATCH option is specified. The object programs produced from this assembly can 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 is assembled with the next program in the sequence. If the END statement is omitted from the last program in the sequence, an END statement is generated by the assembler. When the BATCH option is specified, and more than one assembler program exists in the sequence, a summary line is printed after the final assembly listing. The summary line summarizes the number of assembler programs in the sequence, and the highest return code from the assemblies.

If you want to produce more than one program module, you must either separate the object modules, or write a NAME linkage editor control statement for each load module. The NAME statement must be written at the end of 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 54 on page 155:

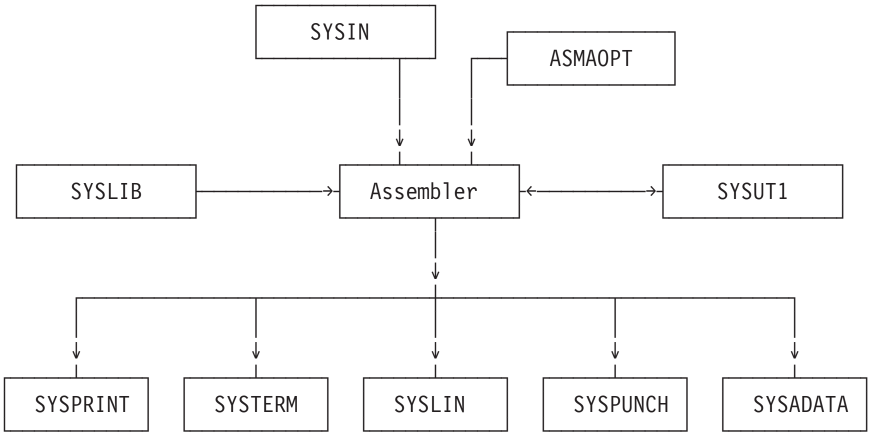


Figure 54. High Level Assembler Files

You can override the ddnames during installation or when invoking the assembler dynamically (see "Invoking the assembler dynamically" on page 152).

High Level Assembler requires this data set:

SYSIN
An input data set containing the source statements to be processed.

In addition, these data sets might be required:

ASMAOPT
An input data set containing an assembler option list.

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 normally for the linkage editor, loader, or binder (only if the OBJECT option is in effect).

SYSADATA
A data set containing associated data output (only if the ADATA option is in effect).

SYSUT1
Assembler work file (only if the WORKFILE option is in effect).

The data sets listed above are described in "Specifying the source data set: SYSIN" on page 157. Table 25 on page 156 describes the characteristics of these data sets, including the characteristics set by the assembler and those you can override. The standard ddname that defines the data set appears as the heading for each data set description.

Table 25. Assembler data set characteristics

Data Set	Access Method	Logical Record Length (LRECL)	Block Size (BLKSIZE)	Record Format (RECFM)
SYSIN	QSAM	80	5	9
ASMAOPT	QSAM	12	7	Fixed/Variable
SYSLIB	BPAM	80	6	9
SYSPRINT	QSAM	1	7 8	10
SYSTERM	QSAM	2	5 8	11
SYSPUNCH	QSAM	80	5	4
SYSLIN	QSAM	3	5	4
SYSADATA	QSAM	32756	32760 or greater 8	VB
SYSUT1	QSAM		32760	Fixed

Notes to Table 25:

1 If you specify EXIT(PRTEXIT) and the user exit specifies the logical record length, the logical record length returned is used, unless the SYSPRINT data set has a variable-length record format in which case the LRECL used is 4 bytes greater than the value returned by the exit. 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, or 137 if the record format is variable-length.

The minimum record length allowed for SYSPRINT is 121, and the maximum allowed is 255. If the record format is variable-length, the LRECL should be at least 125 or 137 depending on the LIST option.

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 must be 80. If you specify the GOFF option the object module can be generated with either fixed-length records of 80 bytes, or variable-length records up to BLKSIZE of 32720.

Hierarchical File System: If you want to copy the object data set to a file in a Hierarchical File System, for example under UNIX System Services, the object data set must be created with fixed-length records.

4 Both fixed and variable formats are supported; the default is fixed.

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* document, or *z/OS DFSMS Program Management*, for the block size requirements of SYSPUNCH and SYSLIN, if you use them as input to the linkage editor, or to the z/OS 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 The blocksize must be equal to or a multiple of the record length if the record format is fixed. If the record format is variable the blocksize must be at least 4 bytes greater than the record length.

8 High Level Assembler supports z/OS System-Determined Blocksize (SDB) for all output data sets except SYSLIN and SYSPUNCH.

System-Determined Blocksize is applicable when all the following conditions are true:
- The operating system is MVS/ESA with an MVS 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, z/OS 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.

9 Set by the assembler to F (or FB) if necessary.

10 Both fixed and variable formats are supported; the default is fixed. 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 modified according to the ASA option.

11 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.

12 The minimum record length allowed for ASMAOPT is 5 if the record format is variable length or 1 if the record format is fixed length. The maximum record length allowed is 32756 if the record format is variable length or 32760 if the record format is fixed length.

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; that is, 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 the option file: ASMAOPT

Define the data set that contains the options with the ASMAOPT DD statement:

```
//ASMAOPT DD   DSN=datasetname,DISP=SHR
```

This data set contains the option list for the assembler.

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 can be concatenated with SYS1.MACLIB. The two libraries must have the same logical record length (80 bytes), but the blocking factors can 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
```

On 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 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=SYSALLDA,
//              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=SYSALLDA,
//              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.

The recommended minimum LRECL for the SYSADATA data set is 8188.

Specifying the utility data data set: SYSUT1

The utility data file is used to store internal structures when the allocated memory is exhausted

Return codes

High Level Assembler issues return codes for use with the IF job control statement and the COND parameter of the JOB and EXEC job control language statements. The IF statement and the COND parameter enable 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 in "FLAG" on page 48. See Appendix F, "High Level Assembler messages," on page 297 for a listing of the assembler errors and their severity codes.

Chapter 8. Linking and running your program on z/OS

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 the binder to convert it into executable machine code.

The program management binder

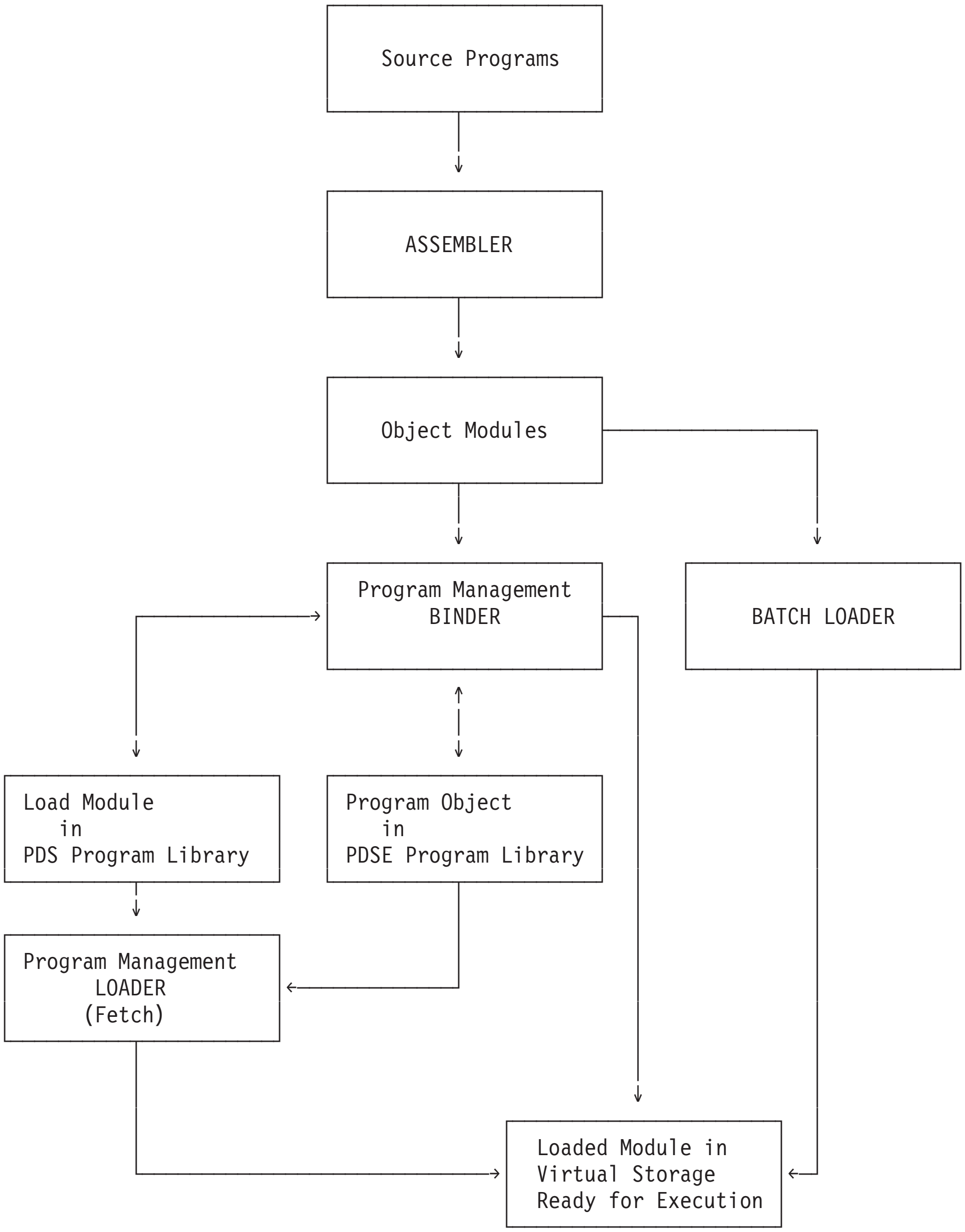
The binder converts object modules into an executable program unit that can either be read directly into virtual storage for execution, or stored in a program library. Executable program units can either be load modules, or *program objects*. You can use the binder to:

- 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 *binder*, unless otherwise stated.

For more information, see the *z/OS MVS Program Management: User's Guide and Reference*.

© Copyright IBM Corp. 1992, 2013


Source Programs
ASSEMBLER
Object Modules
Program Management BINDER
BATCH LOADER
Load Module in PDS Program Library
Program Object in PDSE Program Library
Program Management LOADER (Fetch)
Loaded Module in Virtual Storage Ready for Execution

Components are shown in uppercase.

←→ = Two-way relationship.
Indicates a component can
produce that structure as
output or accept it as input.

Figure 55. Using the program management components

The loader

The loader component of z/OS, and the batch loader component of z/OS perform the same task. Given this, from here on the batch loader is referred to as the *loader*, unless otherwise stated.

The loader combines the basic editing and loading services, that can also be provided by the binder 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 binder.

Creating a load module

The binder 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 binder can process more than one object module, or load module, or program object, and convert them into one or more load modules, or program objects, by using the NAME control statement. See "Batch assembling" on page 154 for an example that uses the NAME control statement.

Creating a load module on z/OS

Figure 56 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=SYSALLDA,
//              DISP=(NEW,PASS),SPACE=(subparms)
//SYSLIB   DD   DSN=dsname,DISP=SHR
//SYSUT1   DD   UNIT=SYSALLDA,SPACE=(subparms)
//SYSLIN   DD   DSN=MYOBJ,DISP=SHR
```

Figure 56. Sample job control for creating a load module

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 172.

Creating a load module on TSO

You can invoke the binder on TSO (Time Sharing Option) with the LINK and LOADGO commands.

The LINK command creates a load module and saves it in a PDS or it creates a program object and saves that in a PDSE.

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 *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 binder 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 binder options using the TSO LINK command” on page 168.

Binder options are discussed in “Binder processing options” on page 166.

For more information about using the LINK and LOADGO commands, see the *z/OS TSO/E Command Reference*.

Input to the binder

Your input to the binder can be:
- One or more object modules.
- Binder 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 binder 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, binder control statements, and linked program modules.

You specify the primary input data set with the SYSLIN DD statement.

Secondary input to the binder 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 call library* process.

An automatic call library contains modules that you can use as secondary input to the binder to resolve external symbols left undefined after all primary input has been processed.

The automatic call library is in the form of:

- Libraries containing object modules, with or without binder control statements.
- Libraries containing linked program modules.

Secondary input for the binder 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, 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 sets (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 binder processing

You need the following data sets for binder processing. Others might 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. Table 26 summarizes the data sets that you need for linking.

Table 26. 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 57 on page 166 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 57. INCLUDE and LIBRARY control statements

Data sets you specify on the INCLUDE statement are processed as the binder 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.

For more information, see the *z/OS MVS Program Management: User's Guide and Reference*.

Output from the binder

SYSLMOD and SYSPRINT are the data sets used for binder output. The output varies depending on the options you select, as shown in Table 27.

Table 27. Options for controlling binder output

To Get This Output	Use This Option
A map of the program modules generated by the binder.	MAP
A cross-reference list of data variables	XREF
Informative messages	Default
Diagnostic messages	Default
Listing of the binder 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.

Binder processing options

Binder options can be specified in one of these ways:
- In your JCL.
- When you invoke the LINK or LOADGO command on TSO.

Table 28 describes some of these options.

Table 28. Link processing options

Option	Action	Comments
LET	Lets you specify the severity level of an error, to control whether the binder marks the program module as non-executable.	The LET option is used differently between the linkage editor and the binder.

Table 28. Link processing options (continued)

Option	Action	Comments
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 binder does not search for library members to resolve external references.	If you specify NCAL, you do not need to use the LIBRARY statement, and you do not need to supply the SYSLIB DD statement.
RENT NORENT	The RENT option indicates to the binder 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 reentrant. NORENT indicates that the object module is not reentrant.	The assembler RENT option and binder RENT option are independent of each other. NORENT is the default binder 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 168.
RMODE 24 | ANY	Use RMODE (residence mode) to override the default RMODE attribute established by the assembler.	See "AMODE and RMODE attributes" on page 168.
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 binder options using the TSO LINK command" on page 168.

Specifying binder options through JCL

In your link JCL, use the PARM statement to specify options:

```
PARM=(binder-options)
PARM.stepname=('binder-options')
```

binder-options
A list of binder options (see Table 28 on page 166). Separate the options with commas.

stepname
The name of the step in the cataloged procedure that contains the PARM statement.

Figure 58 on page 168 shows the job control to pass PARM.C and PARM.L options.

```
//*  ASSEMBLE USING ASMACL
//         JCLLIB ORDER=(ASM.SASMSAM1)
//HLASM  EXEC ASMACL,PARM.C='NODECK,OBJ',
//       PARM.L='LIST,LET,XREF,MAP'
//STEPLIB  DD DISP=SHR,DSN=ASM.SASMMOD1
//C.SYSIN  DD *
       BR   14     < ASSEMBLER PROGRAM GOES HERE
       END
/*
```

Figure 58. Job control to pass PARM.C and PARM.L options

For more information, see the *HLASM Installation and Customization Guide*.

Specifying binder options using the TSO LINK command

You specify binder 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 MAP 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 binder is displayed at your terminal. NOPRINT suppresses any messages.

AMODE and RMODE attributes

Every program that runs in z/OS 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.

z/OS 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.

z/OS uses the RMODE attribute to determine whether a program must be loaded into virtual storage below 16 MB, or can reside anywhere in virtual storage (above or below 16 MB).

Valid AMODE and RMODE specifications are:

Attribute	Meaning
AMODE=24	24-bit addressing mode
AMODE=31	31-bit addressing mode
AMODE=64	64-bit addressing mode
AMODE=ANY	Either 24-bit or 31-bit addressing mode
RMODE=24	The module must reside in virtual storage below 16 MB. 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 do not 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)`
  - `LOADGO PROGRAM1 AMODE(31) RMODE(ANY)`

You can also use binder control statements to override the default AMODE and RMODE values.

Detecting binder errors

The binder produces a listing in the data set defined by the SYSPRINT DD statement. The listing includes any informational or diagnostic messages issued by the binder. Check the load map to make sure that all the modules you expected were included.

For more information, see the *z/OS MVS Program Management: User's Guide and Reference*.

Running your assembled program

When you have 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 172.

Running your assembled program in batch

Figure 59 shows the general job control to run your program in batch.

```
//stepname EXEC PGM=progname[,PARM='user-parameters']
//STEPLIB  DD   DSN=library.dsname,DISP=SHR
//ddname   DD   (parameters for user-specified data sets)
⋮
```

Figure 59. General job control to run a program on z/OS

For more information, see the *HLASM Installation and Customization Guide*.

Running your assembled program on TSO

You use the CALL command to run your program on TSO, as follows:

```
CALL 'JRL.LIB.LOAD(PROGRAM1)'
```

If you omit the descriptive qualifier (LOAD) and the member name (PROGRAM1), the system assumes LOAD and TEMPNAME. If your program module is in the data set JRL.LIB.LOAD(TEMPNAME), and your TSO userid is JRL, enter:

```
CALL LIB
```

Chapter 9. z/OS system services and programming considerations

This chapter describes some of the z/OS 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 cataloged procedures.
- Overriding statements in cataloged 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 *z/OS DFSMSdfp Utilities*.

Figure 60 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 60. Macro library addition procedure

Notes to Figure 60:

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.

© Copyright IBM Corp. 1992, 2013

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 either 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 on z/OS" on page 149 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 61 on page 173 shows the statements that make up the ASMAC procedure.

```
//ASMAC    PROC
//*
//********************************************************************
//*   Licensed Materials - Property of IBM                           *
//*                                                                  *
//*   5696-234   5647-A01                                            *
//*                                                                  *
//*   (C) Copyright IBM Corp. 1992, 2008. All Rights Reserved.       *
//*                                                                  *
//*   US Government Users Restricted Rights - Use,                   *
//*   duplication or disclosure restricted by GSA ADP                *
//*   Schedule Contract with IBM Corp.                               *
//*                                                                  *
//********************************************************************
//*                                                                  *
//*   ASMAC                                                          *
//*                                                                  *
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER AND CAN BE USED     *
//* TO ASSEMBLE PROGRAMS.                                            *
//*                                                                  *
//********************************************************************
//C        EXEC PGM=ASMA90
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR                                        1
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),                    2
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*                                                        3
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),                         4
//             UNIT=SYSALLDA,DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FB,BUFNO=1)
```

Figure 61. Cataloged procedure for assembly (ASMAC)

Notes to Figure 61:

1 This statement identifies the macro library data set. The data set name SYS1.MACLIB is an IBM designation.

2 This statement specifies the assembler work data set. The device class name used here, SYSALLDA, 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 SYSALLDA.

3 This statement defines the standard system output class, SYSOUT=*, as the destination for the assembler listing.

4 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 binder 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 binder as shown in the example. This additional input can be binder 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 binder control statements
/*
```

//L.SYSIN is necessary only if the binder is to combine modules or read editor control information from the job stream.

Figure 62 shows the statements that make up the ASMACL procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMACL   PROC
//*
//********************************************************************
//*   Licensed Materials - Property of IBM                           *
//*                                                                  *
//*   5696-234   5647-A01                                            *
//*                                                                  *
//*   (C) Copyright IBM Corp. 1992, 2008. All Rights Reserved.       *
//*                                                                  *
//*   US Government Users Restricted Rights - Use,                   *
//*   duplication or disclosure restricted by GSA ADP                *
//*   Schedule Contract with IBM Corp.                               *
//*                                                                  *
//********************************************************************
//*                                                                  *
//*     ASMACL                                                       *
//*                                                                  *
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER, LINK-EDITS THE     *
//* NEWLY ASSEMBLED PROGRAM.                                         *
//*                                                                  *
//********************************************************************
//*
//C        EXEC PGM=ASMA90
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),                     1
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),
//             UNIT=SYSALLDA,DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FB,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=SYSALLDA,SPACE=(CYL,(1,1,1)),                  5
//             DSN=&&GOSET(GO)
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(1024,(120,120),,,ROUND),                     6
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*                                                         7
```

Figure 62. Cataloged procedure for assembling and linking (ASMACL)

Notes to Figure 62:

1 In this procedure, the SYSLIN DD statement describes a temporary data set, the object module, which is passed to the binder.

2 This statement runs the binder. The binder options in the PARM field cause the binder to produce a cross-reference table, a module map, and a list of all control statements processed by the binder. The NCAL option suppresses the automatic library call function of the binder.

3 This statement identifies the binder 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 binder from the input stream (object decks, binder control statements, or both) with the input from the assembler.

5 This statement specifies the binder 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 178. If you want to retain the output of the binder, 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 binder.

7 This statement identifies the standard output class as the destination for the binder 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 binder 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 binder control statements
⋮
/*
//G.ddname        DD   (parameters)
//G.ddname        DD   (parameters)
//G.ddname        DD   *
⋮
program input
⋮
/*
```

//L.SYSIN is necessary only if the binder is to combine modules or read binder control information from the job stream.

//G.ddname statements are included only if necessary.

Figure 63 on page 176 shows the statements that make up the ASMACLG procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMACLG  PROC
//*
//********************************************************************
//*   Licensed Materials - Property of IBM                           *
//*                                                                  *
//*   5696-234   5647-A01                                            *
//*                                                                  *
//*   (C) Copyright IBM Corp. 1992, 2008. All Rights Reserved.       *
//*                                                                  *
//*   US Government Users Restricted Rights - Use,                   *
//*   duplication or disclosure restricted by GSA ADP                *
//*   Schedule Contract with IBM Corp.                               *
//*                                                                  *
//********************************************************************
//*                                                                  *
//*     ASMACLG                                                      *
//*                                                                  *
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER, LINK-EDITS THE     *
//* NEWLY ASSEMBLED PROGRAM AND RUNS THE PROGRAM AFTER               *
//* THE LINK-EDIT IS ACCOMPLISHED.                                   *
//*                                                                  *
//********************************************************************
//*
//C       EXEC PGM=ASMA90
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),
//             UNIT=SYSALLDA,DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FB,BUFNO=1)
//L       EXEC PGM=HEWL,PARM='MAP,LET,LIST',COND=(8,LT,C)                      1
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DISP=(,PASS),UNIT=SYSALLDA,SPACE=(CYL,(1,1,1)),                  2
//             DSN=&&GOSET(GO)
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(1024,(120,120),,,ROUND),
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//G       EXEC PGM=*.L.SYSLMOD,COND=((8,LT,C),(8,LT,L))                        3
```

Figure 63. Cataloged procedure for assembly, link, and run (ASMACLG)

Notes to Figure 63:

1 The LET binder option specified in this statement causes the binder to mark the program module as executable, even if errors are encountered during processing.

2 The output of the binder 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 binder 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 64 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 64.

```
//ASMACG   PROC
//*
//*******************************************************************
//*   Licensed Materials - Property of IBM                          *
//*                                                                 *
//*   5696-234   5647-A01                                           *
//*                                                                 *
//*   (C) Copyright IBM Corp. 1992, 2008. All Rights Reserved.      *
//*                                                                 *
//*   US Government Users Restricted Rights - Use,                  *
//*   duplication or disclosure restricted by GSA ADP               *
//*   Schedule Contract with IBM Corp.                              *
//*                                                                 *
//*******************************************************************
//*                                                                 *
//*     ASMACG                                                      *
//*                                                                 *
//* THIS PROCEDURE RUNS THE HIGH LEVEL ASSEMBLER AND WILL USE       *
//* THE LOADER PROGRAM TO RUN THE NEWLY ASSEMBLED PROGRAM.          *
//*                                                                 *
//*******************************************************************
//*
//C        EXEC PGM=ASMA90
//SYSLIB   DD  DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD  DSN=&&SYSUT1,SPACE=(4096,(120,120),,,ROUND),
//             UNIT=SYSALLDA,DCB=BUFNO=1
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSN=&&OBJ,SPACE=(3040,(40,40),,,ROUND),
//             UNIT=SYSALLDA,DISP=(MOD,PASS),
//             DCB=(BLKSIZE=3040,LRECL=80,RECFM=FB,BUFNO=1)
//G        EXEC PGM=LOADER,PARM='MAP,LET,PRINT',COND=(8,LT,C)         1
//SYSLIN   DD  DSN=&&OBJ,DISP=(OLD,DELETE)                            2
//         DD  DDNAME=SYSIN
//SYSLOUT  DD  SYSOUT=*                                                3
```

Figure 64. Cataloged procedure for assembly and running using the loader (ASMACG)

Notes to Figure 64:

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 binder, and the LET option is the same as for the binder.

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 cataloged 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=, ...).

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 61 on page 173), 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 63 on page 176), you might want to suppress the assembler listing, and add the COND= parameter to the EXEC statement that invokes the binder. 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 binder job step //L is suppressed if the return code issued by the assembler (step C) is 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 62 on page 174) 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 binder 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 on z/OS 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, on completion, restore these same values to the general registers. 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.

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 *z/OS MVS Programming: Assembler Services Guide*.

```
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 *z/OS MVS Programming: Assembler Services Reference IAR-XCT*.

Accessing execution parameters

You access information in the PARM field of an EXEC statement by referring 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 65 shows how the PARM field information is structured.

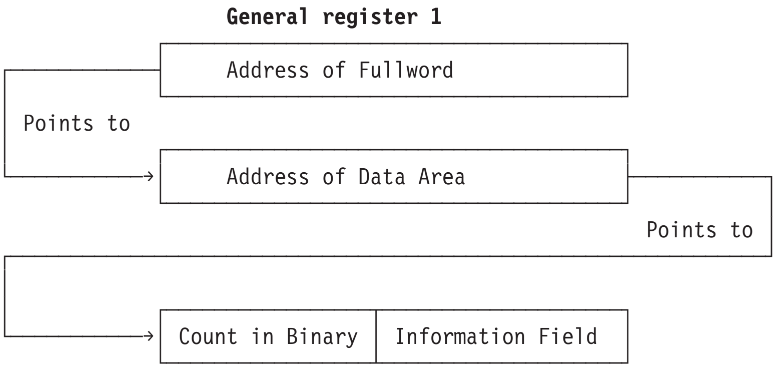


Figure 65. Access to PARM field

Object program 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 z/OS 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 *z/OS MVS Programming: Assembler Services Reference* for details about linkage conventions and the CALL system macro instruction.

Figure 66 on page 182 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 or output operations are performed 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 66. Sample assembler linkage statements for calling subprograms

Modifying program modules

If the editing functions of the binder 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 binder. Otherwise, the first byte of the first control section processed by the binder 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 these 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 binder 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 *z/OS MVS Program Management: User's Guide and Reference*.

Chapter 10. Assembling your program on CMS

This chapter describes how to invoke the assembler on 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 on CMS.
- How to assemble multiple source programs using the BATCH option.
- Special options for invoking the assembler on CMS.
- The data sets used by the assembler.
- The assembler return codes.
- Special diagnostic messages when invoking the assembler on CMS.

To use this section effectively, you should be familiar with the assembler language described in the *HLASM 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 *HLASM 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 79 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 5.

Accessing the assembler

To access the High Level Assembler on 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).

© Copyright IBM Corp. 1992, 2013

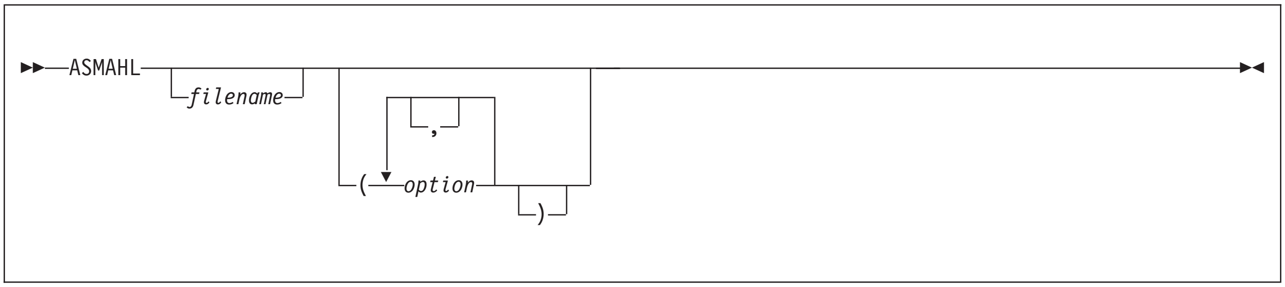
After you linked to the 194 disk as 198, you accessed the 198 disk as disk B on your system. After you have access to the product disk, you can invoke the assembler using the ASMAHL command (see "Invoking the assembler on CMS").

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 about the CP LINK and ACCESS commands, see the applicable *CP Command Reference* for your VM environment, as listed under "Bibliography" on page 399.

Invoking the assembler on CMS

Use the ASMAHL command to invoke and control assembly of assembler source programs on CMS.

The format of the ASMAHL command is:



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 188 for details on each of these methods.

option Represents one or more assembler options, separated by a space or comma, that you want in effect during assembly. These assembler options are equivalent to the options you specify on the PARM parameter of an EXEC job control statement, if you are invoking the assembler on z/OS.

A complete list and discussion of assembler options can be found under Chapter 3, "Controlling your assembly with options," on page 35.

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 on CMS; see "Controlling your assembly" on page 185.

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 assemble 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 executable modules are to be produced, you must either separate the object modules, or write a NAME linkage editor control statement for each load module. The NAME statement must be written at the end of the object module. Figure 67 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
```

Figure 67. Example of creating two load modules on CMS

If separate TEXT files are required, you must either manually separate the BATCH- produced object modules into separate TEXT files, or issue two separate ASMAHL commands for separate source modules.

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 in Chapter 3, “Controlling your assembly with options,” on page 35.

On CMS, there are additional options that can be specified. These are described in Chapter 3, “Controlling your assembly with options,” on page 35, and consist of:

ERASE
Deletes LISTING, TEXT, and SYSADATA files before the assembly begins.

LINECOUNT
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 files as shown in Figure 68.

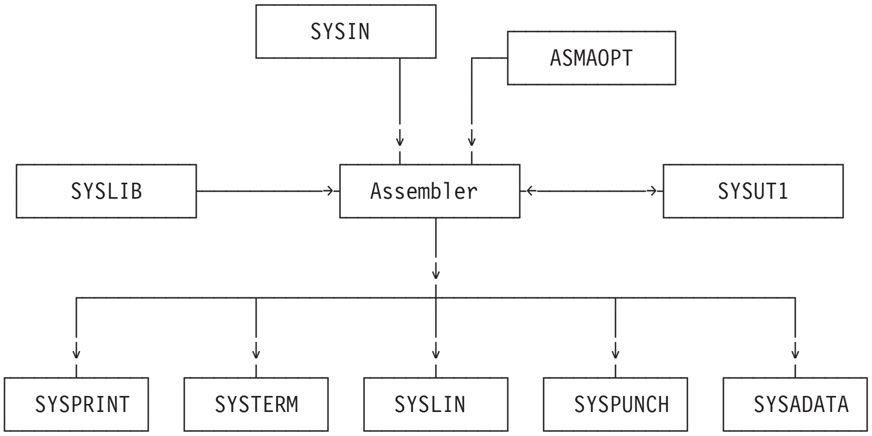


Figure 68. High Level Assembler Files

The ddnames can be overridden during installation.

High Level Assembler requires the following files:

SYSIN
An input file containing the source statements to be processed.

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, normally for punching (only if the DECK option is in effect).

SYSLIN
A file containing object module output normally for the linkage editor (only if the OBJECT option is in effect).

SYSADATA
A file containing associated data output (only if the ADATA option is in effect).

SYSUT1
Assembler work file (only if the WORKFILE option is in effect).

ASMAOPT
An input data set containing an assembler option list.

The files listed above are described in the text following Table 29. The characteristics of these files, those set by the assembler and those you can override, are shown in Table 29.

Table 29. Assembler file characteristics

File	Access Method	Logical Record Length (LRECL)	Block Size (BLKSIZE)	Record Format (RECFM)
SYSIN	QSAM	80	4	7 10
ASMAOPT	QSAM	11	6	Fixed/Variable
SYSLIB	BPAM	80	5	7 10
SYSPRINT	QSAM	1	6	8 10
SYSTERM	QSAM	2	4	9 10
SYSPUNCH	QSAM	12	4	3
SYSLIN	QSAM	12	4	3
SYSADATA	QSAM	32756	32760 or greater	VB
SYSUT1	QSAM		32768	Fixed

Notes to Table 29:

1 If you specify EXIT(PRTEXIT) and the user exit specifies the logical record length, the logical record length returned is used, unless the SYSPRINT data set has a variable-length record format in which case the LRECL used is 4 bytes greater than the value returned by the exit. If EXIT(PRTEXIT) has not been specified 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, or 137 if the record format is variable-length.

The minimum record length allowed for SYSPRINT is 121, and the maximum allowed is 255. If the record format is variable-length, the LRECL should be at least 125 or 137 depending on the LIST option.

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 Both fixed and variable formats are supported; the default is fixed.

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 The blocksize must be equal to or a multiple of the record length if the record format is fixed. If the record format is variable, then the blocksize must be at least 4 bytes greater than the record length.

7 Set by the assembler to F (or FB) if necessary.

8 Both fixed and variable formats are supported; the default is variable. If the FILEDEF command or file 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 FILEDEF command or file label, the record format is modified according to the ASA option.

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 You can specify B, S, or T, or any combination of these.

11 The minimum record length allowed for ASMAOPT is 5. The maximum record length allowed is 32756 if the record format is variable length or 32760 if the record format is fixed length.

12 If you specify the OBJECT option,the logical record length must be 80. If you specify the GOFF option, the object module can be generated with either fixed-length records of 80 bytes, or variable-length records up to a BLKSIZE of 32720.

Specifying the source file: SYSIN

Use one of these methods for specifying your assembler source program:
- Specify the file name 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 File name on the Command Line: Using this method, you specify the file name 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 file type 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 AASMAHL
 (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 file name on the ASMAHL command is optional. If the file name is specified on the ASMAHL command, the file name must match the file name of the file specified on the FILEDEF. Additionally, when using a FILEDEF, the file type need not be ASSEMBLE.

You can read z/OS data sets and z/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 file name 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 file name on the ASMAHL command. The file name 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 file name must be specified on the ASMAHL command. The file name is used as the file name 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 “EXIT” on page 46).

Specify the file name 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 file name specified on the ASMAHL command, or from the FILEDEF for SYSIN, is used as the file name 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 the option file: ASMAOPT

You must issue an ASMAOPT FILEDEF command if you want the assembler to use the options in this file. For example

```
FILEDEF ASMAOPT DISK PROG1 OPTIONS *
```

Specifying macro and copy code libraries: SYSLIB

If you do not 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 do not 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 do not issue SYSPRINT FILEDEF before the ASMAHL command, the ASMAHL command issues the following FILEDEF command:

```
FILEDEF SYSPRINT DISK fn LISTING m1 (RECFM VB BLOCK 13300
```

where *fn* is the file name 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 is on a read/write B disk, the file mode *m*1 is 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 (CMS)” on page 63 for details about the CMS options for SYSPRINT.

Directing assembler messages to your terminal: SYSTERM

If you do not 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 do not 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 file name 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 is on a read/write B disk, the file mode *m*1 is 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 do not 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 32756 BLOCK 32760
```

where *fn* is the file name 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 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 is on a read/write B disk, the file mode *m*1 is set to “B1”.

The recommended minimum LRECL for the SYSADATA data file is 8188.

A FILEDEF command for SYSADATA can be issued before the ASMAHL command is issued to direct the associated data output to a different file.

Specifying the utility data file: SYSUT1

The following FILEDEF command can be used to specify SYSUT1 commands:

```
FILEDEF SYSUT1 DISK fn WORKFILE m1 (RECFM F BLOCK 32768
```

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 "FLAG" on page 48. See Appendix F, "High Level Assembler messages," on page 297 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 non-zero return code.

Two types of messages might be issued:

- Messages that are issued by the assembler (see Appendix F, "High Level Assembler messages," on page 297).
- Messages that are issued by the ASMAHL command processor (see "ASMAHL Command Error Messages (CMS)" on page 346).

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. Table 30 shows the SET EMSG options you can specify, and how they affect the message display.

Table 30. 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 on CMS

There are three ways to run your assembled 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 file name 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 “Bibliography” on page 399.

Using the CMS GENMOD command

When your programs are debugged and tested, you can use the LOAD and INCLUDE commands, 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 file type 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(RLDSAVE
GENMOD PROCESS
```

In this example, CREATE, ANALYZE, and PRINT are TEXT files that you are combining into a module named PROCESS; PROCESS is the file name you are assigning to the module, which has a file type 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 about creating program modules, see the applicable *CMS User′s Guide* for your particular VM environment, as listed under “Bibliography” on page 399.

© Copyright IBM Corp. 1992, 2013

Using the CMS LKED and OSRUN commands

A LOADLIB is another type of library available to you on CMS. LOADLIBs, like MACLIBs and TXTLIBs, are in CMS-simulated partitioned data set 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 file name 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 “Bibliography” on page 399.

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 in one of these situations:

- 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 on to 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 canceled, make sure that stacked input is available if your program requests input from the terminal. For further information about using the CMS batch facility, see the applicable *CMS User's Guide* for your particular VM environment, as listed under “Bibliography” on page 399.

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

This section tells you about the assembler macros supported by CMS, and tells you how to add definitions to a macro library.

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 "Bibliography" on page 399.

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 file name of NEWMAC and file type of MACRO is added to the MACLIB with a file name of MYLIB.

```
MACLIB ADD MYLIB NEWMAC
```

Details of this command are described in the applicable *CMS Command and Macro Reference*, listed under "Bibliography" on page 399.

Operating system programming conventions

Assembler programs executing on 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, on completion, restore these same values to the general registers. 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

© Copyright IBM Corp. 1992, 2013

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 can 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 return is 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 *VM/ESA CMS Application Development Reference for Assembler*.

Passing parameters to your assembler language program

On CMS, you can pass parameters to an assembler language program with 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 eight characters each, and must be separated by spaces.

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 for z/OS in "Accessing execution parameters" on page 181.

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

Chapter 13. Assembling your program on z/VSE

This chapter describes how to invoke High Level Assembler on z/VSE. It describes the job control required to run the assembler, files used by the assembler and return codes. The job control language is described in detail in *z/VSE: System Control Statements*.

Input to the assembler

As input, the assembler accepts a program written in the assembler language as defined in the *HLASM 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.

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. From that file, the object module can be read and processed by the linkage editor or the loader. See Appendix B, “Object deck output,” on page 219 for the format of the object module.

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 5.

Invoking the assembler in batch

The JCL for running an assembly includes:
- A job description.
- Definitions for the files needed.
- A statement to run the assembler.

The following example shows how to run the assembler.

```
// JOB jobname                                                 1
// DLBL IJSYS03,'HLASM.WORK.IJSYS03',0,VSAM,RECSIZE=4096,      2
              RECORDS=(100,50),DISP=(NEW,KEEP),CAT=VSESPUC
// DLBL IJSYSLN,'HLASM.WORK.IJSYSLN',0,VSAM,RECSIZE=322,       3
              RECORDS=(100,50),DISP=(NEW,KEEP),CAT=VSESPUC
// LIBDEF PHASE,SEARCH=(PRD2.PROD)                             4
// LIBDEF SOURCE,SEARCH=(lib.sublib)                           5
// OPTION LINK                                                 6
// EXEC ASMA90,SIZE=ASMA90                                     7
⋮
Assembler source statements
⋮
/*                                                             8
/&                                                             9
```

Figure 69. JCL to assemble a program

1 Identifies the beginning of your job to the operating system. *jobname* is the name you assign to the job.

2 Defines the work file to be used by the assembler. The work file must be on a direct-access storage device. The work file can be a SAM file or a SAM-ESDS file. This statement is not required if IJSYS03 is defined in the System Standard or Partition Standard labels.

3 Defines the SYSLNK file that receives the object records produced from the LINK option. This statement is not required if IJSYSLN is defined in the System Standard or Partition Standard labels.

4 Specifies the sublibrary where the assembler resides.

5 Specifies the sublibraries that are to be searched to locate any macro and copy members, and the ASMAOPT.USER member (if required).

6 Sets the LINK option and the Assembler OBJECT option which causes the assembler to write the object records produced to SYSLNK.

7 Invokes the assembler to 17.9(d)svokes the assemblersbestatearies that thebestate.(not) TJ08-1.2T333(The)-SIZE3(to)-

90017

```
/INPUT
/LOAD    ASMA90,PARM='SIZE(800K)'
/OPTION  NOGO,RESET,DECK,GETVIS=P-240
/FILE    TYPE=ICCF,UNIT=SYSPCH,NAME=ASMOBJ
/INCLUDE ASMPROG
/END
/RUN
```

Figure 70. Entering ICCF commands

Figure 71 on page 202 shows a working example of an ICCF procedure for assembling a program, and generating an object module.

```
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
*  ASMARUN NNNN (OBJ MMMM/*) OPTIONS
*
*  PROCEDURE TO ASSEMBLE A HIGH LEVEL ASSEMBLER PROGRAM
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
&&OPTIONS 0010011
&&IF &&PARMCT NE 0 &&GOTO START
&&TYPE ENTER   NAME (OBJ NAME/*) (OPTIONS)
&&READ &&PARAMS
&&IF &&PARMCT EQ 0 &&EXIT
&&LABEL START
/LIST 1 1 &&PARAM1 &&VARBL5
&&IF &&RETCOD NE *FILE &&GOTO SOUR
&&TYPE *SOURCE MEMBER &&PARAM1 NOT IN LIBRARY OR EMPTY
&&EXIT
&&LABEL SOUR
&&IF &&RETCOD NE *INVALID &&GOTO YESOR
&&TYPE *INVALID PASSWORD OR INVALID ACCESS TO MEMBER &&PARAM1
&&EXIT
&&LABEL YESOR
&&IF &&RETCOD NE *MISSING &&GOTO OKSOR
&&TYPE *ENTER PASSWORD FOR MEMBER &&PARAM1
&&READ &&VARBL5
&&IF &&VARBL5 NE ' ' &&GOTO -START
&&EXIT
&&LABEL OKSOR
&&SET &&VARBL1 &&PARAM1
&&SHIFT 1
&&IF &&PARAM1 NE OBJ &&GOTO NOOBJ
&&SET &&VARBL2 &&PARAM2 ''
&&IF &&VARBL2 EQ '*' &&SET &&VARBL2 ' '
&&IF &&VARBL2 EQ ' ' &&GOTO +INLIB
/LIST 1 1 &&VARBL2
&&IF &&RETCOD  NE *FILE &&GOTO OVERW
/INP NOPROMPT
DUMMY RECORD TO CREATE A MEMBER FOR 'ASMARUN' PROCEDURE OUTPUT
/SAVE &&VARBL2
&&IF &&RETCOD NE *LIBRARY &&GOTO INLIB
&&TYPE *LIBRARY DIRECTORY FULL
&&EXIT
&&LABEL OVERW
&&TYPE *MEMBER &&VARBL2 ALREADY EXISTS. OVERWRITE? (Y/N)
&&READ &&VARBL4
&&IF &&VARBL4 EQ 'Y' &&GOTO INLIB
&&TYPE *NO ASSEMBLY - TRY AGAIN WITH ANOTHER NAME
&&EXIT
&&LABEL INLIB
&&SHIFT 1
&&SHIFT 1
&&LABEL NOOBJ
/INP NOPROMPT
&/LOAD ASMA90 PARM='&&PARAM1,&&PARAM2,&&PARAM3,&&PARAM4,&&PARAM5'
/OPTION NOGO RESET GETVIS=P-240
&&IF &&VARBL2 NE ' ' /FILE TYPE=ICCF,UNIT=SYSPCH,NAME=&&VARBL2
&/INCLUDE &&VARBL1 &&VARBL5
/END
/PEND
/RUN
```

Figure 71. Sample procedure for assembling on ICCF

Invoking the assembler dynamically

To invoke High Level Assembler from a running program, use the CDLOAD and CALL macro instructions.

You can supply assembler options in the CALL macro instruction as shown in Figure 72

```
DYNAMICV CSECT
DYNAMICV RMODE  24
DYNAMICV AMODE  ANY
BEGIN    SAVE   (14,12)
         USING  BEGIN,15
         ST     13,SAVEAREA+4
         LA     13,SAVEAREA
         CDLOAD ASMA90            1
         LR     15,0
         CALL   (15),(OPTIONS)    2    3
         CDDELETE ASMA90
         L      13,SAVEAREA+4
         RETURN (14,12)
SAVEAREA DS     18F
OPTIONS  DC     Y(OPTIONSL)
OPTS     DC     C'XREF(SHORT)'
OPTIONSL EQU    *-OPTS
         END
```

Figure 72. Sample program to call the assembler dynamically

Notes on Figure 72:

1 ASMA90 is the symbolic name of the assembler. The entry point address is returned by CDLOAD in register 0.

2 (15) specifies that the entry point address is in register 15.

3 (OPTIONS) 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 a count of 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 spaces should appear in the list, except within the string specified for the EXIT or SYSPARM options providing the string is enclosed within apostrophes.

Batch assembling

You can assemble a sequence of separate assembler programs with a single invocation of the assembler by specifying the BATCH option. The object programs produced from this assembly can be link-edited into either a single phase or separate phases.

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 is assembled 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 you need to produce separate phases, you must write a phase linkage editor control statement for each phase. The phase statement must be written at the start of the module. The following example shows how to create two phases, SECT1 and SECT2. When multiple phases are produced, they are link-edited as an overlay.

```
         PUNCH ' PHASE SECT1,*'
         END
SECT1    CSECT          Start of first load module
⋮
         Source instructions
⋮
         END            End of first load module
         PUNCH ' PHASE SECT2,*'
         END
SECT2    CSECT          Start of second load module
⋮
         Source instructions
⋮
         END            End of second load module
```

Controlling your assembly

The assembler options are specified on the PARM parameter of the JCL EXEC statement or the ICCF /LOAD job entry statement. The options must be enclosed within apostrophes and be separated by commas.

The assembler options are described in Chapter 3, "Controlling your assembly with options," on page 35. You can also specify some assembler options using the // OPTION job control statement. These are described in Table 31.

Table 31. Assembler options in JCL

Assembler Option	JCL OPTION Equivalent	Comments
ALIGN	ALIGN	
DECK	DECK	The DECK assembler option is always specified using the JCL OPTION statement. If the DECK option is specified on the PARM operand of the JCL EXEC statement, error diagnostic message `ASMA400W` is issued, and the DECK option is ignored.
LIST	LIST	The LIST assembler option is equivalent to specifying LIST(121).
OBJECT	LINK CATAL	The OBJECT assembler option is always specified using the LINK or CATAL option of the JCL OPTION statement. If the OBJECT option is specified on the PARM operand of the JCL EXEC statement, error diagnostic message `ASMA400W` is issued, and the OBJECT option is ignored.
RLD	RLD	

Table 31. Assembler options in JCL (continued)

Assembler Option	JCL OPTION Equivalent	Comments
SYSPARM	SYSPARM	The value specified in the SYSPARM option of the JCL OPTION statement is limited to eight characters. To provide longer values, use the SYSPARM assembler option. The SYSPARM value specified on the PARM operand of the JCL EXEC statement overrides any value specified on the JCL OPTION statement. A null value (// OPTION SYSPARM='') is ignored by the assembler.
TERMINAL	TERM	
XREF	SXREF XREF	The XREF option of the JCL OPTION statement can be used to specify the XREF(FULL) assembler option. The SXREF option of the JCL OPTION statement can be used to specify the XREF(SHORT) assembler option.
WORFILE	WORFILE	A WORFILE (SYSUT1) is used as a spill file should the assembler need to store internal data and has exhausted the memory allocated

Input and output files

Depending on the options in effect, High Level Assembler requires the following files, as shown in Figure 73:

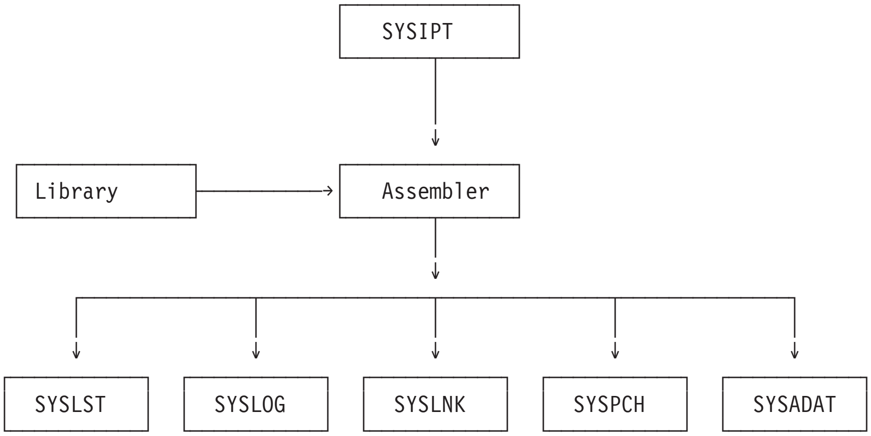


Figure 73. High Level Assembler Files

High Level Assembler requires the following files:

SYSIPT
An input file containing the source statements to be processed.

In addition, the following six files might be required:

Library

Library sublibraries containing macro definitions (for macro definitions not defined in the source program), source code to be called for through COPY assembler instructions, or both. One of the sublibraries might also contain a member (ASMAOPT.USER) which contains an invocation assembler option list.

SYSLST

A file containing the assembly listing (if the LIST option is in effect).

SYSLOG

A file containing flagged statements and diagnostic messages. (only if the TERM option is in effect). SYSLOG is normally assigned to the operator console.

SYSPCH

A file containing object module output, normally for punching (only if the DECK option is in effect).

SYSLNK

A file containing object module output for input to the linkage editor (only if the OBJECT option is in effect).

SYSADAT

A file containing associated data output (only if the ADATA option is in effect).

The files listed above are described in the text following Table 32. The characteristics of these files, those set by the assembler and those you can override, are shown in Table 32.

Table 32. Assembler file characteristics

File	Access Method	Logical Record Length (LRECL)	Block Size (BLKSIZE)	Record Format (RECFM)
SYSIPT	SAM	80	80	FIXED
Library	LIBR API	80	80	FIXED
Library member ASMAOPT.USER	LIBR API	80	80	FIXED
SYSLST	SAM	1	Same as record size	FIXED
SYSLOG	SAM	2	Same as record size	FIXED
SYSPCH	SAM	80	80	FIXED
SYSLNK	SAM	80	80	FIXED
SYSADAT	SAM	32756	32760	VARBLK
SYSUT1	SAM		32760	FIXED
ASMAOPT	SAM	80	32760	FIXED

Notes to Table 32:

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 is set to 121 if SYSLST is assigned to disk or 133 otherwise.

The minimum record length allowed for SYSPRINT is 121, and the maximum allowed is 133.

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 is set to 68. The maximum record length allowed for SYSTERM is 125.

3

Specifying the source file: SYSIPT

Define the file that contains your source code with the SYSIPT ASSGN statement. If you include your source code in your job stream, it must immediately follow the EXEC statement that invokes the assembler, and be terminated with a /* statement.

You can, however, use JCL statements to define a file that contains your source code. For example, to define a direct-access device file, use the DLBL, EXTENT, and ASSGN statements:

```
// DLBL IJSYSIN,'file-ID',0,SD
// EXTENT SYSIPT,volser,1,0,start,tracks
// ASSGN SYSIPT,DISK,VOL=volser,SHR
```

Specifying macro and copy code libraries: LIBDEF job control statement

Include the LIBDEF job control statement if your program contains any macro calls to library macros, or any COPY instructions. LIBDEF statements define the sublibraries that contain macro definition and COPY members.

```
// LIBDEF SOURCE,SEARCH=(lib.sublib)
```

The member name in the sublibrary must match the name you specify in the macro call or COPY instruction. The member type must be `A`, unless the SUBLIB job control option has changed it to `D`. See OPTION statement description in *z/VSE: System Control Statements* for further details.

High Level Assembler does not read edited macros (E-books). To read edited macros from the library, provide a LIBRARY user exit using the EXIT option. See "Processing E-decks" on page 213.

You do not need the LIBDEF SOURCE statement if your assembler source code does not contain any library macro calls or COPY instructions. You also do not need the LIBDEF SOURCE statement if the library members are in a sub-library in the permanent LIBDEF search chain.

Concatenate multiple sublibraries in the search chain if you have multiple macro or copy sublibraries.

Specifying the listing file: SYSLST

The assembler uses this file to produce a listing. You can then direct the output to a printer, a direct-access device, or a magnetic-tape device. The listing includes the results of the default or specified options of the PARM parameter (for example, diagnostic messages, the object code listing). For example:

```
// ASSGN SYSLST,PRT1
```

Directing assembler messages to your console log: SYSLOG

If you specify the TERM assembler option, the assembler writes flagged statements and diagnostic messages to the console log.

Specifying object code files: SYSLNK and SYSPCH

When using the OBJECT assembler option, or DECK assembler option, you can store the object code on disk or tape. The assembler uses the SYSLNK or SYSPCH files you define in your JCL to store the object code.

In this example, the created object module is ready to be passed to the linkage editor:

```
// DLBL IJSYSLN,'file-ID',0,SD
// EXTENT SYSLNK,volser,1,0,start,tracks
// ASSGN SYSLNK,DISK,VOL=volser,SHR
```

You do not need to define SYSLNK in your JCL if the NOOBJECT option is in effect.

The following example defines SYSPCH as a direct-access device file:

```
// DLBL IJSYSPH,'file-ID',0,SD
// EXTENT SYSPCH,volser,1,0,start,tracks
ASSGN SYSPCH,DISK,VOL=volser,SHR
```

You do not need to define the SYSPCH file if the NODECK option is in effect.

Specifying the associated data file: SYSADAT

Use the SYSADAT DLBL statement to define your associated data output file statement:

```
// DLBL SYSADAT,'HLASM.WORK.SYSADAT',0,VSAM,RECORDS=(100,100),        X
               RECSIZE=32760,DISP=(NEW,KEEP),CAT=VSESPUC
```

The associated data file contains information about the assembly. It provides information for use by symbolic debugging and cross-reference tools. The SYSADAT file must be directed to a direct-access storage device and can be a SAM file or SAM-ESDS file.

The recommended minimum LRECL for the SYSADATA data file is 8188.

Specifying the utility data file: SYSUT1

Assembler work file (only if the WORKFILE option is in effect).

Specifying the option file: ASMAOPT

The options file can be used to pass HLASM assembler option to the assembler as opped to passing options via the PARM statement. See "The sources of assembler options" on page 35 for more details

Return codes

High Level Assembler issues return codes that you can check with the IF and ON job control statements. The IF and ON job control statements let you skip or run a job step, depending on the results (indicated by the return code) of a previous job step. See *z/VSE: System Control Statements* for details about the IF and ON job control statements.

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 "FLAG" on page 48. See Appendix F, "High Level Assembler messages," on page 297 for a listing of the assembler errors and their severity codes.

Chapter 14. Link-editing and running your program on z/VSE

If you produce an object module when you assemble your program, it needs further processing before it can run. This further processing, the resolution of external references inserted by the assembler, is performed by the linkage editor. The linkage editor converts an object module into an executable program, which is called a *phase*.

The linkage editor

The linkage editor converts an object module into a phase and catalogs it in a library sublibrary. The phase then becomes a permanent member of that sublibrary, with a member type of PHASE, and can be retrieved at any time and run in either the job that created it or any other job.

Alternatively, you can request the linkage editor to store the phase temporarily, in the virtual I/O area. The phase is then ready to run. Using this method, the linkage editor does not save a permanent copy of the phase. Consequently, after the phase has been run, it cannot be used again without creating another phase. This method is useful during testing.

Creating a phase

The linkage editor processes your assembled program (object module) and prepares it for running. The processed object module becomes a phase.

Optionally, the linkage editor can process more than one object module, and transform those object modules into a single phase.

Figure 74 shows the general job control procedure for creating a phase (link-editing).

```
// JOB jobname
// DLBL IJSYSLN,'file-ID',0,SD
// EXTENT SYSLNK,volser,1,0,start,tracks
// ASSGN SYSLNK,DISK,VOL=volser,SHR
// LIBDEF OBJ,SEARCH=(lib.sublib)
// LIBDEF PHASE,CATALOG=(lib.sublib)
// OPTION CATAL
   ACTION MAP
   PHASE phasenam,*

⋮
// EXEC LNKEDT
/&
```

Figure 74. Sample job control for creating a phase

Input to the linkage editor

Your input to the linkage editor can be:

- One or more object modules (created through the OBJECT or DECK assembler option).
- Linkage editor control statements (including control statements generated using the assembler PUNCH statement).

© Copyright IBM Corp. 1992, 2013

Inputting object modules

The main input to the linkage editor is the SYSLNK file that contains one or more separately assembled object modules, possibly with a PHASE linkage editor control statement.

Additional input to the linkage editor consists of object modules that are not part of the SYSLNK file, but are to be included in the phase.

The additional input can come from sublibraries containing other application object modules.

You can specify which sublibrary contains the additional object modules with the LIBDEF job control statement. If you have multiple sublibraries containing object modules to be included in the phase, concatenate them, as shown in the following example:

```
// LIBDEF OBJ,SEARCH=(PRD2.PROD,SALES.LIB)
```

In this case, the sublibraries PRD2.PROD and SALES.LIB are available for additional input to the linkage editor.

Files for linkage editor processing

You need the following files for linkage editor processing. Table 33 summarizes the function, and permissible device types, for each file.

Table 33. Files used for link-editing

File	Type	Function	Permissible Device Types
SYSIPT[1]	Input	Additional object module input	Card readerMagnetic tapeDirect access
SYSLNK	Input	Object module input, normally the output of the assembler	Direct access
SYSLST[2]	Output	Diagnostic messagesInformative messagesLinkage editor map	PrinterMagnetic tapeDirect access
SYSLOG	Output	Operator messages	Display console
SYSRDR	Input	Control statement input	Card readerMagnetic tapeDirect access
IJSYS01 (SYS001)	Work file	Linkage editor work file	Direct access
User-specified Sublibrary	Library	Catalog sublibrary for the phase[3] External reference and INCLUDE statement resolution[4]	Direct access

Notes:

[1] Object modules read from SYSIPT are written to SYSLNK

[2] If not provided, messages are written to SYSLOG

[3] Required if the phase is to be cataloged

[4] Required for additional object module input

Inputting additional object modules

You can use the INCLUDE linkage editor control statement to specify additional object modules you want included in the phase.

Code the INCLUDE statements before the EXEC statement that invokes the linkage editor:

```
// EXEC ASMA90,SIZE=ASMA90
⋮
/*
```

```
 INCLUDE ASSMPGM
 INCLUDE ASSMPGM1
// EXEC LNKEDT
/&
```

Object modules specified by the INCLUDE statement are written to SYSLNK as job control encounters the statements.

Linkage editor control statements

In addition to object modules, input to the linkage editor includes linkage editor control statements. These statements are described in Table 34.

Table 34. Linkage editor control statements

Statement	Action	Comments
ACTION	Use the ACTION statement to specify linkage editor options. The options are: • MAP—requests the linkage editor to write a linkage editor map to SYSLST. • NOMAP—suppresses the MAP option. • NOAUTO—suppresses the automatic library look up (AUTOLINK) function; the linkage editor does not attempt to resolve external references using the automatic library look-up function. • CANCEL—requests the linkage editor to cancel the job if a linkage editor error occurs. • SMAP—request the linkage editor to produce a sorted listing of CSECT names on SYSLST.	This statement must be the first linkage editor statement in your input stream. ACTION MAP is the default, if SYSLST is assigned.
ENTRY	Use the ENTRY statement to specify the entry point of a phase that has multiple possible entry points.	The default entry point is the load address of the phase.
INCLUDE	Use the INCLUDE statement to include additional object modules in the phase that are not otherwise included.	You can use the INCLUDE statement to include an object module that was cataloged with a different name to the name used in the CALL statement in your program.
PHASE	Use the PHASE statement to provide the linkage editor with a phase name.	You must provide a PHASE statement (and the job control option CATAL) if you want to catalog the phase in a library sublibrary.

For a complete description of these linkage editor control statements, see *z/VSE: System Control Statements*.

Output from the linkage editor

You can obtain a linkage editor storage map, and a listing of linkage editor diagnostics, to help you determine the reasons for particular errors in your program. To do this, use the ACTION MAP linkage editor control statement. If SYSLST is assigned, ACTION MAP is the default. You can specify ACTION NOMAP if you do not want the linkage editor to produce the storage map.

Detecting Link-Edit Errors: After link-editing, you receive a listing of diagnostic messages on SYSLST. Check the linkage editor map to make sure that all the object modules you expected were included.

Unresolved “weak” external references (WXTRN) can be ignored. However, all “strong” external references (EXTRN) should be resolved for a phase to run correctly.

You can find a description of linkage editor messages in *VSE/ESA Diagnostic Tools*.

Running your assembled program

The general job control procedure to run a program on z/VSE is:

```
// DLBL (JCL for user-specified files)
// EXEC progname[,PARM='parameters']
```

⋮

Chapter 15. z/VSE system services and programming considerations

This chapter describes some of the system services and program development facilities that assist you in developing your assembler program on z/VSE. It provides the following information:

- Adding definitions to a macro library.
- Saving and restoring general register contents.
- Ending program execution.
- Accessing execution parameters.
- Processing E-Decks.

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. Use the LIBR utility program for this purpose. Details of this program and its control statements are contained in the applicable *System control statements* publication. The following example adds a new macro definition, NEWMAC, to the system library, PRD2.PROD.

```
// JOB CATMAC
// EXEC LIBR
ACCESS SUBLIB=PRD2.PROD
CATALOG NEWMAC.A REPLACE=YES
          MACRO
          NEWMAC &OP1,&OP2
          LCLA   &PAR1,&PAR2

⋮
          MEND
/+
/*
```

The ACCESS statement specifies the sublibrary into which the macro is cataloged. The CATALOG statement specifies the member name and member type. REPLACE=YES indicates that the member is replaced if it already exists.

Processing E-decks

An E-Deck refers to a macro source book of type E (or type F if `SUBLIB=DF` specified on OPTION statement). You can use these types of macros in your program; however, they must first be converted to source statement format. E-Decks are stored in edited format, and High Level Assembler requires that library macros be stored in source statement format.

You must use a library input exit to analyze a macro definition and, in the case of an E-Deck, call the ESERV program to change, line by line, the E-Deck definition back into source statement format.

See the section titled *Using the High Level Assembler Library Exit for Processing E-Decks* in *z/VSE: Guide to System Functions*. This section describes how to set up the exit and how to use it.

Operating system programming conventions

Assembler programs executing on z/VSE must follow a set of programming conventions to save and restore registers, and access execution parameters. These are described in the following sections.

© Copyright IBM Corp. 1992, 2013

Saving and restoring general register contents

A program should save the values contained in the general registers when it starts to run and, on completion, restore these same values to the general registers. 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 GETVIS, FREEVIS, and CDLOAD, 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 *z/VSE: System Macros Reference*.

```
Name     Operation     Operand

BEGIN    SAVE          (14,12)
         USING         BEGIN,15
⋮
         ST            13,SAVEBLK+4
         LA            13,SAVEBLK
⋮
         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 return is to the operating system, the return code is compared against the condition stated in the IF and ON job control statements.

If return is 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 applicable application programming *Macro Reference* document.

Accessing execution parameters

You access information in the PARM field of an EXEC statement by referring to the contents of general register 1. If you do not specify the PARM field of the JCL EXEC statement, register 1 and register 15 contain the same value on initial entry.

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 75 shows how the PARM field information is structured.

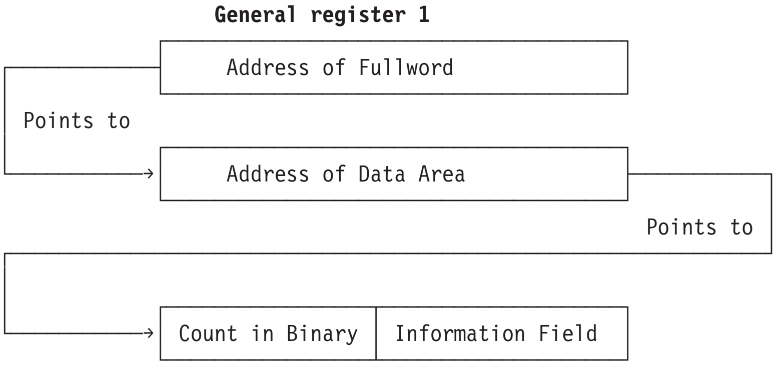


Figure 75. Access to PARM field

Appendix A. 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 machine instructions

High Level Assembler supports assembly of programs using z/Architecture instructions, Extended Architecture instructions, Enterprise System Architecture instructions, and Vector instructions, under all operating systems supported by High Level Assembler.

A generated object program using a specified set of instructions can only run on a processor under an operating system that provides the necessary architecture support for the 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 z/OS and z/VSE but generate different object code.

Wherever the assembled program uses system macros, the system macros for the target system must be used when the program is assembled.

For example, when the assembled program is to be run under z/VSE, the z/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 assembled program is to run.

Migrating object programs

The object module produced by High Level Assembler is portable across all the supported operating systems (but GOFF object files are not portable to z/VSE). Therefore, an assembler program can be assembled under any of the supported operating systems and run under any of the supported operating systems. For example, an assembler program can be assembled under CMS and run under z/VSE if the appropriate macro libraries are used for assembling the program.

The object module is portable across the supported operating systems with the following restrictions:

- Wherever 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 z/VSE linkage editor before 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 z/OS.

  z/VM and Nz/OS

  The TEST option cannot be specified with the GOFF assembler option, which produces the generalized object format module.
- A generalized object format module cannot be ported to a z/VSE or CMS environment.

© Copyright IBM Corp. 1992, 2013

- Some enhancements such as external relative-immediate references and quadword-aligned control sections might not be supported on some z/VM and z/VSE systems.

Appendix B. Object deck output

High Level Assembler produces the object module when you specify either the OBJECT or DECK assembler option.

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 TSO TEST.

Note: If you have specified the GOFF assembler option, High Level Assembler produces the object module in Generalized Object File format (GOFF). For more information about GOFF, refer to *z/OS MVS Program Management: Advanced Facilities*.

The assembler can also produce records using 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	Space
11–12	Variable field count—number of bytes of information in variable field (columns 17–64)
13–14	Space
15–16	ESDID of first SD, XD, CM, PC, ER, or WX in variable field; blank for LD items, which have the ESDID of the preceding SD. This field is blank if all items are LD.
17–64	Variable field. One-to-three 16-byte items of the following format:

- 8-byte external symbol name
- 1-byte ESD type code:

Hex Value	**ESD Type Code**
00	SD
01	LD
02	ER
04	PC
05	CM
06	XD(PR)
0A	WX
0D	Quad-aligned SD

© Copyright IBM Corp. 1992, 2013

0E	Quad-aligned PC
0F	Quad-aligned CM

- 3-byte address
- 1-byte flag:
  - Alignment if XD
  - Space if LD, ER, or WX
  - AMODE/RMODE flags if SD, PC, or CM. Table 35 describes the AMODE and RMODE flag values.

Table 35. AMODE/RMODE flags

Bits	Value	Description
2	0	Use the RMODE bit (bit 5)
	1	RMODE 64
3	0	Use the AMODE bits (bits 6-7)
	1	AMODE 64
4	1	RSECT
5	0	RMODE 24
	1	RMODE 31, RMODE ANY
6–7	00	AMODE 24
	01	AMODE 24
	10	AMODE 31
	11	AMODE ANY

- 3-byte length, LDID, or space

Variable field item 1

17–24	External symbol name
25	ESD type code
26–28	Address
29	Flag
30–32	Length, LDID, or space

Variable field item 2

33–40	External symbol name
41	ESD type code
42–44	Address
45	Flag
46–48	Length, LDID, or space

Variable field item 3

49–56	External symbol name
57	ESD type code
58–60	Address
61	Flag
62–64	Length, LDID, or space

65–72	Space
73–80	Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a non-spaces name field. This name can be 1–8 characters. If the name is fewer than eight characters or if there is no name, the remaining columns contain a record sequence number.

TXT record format

Columns	Contents
1	X'02'
2–4	TXT
5	Space
6–8	Relative address of first instruction on record
9–10	Space
11–12	Byte count—number of bytes in information field (columns 17–72)
13–14	Space
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 non-spaces name field. The name can be 1–8 characters. If the name is fewer than eight 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	Space
11–12	Data field count—number of bytes of information in data field (columns 17–72)
13–16	Space
17–72	Data field: **17–18** Relocation ESDID **19–20** Position ESDID **21** Flag byte. Describes the type of relocation required: **Bit 0** Reserved **Bit 1** If 1, add 4 to address constant length **Bit 2—3** Address constant type: **0** A **1** V **2** Q **3** CXD **Bits 4–5** Address constant length minus 1 **Bit 6** Direction of relocation (0 for +, 1 for -) **Bit 7** If 1, the same P-ID and R-ID for the next RLD item **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 non-spaces name field. The name can be 1–8 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.

Certain combinations in the Flag byte are used to indicate Relative-Immediate relocation items:

```
B'x111 00xx'   2-byte Relative-Immediate reference
B'x111 10xx'   4-byte Relative-Immediate reference
```

The first RLD item on each RLD record must specify the Relocation and Position ESDIDs; thus, the last RLD item on a record must not set the rightmost bit of the flag byte.

For example, if the RLD entries 1, 2, and 3 of the program listing contain the following information:

```
        Position   Relocation
Entry   ESDID      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 are:

```
       │         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 │ Spaces
       │             │          ││           │             │          │
       │             Flag       │Flag        │             Flag       │
       │             (Set)      │(not        │             (not       │
       │                        │set)        │             set)       │
```

END record format

Columns	Contents
1	X'02'
2–4	END
5	Space
6–8	Entry address from operand of END record in source deck (blank if no operand)
9–14	Space
15–16	For a Type 1 END record: ESDID of entry point For a Type 2 END record: Blank
17–24	For a Type 1 END record: Blank For a Type 2 END record: Symbolic entry point name if specified, otherwise blank
25–28	Blank
29–32	Control section length for a CSECT whose length was not specified on its SD ESD item. Byte 29 is zero if this length field is present. (Blank field if not present.)
33	Number of IDR items that follow (EBCDIC 1 or EBCDIC 2), or blank if none

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 Space

73–80 Deck ID, sequence number, or both. The deck ID is the name from the first TITLE statement that has a non-spaces name field. The name can be 1–8 characters. If the name is fewer than eight characters or if there is no name, the remaining columns contain a record sequence number.

SYM record format

If you request it, the assembler writes out symbolic information concerning the assembled program ahead of all other object module records. The format of the output record images is as follows:

Columns
Contents

1 X'02'

2–4 SYM

5–10 Space

11–12 Variable field—number of bytes of text in variable field (columns 17–72)

13–16 Space

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 non-space name field. The name can be 1–8 characters. If the name is fewer than eight 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 text. The items comprising the text are packed together; consequently, only the last record can 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 76 on page 225. The contents of the fields within an individual entry are as follows:

1. Organization (1 byte). The possible values are shown in Table 36.

Table 36. 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)

Table 36. Organization value byte (continued)

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 non-data 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 and hexadecimal, 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	Space	No. of bytes of text	Space	Text - packed entries	Deck Id and Seq. Number
1	3	6	2	4	56	8

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 76. SYM record format

Appendix C. Associated data file output

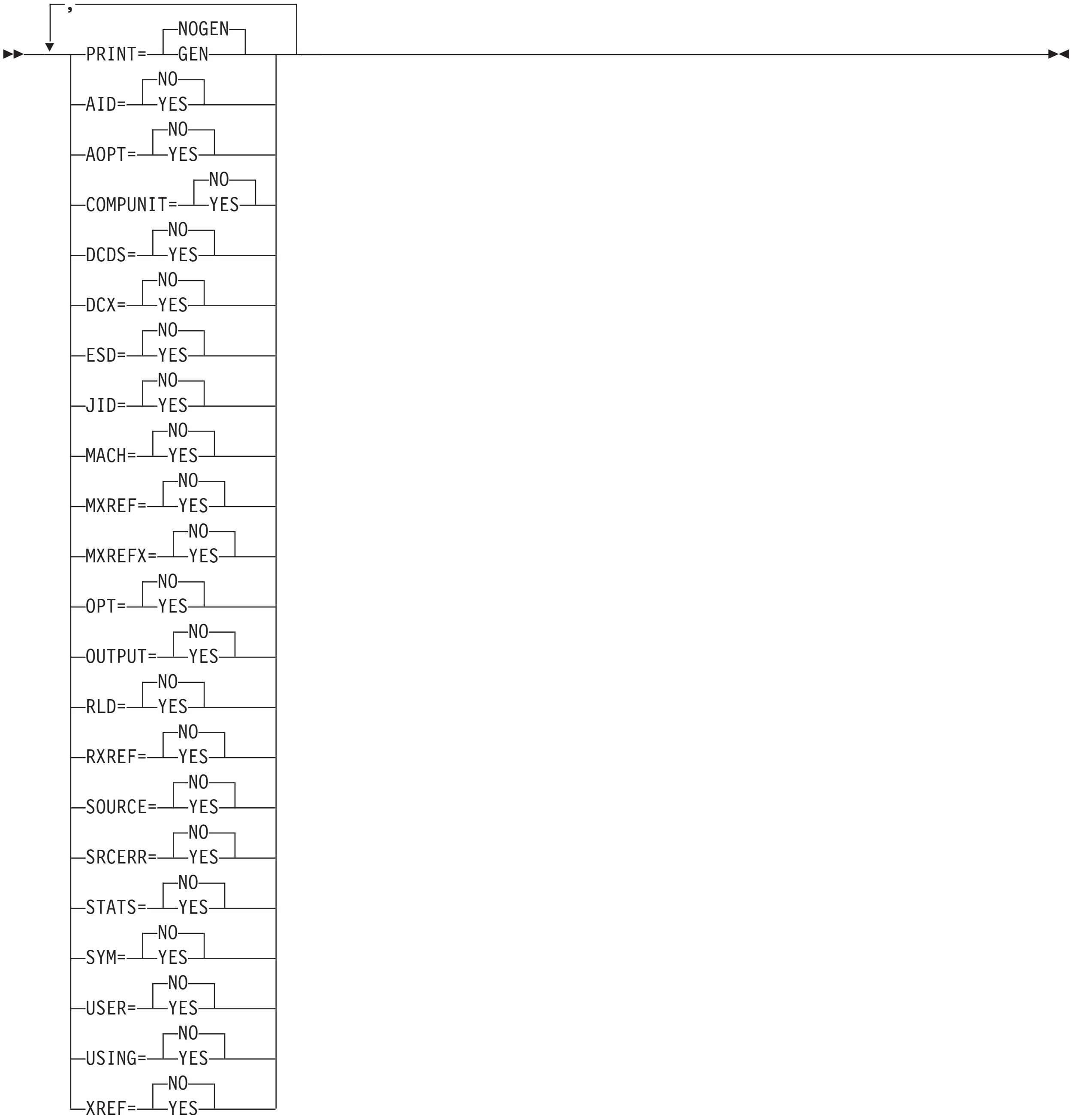
When you specify the ADATA assembler option, a file containing associated data is produced by the assembler. When you specify the ADATA suboption of the GOFF assembler option, ADATA records are written to the object data set as text records. You can specify both ADATA and GOFF(ADATA) to produce ADATA 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 be used by debugging tools or cross reference tools.

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 generalized object format data set. The syntax and parameter keywords for this macro are shown on page 228.

© Copyright IBM Corp. 1992, 2013

ASMADATA



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')
AOPT	Options File Information (Type X'000B')
COMPUNIT	ADATA Compilation Unit Start/End DSECT (Type X'0002')
DCDS	DC/DS DSECT (Type X'0034')
DCX	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')
RXREF	Register Cross Reference DSECT (Type X'0045')
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')

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 address 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.

Offsets within each record are calculated from the beginning of the header section.

The following examples show the format of a DC/DS Record for various DC statements.

1. EXAMPLE1 DC 3F'5,6',HP(X'05')'7'

```
ESDID                   : F'1'
Type Flag               : B'10000000'
Reserved                : X'0000000000'
Statement Number        : F'2'
Number of Operands      : F'2'
Offset of first Operand : F'38'

   Offset of next Operand  : F'108'
   Location Counter        : X'00000000'
   Duplication Factor      : F'3'
   Bit Offset              : B'00000000'
   Type Attribute          : C'F'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'2'
   Offset of first value   : F'68'

      Offset of next value      : F'88'
      Offset of generated value : F'84'
      Byte length               : F'4'
      Bit length                : F'0'
      Generated Value           : X'00000005'

      Offset of next value      : F'0'
      Offset of generated value : F'104'
      Byte length               : F'4'
      Bit length                : F'0'
      Generated Value           : X'00000006'
   Offset of next Operand  : F'0'
   Location Counter        : X'00000018'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000000'
   Type Attribute          : C'H'
   Type Extension          : C' '
   Program Type            : X'00000005'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'138'

      Offset of next value      : F'0'
      Offset of generated value : F'154'
      Byte length               : F'2'
      Bit length                : F'0'
      Generated Value           : X'0007'
```

2. EXAMPLE2 DC P'5,927'

```
ESDID                   : F'1'
Type Flag               : B'10000000'
Reserved                : X'0000000000'
Statement Number        : F'3'
Number of Operands      : F'1'
Offset of first Operand : F'38'

   Offset of next Operand  : F'0'
   Location Counter        : X'0000001A'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000000'
   Type Attribute          : C'P'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'2'
   Offset of first value   : F'68'

      Offset of next value      : F'85'
      Offset of generated value : F'84'
      Byte length               : F'1'
      Bit length                : F'0'
      Generated Value           : X'5C'

      Offset of next value      : F'0'
      Offset of generated value : F'101'
      Byte length               : F'2'
      Bit length                : F'0'
      Generated Value           : X'927C'
```

3. EXAMPLE3 DC B'101',2B'10111'

```
ESDID                   : F'1'
Type Flag               : B'10000000'
Reserved                : X'0000000000'
Statement Number        : F'4'
Number of Operands      : F'2'
Offset of first Operand : F'38'

   Offset of next Operand  : F'85'
   Location Counter        : X'0000001D'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000000'
   Type Attribute          : C'B'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'68'

      Offset of next value      : F'0'
      Offset of generated value : F'84'
      Byte length               : F'1'
      Bit length                : F'0'
      Generated Value           : X'05'            B'00000101'
   Offset of next Operand  : F'0'
   Location Counter        : X'0000001E'
   Duplication Factor      : F'2'
   Bit Offset              : B'00000000'
   Type Attribute          : C'B'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'115'

      Offset of next value      : F'0'
      Offset of generated value : F'131'
      Byte length               : F'1'
      Bit length                : F'0'
      Generated Value           : X'17'            B'00010111'
```

4. EXAMPLE4 DC BL.3'101',BL.5'10111,11001'

```
ESDID                   : F'1'
Type Flag               : B'10000000'
Reserved                : X'0000000000'
Statement Number        : F'5'
Number of Operands      : F'2'
Offset of first Operand : F'38'
```

```
   Offset of next Operand  : F'85'
   Location Counter        : X'00000020'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000000'
   Type Attribute          : C'B'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'68'

      Offset of next value      : F'0'
      Offset of generated value : F'84'
      Byte length               : F'0'
      Bit length                : F'3'
      Generated Value           : X'A0'            B'10100000'
   Offset of next Operand  : F'0'
   Location Counter        : X'00000020'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000011'
   Type Attribute          : C'B'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'2'
   Offset of first value   : F'115'

      Offset of next value      : F'132'
      Offset of generated value : F'131'
      Byte length               : F'0'
      Bit length                : F'5'
      Generated Value           : X'B8'            B'10111000'

      Offset of next value      : F'0'
      Offset of generated value : F'148'
      Byte length               : F'0'
      Bit length                : F'5'
      Generated Value           : X'C8'            B'11001000'
```

5. EXAMPLE5 DC LB'4',2L'9'

   This example shows a DC statement that requires the type extension field to differentiate the attributes of the two floating point operands.

```
ESDID                   : F'1'
Type Flag               : B'10000000'
Reserved                : X'0000000000'
Statement Number        : F'6'
Number of Operands      : F'2'
Offset of first Operand : F'38'

   Offset of next Operand  : F'100'
   Location Counter        : X'00000028'
   Duplication Factor      : F'1'
   Bit Offset              : B'00000000'
   Type Attribute          : C'L'
   Type Extension          : C'B'
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'68'

      Offset of next value      : F'0'
      Offset of generated value : F'84'
      Byte length               : F'16'
      Bit length                : F'0'
      Generated Value           : X'40010000000000000000000000000000'
   Offset of next Operand  : F'0'
   Location Counter        : X'00000038'
   Duplication Factor      : F'2'
   Bit Offset              : B'00000000'
   Type Attribute          : C'L'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'130'

      Offset of next value      : F'0'
      Offset of generated value : F'146'
      Byte length               : F'16'
      Bit length                : F'0'
      Generated Value           : X'41900000000000003300000000000000'
```

6. EXAMPLE6 DC 5Y(*-2),5Y(*-1)

   This example shows a DC statement that requires a DC extension record (X'0035') 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'
Type Flag               : B'10010000'
Reserved                : X'0000000000'
Statement Number        : F'3'
Number of Operands      : F'2'
Offset of first Operand : F'38'

   Offset of next Operand  : F'86'
   Location Counter        : X'00000000'
   Duplication Factor      : F'5'
   Bit Offset              : B'00000000'
   Type Attribute          : C'Y'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'130'

      Offset of next value      : F'0'
      Offset of generated value : F'84'
      Byte length               : F'2'
      Bit length                : F'0'
      Generated Value           : X'FFFE'
   Offset of next Operand  : F'86'
   Location Counter        : X'0000000A'
   Duplication Factor      : F'5'
   Bit Offset              : B'00000000'
   Type Attribute          : C'Y'
   Type Extension          : C' '
   Program Type            : X'00000000'
   Reserved                : X'00000000'
   Number of values        : F'1'
   Offset of first value   : F'116'

      Offset of next value      : F'0'
      Offset of generated value : F'132'
      Byte length               : F'2'
      Bit length                : F'0'
      Generated Value           : X'0009'
```

The object text for the statement is in the following DC Extension Record:

```
ESDID                  : F'1'
Statement Number       : F'3'
Location Counter       : F'0'
Reserved               : X'0000000000000000'
Offset of Object       : F'44'
Length of Object       : F'20'
Object Text            : X'FFFE0000000200040006000900OB000D000F0011'
```

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:

- A 12-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 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 File X'000B'
Provides information about the ASMAOPT file (z/OS and CMS) or library member (z/VSE) used for the assembly, if applicable.

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 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 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.

Register Cross Reference X'0045'
Describes the references to a single register.

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 77 shows part of the listing of an assembler program. If this assembler program is assembled with the ADATA option, the records produced in the associated data file are in the sequence shown below Figure 77.

```
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                   HLASM R6.0  2008/07/11 17.48
                      00000 0001E     1 CSECTNAM CSECT                                                           FIG00010
000000 90EC D00C            0000C     2          STM   14,12,12(13)                                              FIG00020
                 R:F  00000           3          USING CSECTNAM,15                                               FIG00030
000004 0000 0000            00000     4          A     2,FIELD3                                                  FIG00040
** ASMA044E Undefined symbol - FIELD3
000008 98EC D00C            0000C     5          LM    14,12,12(13)                                              FIG00050
00000C 07FE                           6          BR    14                                                        FIG00060
                                      7          DROP  15                                                        FIG00070
                                      8          COPY  ADATA                                                     FIG00080
00000E                                9=FIELD1   DS    CL8                                                       ADA00010
000016                               10=FIELD2   DS    CL8                                                       ADA00020
                                     11          END                                                             FIG00090
```

Figure 77. Sample assembler program for associated data output

Type	**Description**
X'0002'	ADATA Compilation Unit START record
X'0001'	ADATA Identification record
X'0000'	Job Identification 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(13)`
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'0042'
Symbol record for FIELD2

X'0042'
Symbol record for FIELD3

X'0044'
Symbol and Literal Cross Reference record for FIELD3

X'0044'
Symbol and Literal Cross Reference record for FIELD1

X'0044'
Symbol and Literal Cross Reference record for FIELD2

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'0045'
Register Cross Reference records...

⋮

X'0045'
...for each register referenced (0–15)[1]

X'0090'
Assembly Statistics record

X'0002'
ADATA Compilation Unit END record The count value in this record is 54.

Note:

1. There is one X'0045' record for each of the sixteen registers due to the implicit references by the STM and LM instructions.

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 78 on page 238 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 is assembled with the ADATA option, the records produced in the associated data file are in the sequence shown below.

```
 Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                 HLASM R6.0  2008/07/11 17.48
                                     1          print nogen                                                    00001000
                                     2          macro                                                          00002000
                                     3 &NAME    testhla &job                                                   00003000
                                     4          punch '//&job JOB'                                             00004000
                                     5          punch '//STEP1   EXEC  PGM=ABC'                                00005000
                                     6          punch '//DDNAME1 DD    DSN=DSN.&job.,DISP=SHR'                 00006000
                                     7          mend                                                           00007000
                                     8          TESTHLA TESTJOB                                                00008000
                                    12          END                                                            00009000
```

Figure 78. Sample assembler program for macro-only assembly

Type	Description
X'0002'	ADATA Compilation Unit START record
X'0001'	ADATA Identification 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.

Common header section

Each ADATA record contains a 12-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 5 produces architecture level 3 header records. This level is described in the following sections.

Table 37. ADATA record—common header section

Field	Size	Description
Language code	FL1	**16** Assembler
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'000B'** Options 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 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'0045'** Register 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	3
Flag	XL1	**`.... ...0`** Record is not continued **`.... ...1`** Record is continued on the next record **`.... ..0.`** Length fields are big-endian (S/390®, RS/6000®) **`.... ..1.`** Length fields are little-endian (Intel) All other values are reserved.

Table 37. ADATA record—common header section (continued)

Field	Size	Description
Edition Number	FL1	The edition number of this record type. The following list of edition number values can be used to determine the format of each ADATA record. The listed edition number value (or higher) indicates that the record is in the new restructured High Level Assembler Release 5 format. **1** Job Identification record **0** ADATA Identification record **0** Compilation Unit Start/End record **1** Output File Information record **1** Options File Information record **3** Options record **1** External Symbol Dictionary record **1** Source Analysis record **1** Source Error record **1** DC/DS record **1** DC Extension record **1** Machine Instruction record **1** Relocation Dictionary record **1** Symbol record **1** Symbol and Literal Cross Reference record **1** Register Cross Reference record **1** Macro and Copy Code Source Summary record **1** Macro and Copy Code Cross Reference record **1** User Data record **1** USING Map record **2** Assembly Statistics record
Reserved	XL4	
Associated Data Field length	HL2	The length, in bytes, of the data following the header

Note:

1. The mapping of the 12-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 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.
4. If Flag indicates that the record is continued, the continue record follows the current record. The continue record includes the Common Header Section and has the same Record Type. If the continue record is continued then Flag is set with the continuation indicator.

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 spaces. For example, `C'1.6.0   '`.

Field	Size	Description
Product level	HL2	A monotonically increasing numeric value which identifies the current version and release of the assembler
PTF level	CL8	The PTF level number of the assembler that produced the associated data file
System ID	CL24	The system identification of the operating system on which the assembly was run. The value of the field is set to the value of the system variable &SYSTEM_ID.
Jobname	CL8	The job name of the assembly job
Stepname	CL8	The z/OS step name of the assembly step
Procstep	CL8	The z/OS procedure step name of the assembly procedure step
Number of input files (`SYSIN`)	FL4	The number of input files in this record. The groups of eleven input-file fields below occur *n* times depending on the value in this field.
Offset of first input-file	FL4	The offset from the beginning of this record to the first group of input-file fields. A value of binary zeros indicates that there are no input files.
		Start of input-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next input-file	FL4	The offset from the beginning of this record to the next group of input-file fields. A value of binary zeros indicates that there are no more input files.
...Input file number	FL4	The assigned sequence number of the input file
...Input file name offset	FL4	The offset from the beginning of this record to the input file name
...Input file name length	FL4	The length of the input file name
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number
...Volume serial number length	FL4	The length of the volume serial number
...Member name offset	FL4	The offset from the beginning of this record to the member name. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name. If no member name is applicable, this field contains binary zeros.
...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
		End of input-file information group.

Field	Size	Description

Note:

1. 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.
2. Where the number of input files exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of input files (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent input files. The count of the number of input files is a count for the current record.

   Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

ADATA Identification Record—X'0001'

Field	Size	Description
Time (binary)	XL8	Universal Time (UT) with the low-order bit representing 1 microsecond. This time can 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	XL2	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.
	XL2	Reserved
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 might 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.

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 GOFF (z/OS and CMS) 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	FL4	The number of primary object-files in this record. The groups of eleven primary object-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Offset of first primary object-file	FL4	The offset from the beginning of this record to the first group of primary object-file fields. A value of binary zeros indicates that there are no primary object-files.
Number of secondary object-file (`PUNCH`) output files	FL4	The number of secondary (punch) object-files in this record. The groups of eleven secondary object-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Offset of first secondary object-file	FL4	The offset from the beginning of this record to the first group of secondary object-file fields. A value of binary zeros indicates that there are no secondary object-files.
Number of listing (`PRINT`) output files	FL4	The number of listing (print) files in this record. The groups of eleven listing-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Offset of first listing-file	FL4	The offset from the beginning of this record to the first group of listing-file fields. A value of binary zeros indicates that there are no listing files.
Number of terminal (`TERM`) output files	FL4	The number of terminal files in this record. The groups of eleven terminal-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Offset of first terminal-file	FL4	The offset from the beginning of this record to the first group of terminal-file fields. A value of binary zeros indicates that there are no terminal files.
Number of associated-data (`ADATA`) output files	FL4	The number of associated-data files in this record. The groups of eleven associated-data output-file fields below occur *n* times depending on the value in this field. (This number is normally 1.)
Offset of first associated-data file	FL4	The offset from the beginning of this record to the first group of associated-data-file fields. A value of binary zeros indicates that there are no associated-data files.

Field	Size	Description
		Start of primary object-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next primary object-file	FL4	The offset from the beginning of this record to the next group of primary object-file fields. A value of binary zeros indicates that there are no more primary object-files.
...Primary object-file file number	FL4	The assigned sequence number of the primary object-file
...Primary object-file name offset	FL4	The offset from the beginning of this record to the output file name for the primary object-file
...Primary object-file name length	FL4	The length of the output file name for the primary object-file
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number for the primary object-file
...Volume serial number length	FL4	The length of the volume serial number for the primary object-file
...Member name offset	FL4	The offset from the beginning of this record to the member name in the primary object-file. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name in the primary object-file. If no member name is applicable, this field contains binary zeros.
...Primary object-file name	CL(n)	The name of the primary object-file for the assembly
...Volume serial number	CL(n)	The volume serial number of the volume on which the primary object-file resides
...Member name	CL(n)	Where applicable, the name of the member in the primary object-file.
		End of primary object-file information group.
		Start of secondary object-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next secondary object-file	FL4	The offset from the beginning of this record to the next group of secondary object-file fields. A value of binary zeros indicates that there are no more secondary object-files.
...Secondary object- file file number	FL4	The assigned sequence number of the secondary object-file
...Secondary object- file name offset	FL4	The offset from the beginning of this record to the output file name for the secondary object-file
...Secondary object- file name length	FL4	The length of the output file name for the secondary object-file
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number for the secondary object-file
...Volume serial number length	FL4	The length of the volume serial number for the secondary object-file
...Member name offset	FL4	The offset from the beginning of this record to the member name in the secondary object-file. If no member name is applicable, this field contains binary zeros.

Field	Size	Description
...Member name length	FL4	The length of the member name in the secondary object-file. If no member name is applicable, this field contains binary zeros.
...Secondary object- file name	CL(n)	The name of the secondary object-file for the assembly
...Volume serial number	CL(n)	The volume serial number of the volume on which the secondary object-file resides
...Member name	CL(n)	Where applicable, the name of the member in the secondary object-file.
		End of secondary object-file information group.
		Start of listing-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next listing-file	FL4	The offset from the beginning of this record to the next group of listing-file fields. A value of binary zeros indicates that there are no more listing files.
...Listing-file file number	FL4	The assigned sequence number of the listing file
...Listing-file name offset	FL4	The offset from the beginning of this record to the output file name for the listing file
...Listing-file name length	FL4	The length of the output file name for the listing file
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number for the listing file
...Volume serial number length	FL4	The length of the volume serial number for the listing file
...Member name offset	FL4	The offset from the beginning of this record to the member name in the listing file. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name in the listing file. If no member name is applicable, this field contains binary zeros.
...Listing-file name	CL(n)	The name of the listing file for the assembly
...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 in the listing file.
		End of listing-file information group.
		Start of terminal-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next terminal-file	FL4	The offset from the beginning of this record to the next group of terminal-file fields. A value of binary zeros indicates that there are no more terminal files.
...Terminal-file file number	FL4	The assigned sequence number of the terminal file
...Terminal-file name offset	FL4	The offset from the beginning of this record to the output file name for the terminal file
...Terminal-file name length	FL4	The length of the output file name for the terminal file
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number for the terminal file
...Volume serial number length	FL4	The length of the volume serial number for the terminal file

Field	Size	Description
...Member name offset	FL4	The offset from the beginning of this record to the member name in the terminal file. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name in the terminal file. If no member name is applicable, this field contains binary zeros.
...Terminal-file name	CL(n)	The name of the terminal file for the assembly
...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 in the terminal file.
		End of terminal-file information group.
		Start of associated-data-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next associated-data file	FL4	The offset from the beginning of this record to the next group of associated-data-file fields. A value of binary zeros indicates that there are no more associated-data files.
...Associated-data-file file number	FL4	The assigned sequence number of the associated-data file
...Associated-data file name offset	FL4	The offset from the beginning of this record to the output file name for the associated-data file
...Associated-data file name length	FL4	The length of the output file name for the associated-data file
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number for the associated-data file
...Volume serial number length	FL4	The length of the volume serial number for the associated-data file
...Member name offset	FL4	The offset from the beginning of this record to the member name in the associated-data file. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name in the associated-data file. If no member name is applicable, this field contains binary zeros.
...Associated-data file name	CL(n)	The name of the associated-data file for the assembly
...Volume serial number	CL(n)	The volume serial number of the volume on which the associated-data file resides
...Member name	CL(n)	Where applicable, the name of the member in the associated-data file
		End of associated-data-file information group.

Note:

Where the number of output files exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of output files (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent output files. The count of the number of output files is a count for the current record.

Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

Options File Information—X'000B'

The Options File Information record provides data about any option files passed to the assembler; using an external file (z/OS and CMS) with the DDname ASMAOPT or library member (z/VSE) with the name and type ASMAOPT.USER.

Field	Size	Description
Number of option (`ASMAOPT`) input files	FL4	The number of option files in this record. The groups of eleven option-file fields below occur *n* times depending on the value in this field.
Offset of option file	FL4	The offset from the beginning of this record to the first group of option-file fields. A value of binary zeros indicates that there are no option files.
		Start of option-file information groups, one group per file. The ellipses (...) indicate the fields are grouped.
...Offset of next option file	FL4	The offset from the beginning of this record to the next group of option-file fields. A value of binary zeros indicates that there are no more option files.
...Option file number	FL4	The assigned sequence number of the option file
...Option file name offset	FL4	The offset from the beginning of this record to the option file name
...Option file name length	FL4	The length of the option file name
...Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number
...Volume serial number length	FL4	The length of the volume serial number
...Member name offset	FL4	The offset from the beginning of this record to the member name. If no member name is applicable, this field contains binary zeros.
...Member name length	FL4	The length of the member name. If no member name is applicable, this field contains binary zeros.
...Option file name	CL(n)	The name of the option file for the assembly
...Volume serial number	CL(n)	The volume serial number of the (first) volume on which the option file resides
...Member name	CL(n)	Where applicable, the name of the member in the option file
		End of option-file information group.

Note:

Where the number of option files exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of option files (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent option files. The count of the number of option files is a count for the current record.

Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

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 is specified, bit 7 in option byte 8 is 1, and the PROFILE value field contains the profile member name.

The layout of the first 12 option bytes matches that of the assembler's option bytes in the ASMADOPT module.

Field	Size	Description

Option Byte 1	XL1	`1... ....`
		Bit 1 = DECK, Bit 0 = NODECK
		`.1.. ....`
		Bit 1 = OBJECT, Bit 0 = NOOBJECT
		`..1. ....`
		Bit 1 = LIST, Bit 0 = NOLIST
		`...1 ....`
		Bit 1 = XREF, Bit 0 = NOXREF
		`.... 1...`
		Bit 1 = RENT, Bit 0 = NORENT
		`.... .1..`
		Bit 1 = TEST, Bit 0 = NOTEST
		`.... ..1.`
		Bit 1 = BATCH, Bit 0 = NOBATCH
		`.... ...1`
		Bit 1 = ALIGN, Bit 0 = NOALIGN

Option Byte 2	XL1	`1... ....`
		Bit 1 = ESD, Bit 0 = NOESD
		`.1.. ....`
		Bit 1 = RLD, Bit 0 = NORLD
		`..1. ....`
		Bit 1 = XREF(SHORT), Bit 0 = not XREF(SHORT)
		`...1 ....`
		Bit 1 = TRACE specified, Bit 0 = not specified
		`.... 1...`
		Bit 1 = XREF(FULL), Bit 0 = not XREF(FULL)
		`.... .1..`
		Bit 1 = SIZE(MAX..,ABOVE), Bit 0 = not SIZE(MAX..,ABOVE)
		`.... ..1.`
		Bit 1 = XREF(UNREFS), Bit 0 = not XREF(UNREFS)
		`.... ...1`
		Bit 1 = RXREF, Bit 0 = NORXREF

Option Byte 3	XL1	`1... ....`
		Bit 1 = TERM, Bit 0 = NOTERM
		`.1.. ....`
		Bit 1 = TERM(NARROW), Bit 0 = not TERM(NARROW)
		`..1. ....`
		Bit 1 = DBCS, Bit 0 = NODBCS
		`...1 ....`
		Bit 1 = DXREF, Bit 0 = NODXREF
		`.... 1...`
		Bit 1 = FOLD, Bit 0 = NOFOLD
		`.... .1..`
		Bit 1 = SIZE specified, Bit 0 = not specified
		`.... ..1.`
		Bit 1 = FLAG(PUSH), Bit 0 = FLAG(NOPUSH)
		`.... ...1`
		Bit 1 = THREAD, Bit 0 = NOTHREAD

Field	Size	Description

Option Byte 4	XL1	`1... ....`
		Bit 1 = PCONTROL(ON), Bit 0 = not PCONTROL(ON)
		`.1.. ....`
		Bit 1 = PCONTROL(GEN), Bit 0 = not PCONTROL(GEN)
		`..1. ....`
		Bit 1 = PCONTROL(DATA), Bit 0 = not PCONTROL(DATA)
		`...1 ....`
		Bit 1 = PCONTROL(UHEAD), Bit 0 = not PCONTROL(UHEAD)
		`.... 1...`
		Bit 1 = PCONTROL(MSOURCE), Bit 0 = not PCONTROL(MSOURCE)
		`.... .1..`
		Bit 1 = SECTALGN specified, Bit 0 = not specified
		`.... ..1.`
		Reserved
		`.... ...1`
		Reserved

Option Byte 5	XL1	`1... ....`
		Bit 1 = ASA, Bit 0 = NOASA (z/OS and CMS)
		`.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 = INEXIT, Bit 0 = NOINEXIT
		`.... .1..`
		Bit 1 = LIBEXIT, Bit 0 = NOLIBEXIT
		`.... ..1.`
		Bit 1 = PRTEXIT, Bit 0 = NOPRTEXIT
		`.... ...1`
		Bit 1 = OBJEXIT, Bit 0 = NOOBJEXIT

Field	Size	Description

Option Byte 6	XL1	`1... ....`
		Bit 1 = SYSPARM specified, Bit 0 = not specified
		`.1.. ....`
		Bit 1 = FLAG specified, Bit 0 = not specified
		`..1. ....`
		Bit 1 = LANGUAGE specified, Bit 0 = not specified
		`...1 ....`
		Bit 1 = LINECOUNT specified, Bit 0 = not specified
		`.... 1...`
		Bit 1 = OPTABLE/MACHINE specified, Bit 0 = not specified
		`.... .1..`
		Bit 1 = ADATA, Bit 0 = NOADATA
		`.... ..1.`
		Bit 1 = ADEXIT, Bit 0 = NOADEXIT
		`.... ...1`
		Bit 1 = TRMEXIT, Bit 0 = NOTRMEXIT

Option Byte 7	XL1	`1... ....`
		Bit 1 = LIST(121), Bit 0 = not LIST(121) (z/OS and CMS)
		`.1.. ....`
		Bit 1 = LIST(133), Bit 0 = not LIST(133) (z/OS and CMS)
		`..1. ....`
		Bit 1 = LIST(MAX), Bit 0 = not LIST(MAX) (z/OS and CMS)
		`...1 ....`
		Reserved
		`.... 1...`
		Reserved
		`.... .1..`
		Reserved
		`.... ..1.`
		Reserved
		`.... ...1`
		Reserved

Field	Size	Description
Option Byte 8	XL1	`1... ....` Bit 1 = MXREF, Bit 0 = NOMXREF
		`.1.. ....` Bit 1 = MXREF(FULL), Bit 0 = not MXREF(FULL)
		`..1. ....` Bit 1 = MXREF(SOURCE), Bit 0 = not MXREF(SOURCE)
		`...1 ....` Bit 1 = MXREF(XREF), Bit 0 = not MXREF(XREF)
		`.... 1...` Bit 1 = TRANSLATE specified, Bit 0 = NOTRANSLATE
		`.... .1..` Bit 1 = GOFF, Bit 0 = NOGOFF (z/OS and CMS)
		`.... ..1.` Bit 1 = GOFF(ADATA), Bit 0 = GOFF(NOADATA) (z/OS and CMS)
		`.... ...1` Bit 1 = PROFILE specified, Bit 0 = NOPROFILE
Option Byte 9	XL1	`1... ....` Bit 1 = FLAG(RECORD), Bit 0 = FLAG(NORECORD)
		`.1.. ....` Bit 1 = PCONTROL(MCALL), Bit 0 = not PCONTROL(MCALL)
		`..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)

Field	Size	Description

Option Byte 10	XL1	`1... ....`
		Reserved
		`.1.. ....`
		Reserved
		`..1. ....`
		Bit 1 = COMPAT(TRANSDT), Bit 0 = COMPAT(NOTRANSDT)
		`...1 ....`
		Bit 1 = WORKFILE, Bit 0 = not specified
		`.... 1...`
		Bit 1 = OPTABLE/MACHINE(LIST), Bit 0 = (NOLIST)
		`.... .1..`
		Bit 1 = CODEPAGE specified, Bit 0 = not specified
		`.... ..1.`
		Bit 1 = Option errors encountered, Bit 0 = no errors encountered
		`.... ...1`
		Bit 1 = INFO, Bit 0 = NOINFO

Option Byte 11	XL1	`1... ....`
		Bit 1 = FLAG(EXLITW), Bit 0 = FLAG(NOEXLITW)
		`.1.. ....`
		Bit 1 = TYPECHECK(MAGNITUDE), Bit 0 = TYPECHECK(NOMAGNITUDE)
		`..1. ....`
		Reserved
		`...1 ....`
		Bit 1 = TYPECHECK(REGISTER), Bit 0 = TYPECHECK(NOREGISTER)
		`.... 1...`
		Bit 1 = COMPAT(CASE), Bit 0 = COMPAT(NOCASE)
		`.... .1..`
		Bit 1 = COMPAT(SYSLIST), Bit 0 = COMPAT(NOSYSLIST)
		`.... ..1.`
		Bit 1 = COMPAT(LITTYPE), Bit 0 = COMPAT(NOLITTYPE)
		`.... ...1`
		Bit 1 = COMPAT(MACROCASE), Bit 0 = COMPAT(NOMACROCASE)

Field	Size	Description

Option Byte 12	XL1	`1... ....`
		Bit 1 = FLAG(USING0), Bit 0 = (NOUSING0)
		`.1.. ....`
		Bit 1 = LIBMAC, Bit 0 = NOLIBMAC
		`..1. ....`
		Bit 1 = RA2, Bit 0 = NORA2
		`...1 ....`
		Bit 1 = FLAG(ALIGN), Bit 0 = FLAG(NOALIGN)
		`.... 1...`
		Bit 1 = FLAG(CONT), Bit 0 = FLAG(NOCONT)
		`.... .1..`
		Bit 1 = FLAG(SUBSTR), Bit 0 = FLAG(NOSUBSTR)
		`.... ..1.`
		Bit 1 = FLAG(IMPLEN), Bit 0 = FLAG(NOIMPLEN)
		`.... ...1`
		Bit 1 = FLAG(PAGE0), Bit 0 = FLAG(NOPAGE0)

Option Byte 13	XL1	`1... ....`
		Bit 1 = SUPRWARN, Bit 0 = NOSUPRWARN
		`.1.. ....`
		Reserved
		`..1. ....`
		Reserved
		`...1 ....`
		Reserved
		`.... 1...`
		Reserved
		`.... .1..`
		Reserved
		`.... ..1.`
		Reserved
		`.... ...1`
		Reserved

Option Byte 14	XL1	`1... ....`
		Reserved
		`.1.. ....`
		Reserved
		`..1. ....`
		Reserved
		`...1 ....`
		Reserved
		`.... 1...`
		Reserved
		`.... .1..`
		Reserved
		`.... ..1.`
		Reserved
		`.... ...1`
		Reserved

	XL4	Reserved

Field	Size	Description
Extra Byte 1	XL1	**1... ....** Bit 1 = COMPAT, Bit 0 = NOCOMPAT **.1.. ....** Bit 1 = EXIT, Bit 0 = NOEXIT **..1. ....** Bit 1 = PCONTROL, Bit 0 = NOPCONTROL **...1 ....** Bit 1 = PESTOP, Bit 0 = NOPESTOP **.... 1...** Bit 1 = SUBLIB(DF), Bit 0 = SUBLIB(AE) (z/VSE) **.... .1..** Reserved **.... ..1.** Reserved **.... ...1** Reserved
	XL4	Reserved
CODEPAGE value	CL4	Value from CODEPAGE(xxxx) option in effect for the assembly.
FLAG value	FL1	Value from FLAG(n) option in effect for the assembly. Zero if not provided.
LANGUAGE value	CL3	Value from LANGUAGE(xxx) option in effect for the assembly.
LINECOUNT value	HL2	Value from LINECOUNT(n) option in effect for the assembly.
OPTABLE	CL3	Value from OPTABLE(xxx) option in effect for the assembly.
PROFILE value	CL8	Value from PROFILE(xxxxxxxx) option in effect for the assembly. Blank if not provided.
SECTALGN value	FL4	Value from SECTALGN(n) option in effect for the assembly.
TRANSLATE value	CL2	Value from TRANSLATE(xx) option in effect for the assembly. Blank if not provided.
USING(LIMIT) value	HL2	Value from USING(LIMIT(n)) option in effect for the assembly. Zero if not provided.
USING(WARN) value	FL1	Value from USING(WARN(n)) option in effect for the assembly. Zero if not provided.
	XL32	Reserved
PARM offset	FL4	Offset from the beginning of this record to the PARM string supplied
PARM length	FL4	Length of the PARM string supplied
SYSPARM offset	FL4	Offset from the beginning of this record to the SYSPARM string supplied
SYSPARM length	FL4	Length of the SYSPARM string supplied
Input exit name offset	FL4	Offset from the beginning of this record to the INEXIT program name
Input exit name length	FL4	Length of the INEXIT program name
Input exit string offset	FL4	Offset from the beginning of this record to the string supplied to INEXIT

Field	Size	Description
Input exit string length	FL4	Length of string supplied to INEXIT
Library exit name offset	FL4	Offset from the beginning of this record to the LIBEXIT program name
Library exit name length	FL4	Length of the LIBEXIT program name
Library exit string offset	FL4	Offset from the beginning of this record to the string supplied to LIBEXIT
Library exit string length	FL4	Length of string supplied to LIBEXIT
Print exit name offset	FL4	Offset from the beginning of this record to the PRTEXIT program name
Print exit name length	FL4	Length of the PRTEXIT program name
Print exit string offset	FL4	Offset from the beginning of this record to the string supplied to PRTEXIT
Print exit string length	FL4	Length of string supplied to PRTEXIT
Object exit name offset	FL4	Offset from the beginning of this record to the OBJEXIT program name
Object exit name length	FL4	Length of the OBJEXIT program name
Object exit string offset	FL4	Offset from the beginning of this record to the string supplied to OBJEXIT
Object exit string length	FL4	Length of string supplied to OBJEXIT
ADATA exit name offset	FL4	Offset from the beginning of this record to the ADEXIT program name
ADATA exit name length	FL4	Length of the ADEXIT program name
ADATA exit string offset	FL4	Offset from the beginning of this record to the string supplied to ADEXIT
ADATA exit string length	FL4	Length of string supplied to ADEXIT
TERM exit name offset	FL4	Offset from the beginning of this record to the TRMEXIT program name
TERM exit name length	FL4	Length of the TRMEXIT program name
TERM exit string offset	FL4	Offset from the beginning of this record to the string supplied to TRMEXIT
TERM exit string length	FL4	Length of string supplied to TRMEXIT
PARM string	CL(n)	Field to contain the invocation option string that is being used for the assembly
SYSPARM string	CL(n)	Field to contain the SYSPARM string that is being used for the assembly
Input exit name	CL(n)	INEXIT program name
Input exit string	CL(n)	Field to contain the string to be passed to the INEXIT program
Library exit name	CL(n)	LIBEXIT program name
Library exit string	CL(n)	Field to contain the string to be passed to the LIBEXIT program
Print exit name	CL(n)	PRTEXIT program name
Print exit string	CL(n)	Field to contain the string to be passed to the PRTEXIT program
Object exit name	CL(n)	OBJEXIT program name

Field	Size	Description
Object exit string	CL(n)	Field to contain the string to be passed to the OBJEXIT program
ADATA exit name	CL(n)	ADEXIT program name
ADATA exit string	CL(n)	Field to contain the string to be passed to the ADEXIT program
TERM exit name	CL(n)	TRMEXIT program name
TERM exit string	CL(n)	Field to contain the string to be passed to the TRMEXIT program

External Symbol Dictionary Record—X'0020'

Field	Size	Description
Record Type	XL1	X'00' Section Definition (CSECT) SD [illegible] 89.9(ype.60Tj9.333ERTd(-pe.60Tj) TJ

Field	Size	Description
Section Length	FL4	The length of the section
Owner ID	FL4	ESDID of the SD or ED in which this symbol was defined
	XL8	Reserved
External Name offset	FL4	The offset from the beginning of this record to the external name. A value of binary zeros indicates that there is no external name.
External Name length	FL4	Number of characters in the external name (zero if private code, unnamed common, or unnamed DSECT)
Alias Name offset	FL4	The offset from the beginning of this record to the alias name. A value of binary zeros indicates that there is no alias name.
Alias Name length	FL4	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
ESDID	FL4	The ESDID for the source record.
Statement number	FL4	The statement number of the source record.
Input record number	FL4	The input source record number within the current input file. If the source line is macro-generated (that is, the input record origin value is X'02'), this field contains binary zero. 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	FL4	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.

Field	Size	Description
Parent assigned file number	FL4	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.
Input record origin	XL1	**X'01'** Source line from primary input **X'02'** Source line from Macro generation. **X'03'** Source line from library member. **X'04'** Reserved **X'05'** Source line from AINSERT internal buffer.
Parent record origin	XL1	**X'01'** Source line from primary input **X'02'** Source line from Macro generation. **X'03'** Source line from library member. **X'04'** Reserved **X'05'** Source line from AINSERT internal buffer
Print flags	XL1	**X'80'** PRINT GEN **X'40'** PRINT DATA **X'20'** PRINT ON **X'10'** PRINT NOMSOURCE (0 = PRINT MSOURCE) **X'08'** PRINT UHEAD **X'04'** PRINT MCALL
	XL2	Reserved
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.
Assembler operation code	XL1	The assembler operation code for assembler instructions. (See note 2 on page 259). 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
	AL4	Reserved
Address 1	AL4	The address 1 field from the assembly
	AL4	Reserved
Address 2	AL4	The address 2 field from the assembly
Offset of name entry in statement field	FL4	Zero if name entry not present or if the name begins at the beginning of the source record (see note 1 on page 259).
Length of name entry	FL4	Zero if name entry not present (see note 1 on page 259)

Field	Size	Description
Offset of operation entry in statement field	FL4	Zero if operation entry not present (see note 1)
Length of operation entry	FL4	Zero if operation entry not present (see note 1)
Offset of operand entry in statement field	FL4	Zero if operand entry not present (see note 1)
Length of operand entry	FL4	Zero if operand entry not present (see note 1)
Offset of remarks entry in statement field	FL4	Zero if remarks entry not present (see note 1)
Length of remarks entry	FL4	Zero if remarks entry not present (see note 1)
Offset of continuation indicator field	FL4	Zero if no continuation indicator present (see note 1)
	XL4	Reserved
Input macro or copy member name offset	FL4	The offset from the beginning of this record to the input macro or copy member name. A value of binary zeros indicates that there is no input macro or copy member name.
Input macro or copy member name length	FL4	Zero if the input record line does not come from a macro or a copy member
Parent macro or copy member name offset	FL4	The offset from the beginning of this record to the parent macro or copy member name. A value of binary zeros indicates that there is no parent macro or copy member name.
Parent macro or copy member name length	FL4	Zero if the parent record line does not come from a macro or a copy member
Source record offset	FL4	The offset from the beginning of this record to the source record.
Source record length	FL4	The length of the actual source record following
	XL8	Reserved
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 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
2. The assembler operation code field can contain the operation code values shown in Table 38 on page 260.

Table 38. Assembler operation code values

Operation Code	Assembler Instruction	Operation Code	Assembler Instruction	Operation Code	Assembler Instruction
X'00'	GBLA	X'1B'	START	X'36'	Reserved
X'01'	GBLB	X'1C'	CSECT	X'37'	MHELP
X'02'	GBLC	X'1D'	DSECT	X'38'	AREAD
X'03'	LCLA	X'1E'	COM	X'39'	Reserved
X'04'	LCLB	X'1F'	EQU	X'3A'	Reserved
X'05'	LCLC	X'20'	ORG	X'3B'	WXTRN
X'06'	SETA	X'21'	END	X'3C'	Reserved
X'07'	SETB	X'22'	LTORG	X'3D'	AMODE
X'08'	SETC	X'23'	USING	X'3E'	RMODE
X'09'	AIF	X'24'	DROP	X'3F'	RSECT
X'0A'	AGO	X'25'	ACTR	X'40'	CCW0
X'0B'	ANOP	X'26'	DC	X'41'	CCW1
X'0C'	COPY	X'27'	DS	X'42'	EXITCTL
X'0D'	MACRO	X'28'	CCW	X'43'	ASPACE
X'0E'	MNOTE	X'29'	CNOP	X'44'	AEJECT
X'0F'	MEXIT	X'2A'	LOCTR	X'45'	ALIAS
X'10'	MEND	X'2B'	DXD	X'46'	CEJECT
X'11'	ICTL	X'2C'	CXD	X'47'	ADATA
X'12'	ISEQ	X'2D'	Reserved	X'48'	SETAF
X'13'	PRINT	X'2E'	OPSYN	X'49'	SETCF
X'14'	SPACE	X'2F'	PUSH	X'4A'	CATTR
X'15'	EJECT	X'30'	POP		(z/OS & CMS)
X'16'	PUNCH	X'31'	Reserved	X'4B'	ACONTROL
X'17'	REPRO	X'32'	Reserved	X'4C'	XATTR
X'18'	TITLE	X'33'	Literal		(z/OS & CMS)
X'19'	ENTRY	X'34'	Reserved	X'4D'	AINSERT
X'1A'	EXTRN	X'35'	Reserved		

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 Flag	XL1	Flag Byte for Error Message. X'80' Message suppressed
Error message offset	FL4	The offset from the beginning of this record to the error message.
Error message length	FL4	The length of the error message text
	XL8	Reserved
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.

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'0035'). 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

	XL5	Reserved
Statement Number	FL4	The statement number of the source line that generated this text, if known. Otherwise it contains zeros.
Number of operands	FL4	The number of operands defined by the source record. The groups of nine operand fields below occur *n* times depending on the value in this field.
Offset of operands	FL4	The offset from the beginning of this record to the first group of operand fields. A value of binary zeros indicates that there are no operands.
		Start of operand group, one group per operand. The ellipses (...) indicate the fields are grouped.
...Offset of next operand	FL4	The offset from the beginning of this record to the next group of operand fields. A value of binary zeros indicates that there are no more operands.
...Location Counter	FL4	The location counter for this operand.
...Duplication Factor	FL4	The duplication factor for this operand.
...Bit Offset	XL1	The offset within byte (0–7) for bit-length operands.
...Type Attribute	XL1	The value that the assembler Type Attribute reference returns (see “Type attribute (T')” in the *HLASM Language Reference*).
...Type Extension	CL1	The type extension for this operand.

Field	Size	Description
...Program Type	XL4	The value that the assembler SYSATTRP() internal function returns (see "SYSATTRP" in the *HLASM Language Reference*).
	XL4	Reserved
...Number of values	FL4	The number of nominal values defined by this operand. The groups of five nominal-value fields below occur *n* times depending on the value in this field.
...Offset of values	FL4	The offset from the beginning of this record to the first group of nominal-value fields. A value of binary zeros indicates that there are no nominal values.
		End of operand group.
		Start of nominal-value group, one group per value. The ellipses (......) indicate the fields are grouped.
......Offset of next nominal value	FL4	The offset from the beginning of this record to the next group of nominal-value fields. A value of binary zeros indicates that there are no more nominal values.
......Value offset	FL4	The offset from the beginning of this record to the generated nominal value. A value of binary zeros indicates that there is no generated nominal value.
......Byte length	FL4	The byte length of the nominal value, if defined with a byte length.
......Bit length	FL4	The bit length of the nominal value, if defined with a bit length.
......Nominal 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. 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 leftmost 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.
		End of nominal-value group.

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 inter-statement boundary alignment 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 needs to be calculated using the duplication factor, number of nominal values, and the length of each nominal value.
3. High Level Assembler creates the DC 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 (*).

   See page Example6.
4. Where the number of operands exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of operands (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent operands. The count of the number of operands is a count for the current record.

   Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

DC Extension Record—X'0035'

Field	Size	Description
ESDID	FL4	The ESDID for the record
Statement number	FL4	The statement number of the source line that generated this text, if known. Zero otherwise.
Location Counter	FL4	Address (offset) of the text within the module
	XL8	Reserved
Object text offset	FL4	The offset from the beginning of this record to the generated object text.
Object text length	FL4	The length of the following object text
Object text	XL(n)	The actual object text

Note:

Where the amount of object text exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current length of the object text (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent object text. The length of the object text is the length for the current record.

Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces. The exception is the location-counter field which contains the correct location for the start of the continued object text.

Machine Instruction Record—X'0036'

Field	Size	Description
ESDID	FL4	The ESDID for the machine instruction record
Statement number	FL4	The statement number of the source record
Location Counter	FL4	The location counter for this instruction
	XL8	Reserved

Field	Size	Description
Instruction offset	FL4	The offset from the beginning of this record to the machine instruction.
Instruction length	FL4	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.
	AL4	Reserved
Address	AL4	The assembled address of the field where the address constant is stored. Not all relocation types supported by GOFF object files are reported in the Relocation Dictionary Record.

Flags	XL1	**`x... ....`**
		Reserved
		`.1.. ....`
		Add 4 to length of ADCON
		`..11 ....`
		ADCON type - CXD (3)
		`..10 ....`
		ADCON type - Q (2)
		`..01 ....`
		ADCON type - V (1)
		`..00 ....`
		ADCON type - A (0)
		`.... 11..`
		Contains length-1 of ADCON
		`.... ..1.`
		Relocation sign: 1 = -, 0 = +
		`.... ...1`
		Following RLD item has same R and P pointer

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.
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. This is zero for an undefined symbol type.

Field	Size	Description
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
Duplication Factor	FL4	Number of times the first operand field named by the symbol occurs. This is zero for an undefined symbol type.
Type Attribute	XL1	The value that the assembler Type Attribute reference returns (see "Type attribute (T')" in the *HLASM Language Reference*).
Assembler Type	CL4	The value that the assembler SYSATTRA() internal function returns (see "SYSATTRA" in the *HLASM Language Reference*).
Program Type	XL4	The value that the assembler SYSATTRP() internal function returns (see "SYSATTRP" in the *HLASM Language Reference*).
Length attribute	FL4	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.
Scaling 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.

Symbol Flags	XL1	**`1... ....`**
		Bit 1 = 1, the symbol is relocatable, Bit 0 = the symbol is absolute. This bit is zero for an undefined symbol type.
		`11.. ....`
		Complex relocatable
		`..1. ....`
		Reserved
		`...1 ....`
		Reserved
		`.... 1...`
		Reserved
		`.... .1..`
		Reserved
		`.... ..1.`
		Reserved
		`.... ...1`
		Reserved

	XL7	Reserved

Field	Size	Description
Symbol name offset	FL4	The offset from the beginning of this record to the symbol name.
Symbol name length	FL4	Number of characters in the symbol name
Symbol name	CL(n)	The symbol name.

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 is provided. The symbol flags can be checked to determine whether the “EQU” is relocatable or absolute.

Symbol and Literal Cross Reference Record—X'0044'

Field	Size	Description
Statement Number	FL4	The statement number where the symbol or literal is defined or declared
Relocatability Type	CL1	`C' '` Simple relocatable symbol `C'A'` Absolute symbol `C'C'` Complex relocatable symbol
	XL7	Reserved
Name offset	FL4	The offset from the beginning of this record to the symbol or literal name.
Name length	FL4	The length of the symbol or literal name
Total references	FL4	The total number of references to the symbol or literal for the assembly
Number of references	FL4	The number of references to the symbol or literal in this record The groups of two reference fields below occur *n* times depending on the value in this field. The reference groups are contiguous so they might be treated as an array.
Offset of references	FL4	The offset from the beginning of this record to the first group of reference fields. A value of binary zeros indicates that there are no references.
Name	CL(n)	The symbol or literal name.
		Start of reference groups, one group per reference. The ellipses (...) indicate the fields are grouped.
...Statement Number	FL4	The statement number on which the symbol or literal is referenced
...Reference Flag	CL1	`C' '` No branch or modification `C'M'` Modification `C'B'` Branched to `C'U'` USING statement `C'D'` DROP statement `C'X'` Target of an execute instruction
		End of reference groups.

Field	Size	Description

Note:

Where the number of references exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of references (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent references. The count of the number of references is a count for the current record.

Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

Register Cross Reference Record—X'0045'

Field	Size	Description
Register number	XL1	The register number (X'0' to X'F')
Register Type	CL1	G—General
	XL2	Reserved
Total references	FL4	The total number of references to the register for the assembly
Number of references	FL4	The number of references to the register in this record The groups of two reference fields below occur *n* times depending on the value in this field. The reference groups are contiguous so they might be treated as an array.
Offset of references	FL4	The offset from the beginning of this record to the first group of reference fields. A value of binary zeros indicates that there are no references.
		Start of reference groups, one group per reference. The ellipses (...) indicate the fields are grouped.
...Statement Number	FL4	The statement number on which the register is referenced
...Reference Flag	CL1	`C' '` No branch or modification `C'M'` Modification `C'B'` Branched to `C'U'` USING statement `C'D'` DROP statement `C'N'` Index register
		End of reference groups.

Note:

Where the number of references exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of references (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent references. The count of the number of references is a count for the current record.

Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

Library Record—X'0060'

Field	Size	Description
Concatenation number	FL4	The library concatenation number
Library name offset	FL4	The offset from the beginning of this record to the library (file) name
Library name length	FL4	The length of the library (file) name
Volume serial number offset	FL4	The offset from the beginning of this record to the volume serial number
Volume serial number length	FL4	The length of the volume serial number
DDNAME offset	FL4	The offset from the beginning of this record to the DDNAME
DDNAME length	FL4	The length of the DDNAME
Number of members	FL4	The number of macros and copy code members in this record The groups of three member fields below occur *n* times depending on the value in this field.
Offset of members	FL4	The offset from the beginning of this record to the first group of member fields. A value of binary zeros indicates that there are no members.
Library name	CL(n)	The name of the library (file) from which the macro or copy member was retrieved, or "PRIMARY INPUT" for an in-stream macro. Under z/VSE, this field contains the library and sublibrary name.
Volume serial number	CL(n)	The volume serial number of the (first) volume on which the library resides
DDNAME	CL(n)	The DDNAME of the library.
		Start of member groups, one group per member. The ellipses (...) indicate the fields are grouped.
...Offset of next member	FL4	The offset from the beginning of this record to the next group of member fields. A value of binary zeros indicates that there are no more members.
...Member name offset	FL4	The offset from the beginning of this record to the macro or copy code member name
...Member name length	FL4	The length of the macro or copy code member name
...Member name	CL(n)	The name of the macro or copy code member that has been used. If the source is "PRIMARY INPUT", then this field contains the macro name from the source program.
		End of member groups.

Note:

1. 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.
2. Where the number of members exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of members (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent members. The count of the number of members is a count for the current record.

   Fields that have been written not be repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.

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 Number	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 embedded in the primary source, using the LIBMAC option `C' '` The flag is blank except in special cases, as described above
	XL8	Reserved
Member or macro name offset	FL4	The offset from the beginning of this record to the member or macro name
Member or macro name length	FL4	The length of the member or macro name
Parent name offset	FL4	The offset from the beginning of this record to the parent (caller) member or macro name
Parent name length	FL4	The length of the parent (caller) member or macro name
Total references	FL4	The total number of references to the member or macro for the assembly
Number of references	FL4	The number of references to the member or macro by the parent. The groups of two reference fields below occur *n* times depending on the value in this field. The reference groups are contiguous so they might be treated as an array.
Offset of references	FL4	The offset from the beginning of this record to the first group of reference fields. A value of binary zeros indicates that there are no references.
Member or macro name	CL(n)	The name of the member or macro.
Parent macro name	CL(n)	The name of the macro that called this macro or issued the COPY instruction. This field contains "PRIMARY INPUT" when the member or macro is called directly from the primary input file.
		Start of reference groups, one group per reference. The ellipses (...) indicate the fields are grouped.
...Statement Number	FL4	The statement number on which the member is copied or included, or the statement number on which the macro is called
...Reference Flag	CL1	`C' '` Blank means that the reference is caused by a macro call `C'C'` Reference is caused by a COPY instruction
		End of reference groups.

Field	Size	Description
Note: Where the number of references exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of references (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent references. The count of the number of references is a count for the current record. Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.		

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 offset	FL4	The offset from the beginning of this record to the user data
User data length	FL4	The length of the user data
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	FL4	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.
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.

Field	Size	Description
USING ESDID	FL4	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.
	XL1	Reserved
USING range	FL4	The value of the USING range.
	XL2	Reserved
Label offset	FL4	The offset from the beginning of this record to the label and USING text. A value of binary zeros indicates that there is no label or USING text.
Label length	FL4	The length of the label and USING text. This length field is rounded up to a doubleword boundary. Hence if the text is 13 bytes in length, the length is set at 16 and the text space padded on the right. This is zero for PUSH and POP.
Label	CL(n)	The source text for the label and USING from the source USING record. This field is null for PUSH and POP.

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 an in-storage assembly.
Primary input records	FL4	The number of primary input records read for the assembly row.
Library records	FL4	The number of library records read for the assembly row.
Work file reads	FL4	The number of work file reads for the assembly row.
Print records written	FL4	The number of print records written for the assembly row.
Object records written	FL4	The number of object records written for the assembly row.
Work file writes	FL4	The number of work file writes for the assembly row.
ADATA file writes	FL4	The number of associated data (ADATA) file writes for the assembly.
ADATA calls	FL4	The number of calls to the associated data (ADATA) exit. This field is zero if no exit is present.

Field	Size	Description
ADATA added records	FL4	The number of records added by the associated data (ADATA) exit. This field is zero if no exit is present.
ADATA deleted records	FL4	The number of records deleted by the associated data (ADATA) exit. This field is zero if no exit is present.
ADATA diagnostic messages	FL4	The number of diagnostic messages returned by the associated data (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.
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. (z/OS and CMS) This field is zero if no exit is present. Reserved (z/VSE)
Object added records	FL4	The number of records added by the OBJECT exit. (z/OS and CMS) This field is zero if no exit is present. Reserved (z/VSE)
Object deleted records	FL4	The number of records deleted by the OBJECT exit. (z/OS and CMS) This field is zero if no exit is present. Reserved (z/VSE)

Field	Size	Description
Object diagnostic messages	FL4	The number of diagnostic messages returned by the OBJECT exit. (z/OS and CMS) This field is zero if no exit is present. Reserved (z/VSE)
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.
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

Field	Size	Description
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 used by this assembly. (z/OS and CMS) The low-order bit represents 1 microsecond. Reserved (z/VSE)
ASMAOPT input records	FL4	The number of ASMAOPT input records read for the assembly
	XL4	Reserved
Number of functions	FL4	The number of external functions in this record The groups of eight external-function fields below occur *n* times depending on the value in this field.
Offset of functions	FL4	The offset from the beginning of this record to the first group of external-function fields. A value of binary zeros indicates that there are no external functions.
		Start of external-function groups, one group per function. The ellipses (...) indicate the fields are grouped.
...Offset of next external function	FL4	The offset from the beginning of this record to the next group of external-function fields. A value of binary zeros indicates that there are no more external functions.
...	XL4	Reserved
...SETAF function calls	FL4	The number of times the function was called from a SETAF assembler instruction.
...SETCF function calls	FL4	The number of times the function was called from a SETCF assembler instruction.
...Messages issued	FL4	The number of times the function requested that a message be issued
...Messages severity	HL2	The maximum severity for the messages issued by this function
...External function name offset	FL4	The offset from the beginning of this record to the external function name
...External function name length	FL4	The length of the external function name
...External function name	CL(n)	The external function module name
		End of external-function groups.

Field	Size	Description
Note: Where the number of functions exceeds the record size for the associated data file, the record is continued on the next record. The continuation flag is set in the common header section of the record. The current number of functions (for that record) is stored in the record and the record written to the associated data file. The next record contains the subsequent functions. The count of the number of functions is a count for the current record. Fields that have been written are not repeated in the next record. Fixed-length fields are initialized to binary zeros or spaces, and variable-length fields have a length of binary zeros.		

Appendix D. 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 these descriptions refer to corresponding characters in the listing that precedes the descriptions.

For more details about this program, see “ASMASAMP” in the *HLASM Installation and Customization Guide*.

```
                                        High Level Assembler Option Summary                                  Page    1
                                                                                                  HLASM R6.0  2008/07/11 17.48
  No Overriding ASMAOPT Parameters
  No Overriding Parameters
  No Process Statements


  Options for this Assembly
  NOADATA
    ALIGN
  NOASA
    BATCH
    CODEPAGE(047C)
  NOCOMPAT
  NODBCS
  NODECK
    DXREF
    ESD
  NOEXIT
    FLAG(0,ALIGN,CONT,EXLITW,NOIMPLEN,NOPAGE0,PUSH,RECORD,NOSUBSTR,USING0)
  NOFOLD
  NOGOFF
  NOINFO
    LANGUAGE(EN)
  NOLIBMAC
    LINECOUNT(60)
    LIST(121)
    MACHINE(,NOLIST)
    MXREF(SOURCE)
    OBJECT
    OPTABLE(UNI,NOLIST)
  NOPCONTROL
  NOPESTOP
  NOPROFILE
  NORA2
  NORENT
    RLD
    RXREF
    SECTALGN(8)
    SIZE(MAX)
  NOSUPRWARN
    SYSPARM()
  NOTERM
  NOTEST
    THREAD
  NOTRANSLATE
    TYPECHECK(MAGNITUDE,REGISTER)
    USING(NOLIMIT,MAP,WARN(15))
    XREF(SHORT,UNREFS)

  No Overriding DD Names
```

© Copyright IBM Corp. 1992, 2013

```
BIGNAME                                       External Symbol Dictionary                                       Page    2
Symbol   Type   Id     Address  Length   Owner Id Flags Alias-of                            HLASM R6.0  2008/07/11 17.48
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 R6.0  2008/07/11 17.48
                                      2 **************************************************************          00002000
                                      3 *   Licensed Materials - Property of IBM                     *          00003000
                                      4 *                                                            *          00004000
                                      5 *   5696-234   5647-A01                                      *          00005000
                                      6 *                                                            *          00006000
                                      7 *   (C) Copyright IBM Corp. 1992, 2000. All Rights Reserved. *          00007000
                                      8 *                                                            *          00008000
                                      9 *   US Government Users Restricted Rights - Use,             *          00009000
                                     10 *   duplication or disclosure restricted by GSA ADP          *          00010000
                                     11 *   Schedule Contract with IBM Corp.                         *          00011000
                                     12 *                                                            *          00012000
                                     13 **************************************************************          00013000
                                     14 *********************************************************************   00014000
                                     15 * DISCLAIMER OF WARRANTIES                                          *   00015000
                                     16 *  The following enclosed code is sample code created by IBM        *   00016000
                                     17 *  Corporation. This sample code is licensed under the terms of     *   00017000
                                     18 *  the High Level Assembler license, but is not part of any         *   00018000
                                     19 *  standard IBM product.  It is provided to you solely for the      *   00019000
                                     20 *  purpose of demonstrating the usage of some of the features of    *   00020000
                                     21 *  High Level Assembler.  The code is not supported by IBM and      *   00021000
                                     22 *  is provided on an "AS IS" basis, without warranty of any kind.   *   00022000
                                     23 *  IBM shall not be liable for any damages arising out of your      *   00023000
                                     24 *  use of the sample code, even if IBM has been advised of the      *   00024000
                                     25 *  possibility of such damages.                                     *   00025000
                                     26 *********************************************************************   00026000
000000                00000 000DE    27 a        csect                                                          00027000
                 R:8  00000          28          using *,8                                                      00028000
000000 1BFF                          29          sr    15,15      Set return code to zero                       00029000
000002 07FE                          30          br    14          and return.                                  00030000
                                     32 ********************************************************************** 00032000
                                     33 *              PUSH  and POP  statements                             * 00033000
                                     34 * Push down the PRINT statement, replace it, retrieve original       * 00034000
                                     35 ********************************************************************** 00035000
                                     37          push  print     Save Default setting '  PRINT ON,NODATA,GEN'   00037000
                                B    38          print nogen,data                                               00038000
000004 0A23                          39          wto   mf=(E,(1))                    Expansion not shown        00039000
```

A The external symbol dictionary shows a named common statement. The named common section is defined in statement 216.

B Statement 37: Save the status of the PRINT statement.

Statement 38: Modify the print options to DATA and NOGEN.

Statement 39: Macro call; the expansion (statements 40 and 41) is not printed.

Statement 42: All 28 bytes of data are displayed to the two-operand DC.

Statement 43: Restore earlier status of PRINT.

Statement 45: This statement is not printed. It is a nested macro call. The MCALL operand of the PRINT instruction or the PCONTROL assembler option control the printing of nested macro calls.

Statements 46: The generated output of the macro WTO is shown, but only two bytes of data are shown.

```
000006 01230ABC0102030A           C  42  dc x'123,ABC',(reallylongsymbol-transylvania)b'1,10,11,1010,1011,1100' 00040000
00000E 0B0C0102030A0B0C
000016 0102030A0B0C0102
00001E 030A0B0C
                                     43          pop   print                   Restore default PRINT setting    00041000
                                     44          wto   mf=(E,(1))                    Expansion shown            00042000
000022 0A23                          46+         SVC   35                       ISSUE SVC 35               @L2C 01-WTO
000024 01230ABC0102030A              47  dc x'123,ABC',(reallylongsymbol-transylvania)b'1,10,11,1010,1011,1100' 00043000
                                     49 **********************************************************************  00045000
                                     50 *                     LOCTR  instruction                             *  00046000
                                     51 * LOCTR allows 'REMOTE' assembly of constant                         *  00047000
                                     52 **********************************************************************  00048000
000040 5850 80AC            000AC    54          l     5,constant                                               00050000
0000AC                000AC 000DE  D 55 deecees  loctr                                                          00051000
0000AC 00000005                      56 constant dc    f'5'      Constant coded here, assembled behind LOCTR A  00052000
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 R6.0  2008/07/11 17.48
000044                00000 000DE    57 a        loctr                Return to 1st LOCTR in CSECT A            00053000
                                     59 **********************************************************************  00055000
                                     60 *  3 operand EQUATE with forward reference in 1ST operand            *  00056000
                                     61 **********************************************************************  00057000
000044 1812                          63 a5       lr    1,2            L'A5 = 2, T'A5 = I                        00059000
                                     64          print data                                                     00060000
000046 0000
000048 413243F6A8885A30              65 a7 dc l'3.14159265358979323846264338327950288841972' L'A7 = 16,T'A7 = L  00061000
000050 338D313198A2E037
                                     66 &type    setc  t'a7                                                     00062000
                                E    67 a8       equ   b5,l'a5,c'&type'                                         00063000
                      000B0 00002      +a8       equ   b5,l'a5,c'L'                                             00063000
```

C Statements 42 and 47: 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 144 and 145.

D Statements 55, 57, 194, 212, and 213 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 sections. LOCTR has many of the features of a control section; for example, all 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 27, 57, and 213. The location counter value shown each time is the continued value of the LOCTR. Conversely, 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 (at statement 56) is in LOCTR DEECEES but the instruction referring to it (statement 54) 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 second or third 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 operand field must be previously defined. You cannot express the type attribute of A7 directly in the EQU statement. The EQU statement at 67 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    5
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                     69 ********************************************************************** 00065000
                                     70 * Implicit declaration of locals &A, &C -- Use of SETC dup factor to * 00066000
                                     71 *   produce SETC string longer than 8, MNOTE in open code            * 00067000
                                     72 ********************************************************************** 00068000
                                     74 &la8     seta  l'a8                                                    00070000
                                F    75 &ta8     setc  t'a8                                                    00071000
                                     76          mnote *,'Length of A8 = &LA8, Type of A8 = &TA8'              00072000
                                G      +*,Length of A8 = 2, Type of A8 = L                                     00072000
                                     78 &a       seta  2                                                       00074000
                                     79 &c       setc  (&a+3)'STRING,'                                         00075000
                                H    80          mnote *,'&&C has value = &c'                                  00076000
                                       +*,&C has value = STRING,STRING,STRING,STRING,STRING,                   00076000
                                     82 ********************************************************************** 00078000
                                I    83 * Examples of 4 byte self-defined terms, unary + and -               * 00079000
                                     84 ********************************************************************** 00080000
000058 7FFFFFFFC1C2C3C4              86          dc    a(2147483647,C'ABCD',X'ffffffff')                       00082000
000060 FFFFFFFF
000064 181D                          87          lr    -1+2,16+-3                                              00083000
                     FFFFE8          89 X        equ   4*-6                                                    00085000
```

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 86 through 89 show 4-byte self-defining values and unary + and -. The value of X appears later in a literal address constant (see statement 296).

```
BIGNAME  Insert Programmer Macro in Source Stream now                                                          Page    6
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                     91 *********************************************************************** 00087000
                                     92 *  Mixed keywords and positional parameters, extended AGO and AIF    *  00088000
                                     93 *     statements, declaration and use of subscripted SET symbols,    *  00089000
                                     94 *       Use of created SET symbols, extended SET statements          *  00090000
                                     95 *********************************************************************** 00091000
                                J    97          macro                                                          00093000
                                     98          demo  &p1,&key1=A,&p2,&key2=1,&p3,&key3=3,&p4                  00094000
                                K    99 &loc(1)  setc  '2','3'        &LOC is dimensioned LCLC by default       00095000
                                    100          gblc  &xa(5),&xb(20),&xc(1)                                    00096000
                                    101          aif   ('&system_id'(1,3) eq 'VSE').vse                         00097000
                                L   102          &p1   &syslist(4),&syslist(5),&syslist(6),mf=E                 00098000
                                    103          ago   .notvse                                                  00099000
                                    104 .vse     anop                 Use VSE WRITE macro parameters            00100000
                                    105          &p1   &syslist(4),SQ,&syslist(6)                               00101000
                                    106 .notvse  anop                                                           00102000
                                    107 &n       seta  1                                                        00103000
                                M   108          ago   (&key2).mnote1,.mnote2,.mnote3                           0010400
                                    109 &n       seta  2                                                        00105000
                                    110          mnote *,'&&KEY2 not 1,2, or 3---Use &&KEY3 in place of it'     00106000
                                N   111          aif   (&key3 eq 1).mnote1,                                    X00107000
                                                       (&key3 eq 2).mnote2,(&key3 eq 3).mnote3                   00108000
                                    112          mnote *,'Both &&KEY2 and &&KEY3 fail to qualify'               00109000
                                    113          ago   .common                                                  00110000
                                    114 .mnote1  mnote *,'&&KEY&LOC(&N) = 1'                                    00111000
                                    115          ago   .common                                                  00112000
                                    116 .mnote2  mnote *,'&&KEY&LOC(&N) = 2'                                    00113000
                                    117          ago   .common                                                  00114000
                                    118 .mnote3  mnote *,'&&KEY&LOC(&N) = 3'                                    00115000
                                    119 .common  l     5,8(,10)       Note that opcodes, operands & comments    00116000
                                    120 &xb(2) sr 9,10                  on MODEL statements                     00117000
                                O   121 &(x&key1)(2) lm 12,13,=a(a5,x)    are kept in place unless displaced    00118000
                                    122 &p2 st 7,&p3                        as a result of substitution         00119000
                                    123          mend                                                           00120000
                                    125 *****          DEMO  MACRO  instruction (call)                          00122000
                                P   127          gblc  &xa(1),&xb(2),&xc(3)                                     00124000
                                    128 &xa(1)   setc  'A','MISSISSIPPI'                                        00125000
                                    129 &xb(1)   setc  'B','SUSQUEHANNA'                                        00126000
                                    130 &xc(1)   setc  'C','TRANSYLVANIA'                                       00127000
                                Q   131          demo  key3=2,write,reallylongsymbol,                          M00128000
                                                       a8+8*(b5-constant-7)(3),key1=C,(6),SF,                  N00129000
                                                       (8),key2=7                                               00130000
000066 1816                         134+         LR    1,6                               LOAD DECB ADDRESS      03-IHBRD
000068 9220 1005      00005         135+         MVI   5(1),X'20'               SET TYPE FIELD                  03-IHBRD
00006C 5081 0008            00008   136+         ST    8,8(1,0)                          STORE DCB ADDRESS      03-IHBRD
000070 58F1 0008            00008   137+         L     15,8(1,0)          LOAD DCB ADDRESS                      03-IHBRD
000074 58F0 F030            00030   138+         L     15,48(0,15)                       LOAD RDWR ROUTINE ADDR 03-IHBRD
000078 05EF                         139+         BALR  14,15                             LINK TO RDWR ROUTINE   03-IHBRD
                                    140+*,&KEY2 not 1,2, or 3---Use &KEY3 in place of it                        01-00110
                                    141+*,&KEY3 = 2                                                             01-00116
00007A 5850 A008            00008   142+         l     5,8(,10)       Note that opcodes, operands & comments    01-00119
00007E 1B9A                     R   143+SUSQUEHANNA sr 9,10             on MODEL statements                     01-00120
000080 98CD 8090            00090   144+TRANSYLVANIA lm 12,13,=a(a5,x)    are kept in place unless displaced    01-00121
BIGNAME  Insert Programmer Macro in Source Stream now                                                          Page    7
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
000084 5073 8098            00098   145+reallylongsymbol st 7,a8+8*(b5-constant-7)(3)                            X01-00122
                                       +                                    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 can be used. The positional parameters are identified and numbered 1, 2, 3 from left to right; keywords are skipped over in numbering positional parameters.

K Statement 99 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 102 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 is treated as a macro, it is still expanded as a machine or assembler operation.

M Statement 108 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 109.

N Statement 111 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 121 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 131, 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 121, &(x&key1)(2), becomes &XC(2). Statement 130 assigns the value C to &XC(1), and the value `TRANSYLVANIA` to &XC(2). The model statement (121) is generated at statement 144; the name field contains `TRANSYLVANIA`. The sequence field of statement 144, shows that this statement is a level 01 expansion of a macro, and the corresponding model statement is statement number 121.

You can use created SET symbols wherever regular SET symbols are used; for example: in declarations, name fields, operands of SET statements, model statements. Likewise, they are subject to all the restrictions of regular SET symbols.

P In statements 127 and 128, &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 131 is the macro instruction DEMO. &P1 has the value WRITE. Therefore, the model statement at statement 102 becomes an inner macro instruction, WRITE, producing the code at statements 134–139. 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 144 and 145 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).

```
BIGNAME  Insert Programmer Macro in Source Stream now                                                          Page    8
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                    147 **********************************************************************  00132000
                                    148 *  Copy 'NOTE' macro in from maclib, rename it 'MARK', call it under *  00133000
                                    149 *    its ALIAS -- in expansion of MARK, notice reference back to     *  00134000
                                    150 *      definition statements in 'columns' 76-80 of expansion         *  00135000
                                    151 **********************************************************************  00136000
                                S   153          copy  note                                                   00138000
                                    154=         MACRO                                                          00010000
                                    155=&NAME    NOTE  &DCB,&DUMMY=,&TYPE=REL                                   00020000
                                    156=.* $MAC(NOTE):                                                          00030000
                                    157=.*  5665-XA2                                                            00032000
                                    158=.*  CONTAINS RESTRICTED MATERIALS OF IBM                                00034000
                                    159=.*  (C) COPYRIGHT IBM CORP. 1984                                        00036000
                                    160=.*  LICENSED MATERIALS - PROPERTY OF IBM                                00038000
                                    161=.*  REFER TO COPYRIGHT INSTRUCTIONS                                     00040000
                                    162=.*  FORM NUMBER G120-2083.                                              00042000
                                    163=.* STATUS =  MVS/XA  DFP RELEASE 1.2                              @H1   00044000
                                    164=.*                                                                      00046990
                                    165=.* CHANGE ACTIVITY =                                                    00120000
                                    166=.*                                                                      00130000
                                    167=.* $H1=3480,JDP1111,,STLPKH:  3480 SUPPORT                           *  00140000
                                    168=.*                                                                      00150000
                                    169=         AIF   ('&DCB' EQ '').ERR                                       00160000
                                    170=&NAME    IHBINNRA &DCB                                                  00170000
                                    171=         AIF   ('&TYPE' NE 'REL').NOTREL                           @H1A 00180000
                                    172=         L     15,84(0,1)                        LOAD NOTE  RTN ADDRESS 00190000
                                    173=         BALR  14,15                             LINK TO NOTE  ROUTINE  00200000
                                    174=         MEXIT                                                          00210000
                                    175=.NOTREL  AIF   ('&TYPE' NE 'ABS').ERR1                             @H1A 00220000
                                    176=         SLR   0,0                      INDICATES NOTE MACRO       @H1A 00230000
                                    177=         LA    15,32                    ROUTER CODE                @H1A 00240000
                                    178=         SVC   109                      SUPERVISOR CALL            @H1A 00250000
                                    179=         MEXIT                                                     @H1A 00260000
                                    180=.ERR1    MNOTE 8,'INVALID PARAMETER FOR TYPE'                 @H1A      00270000
                                    181=         MEXIT                                                     @H1A 00280000
                                    182=.ERR     IHBERMAC 6                                                     00290000
                                    183=         MEND                                                           00300000
                                T   186 mark     opsyn note      Comments of generated statements occupy same   00141000
                                    187          mark  (6)         'COLUMNS' as those in MODEL statements       00142000
000088 1816                         189+         LR    1,6                               LOAD PARAMETER REG 1   02-IHBIN
00008A 58F0 1054            00054   190+         L     15,84(0,1)                        LOAD NOTE  RTN ADDRESS 01-00172
00008E 05EF                         191+         BALR  14,15                             LINK TO NOTE  ROUTINE  01-00173
                                    193 **********************************************************************  00144000
0000B0                000AC 000DE   194 deecees  loctr        Switch to alternate location counter              00145000
0000B0 0B0000B000000050             195 b5       ccw   X'0b',b5,0,80                                            00146000
                                    197 **********************************************************************  00148000
                                    198 *  Display of &SYSTIME, &SYSDATE, &SYSPARM and &SYSLOC               *  00149000
                                    199 **********************************************************************  00150000
                                    201          print nodata                                                   00152000
                                U   202   dc  c'TIME = &systime, DATE = &sysdate, PARM = &sysparm'              00153000
BIGNAME  Insert Programmer Macro in Source Stream now                                                          Page    9
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
0000B8 E3C9D4C5407E40F1                +  dc  c'TIME = 17.22, DATE = 06/09/04, PARM = '                         00153000
                                    204          macro                                                          00155000
                                    205          locate                                                         00156000
                                V   206 &sysect  csect     Display of current control section                   00157000
                                    207 &sysloc  loctr         and location counter                             00158000
                                    208          mend                                                           00159000
                                    210          locate                                                         00161000
0000DE                000AC 000DE   211+a        csect     Display of current control section                   01-00206
0000DE                000AC 000DE   212+deecees  loctr         and location counter                             01-00207
000090                00000 000DE   213 a        loctr                                                          00162000
```

S Library macros can be inserted into the source stream as programmer macros by use of a COPY statement. The result (statements 154 to 183) is treated as a source-stream macro definition. When a library macro is brought in and expanded by use of a macro instruction, the assembler:

1. Looks up the macro by its member-name.
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 186, MARK is made a synonym for NOTE. To identify the NOTE macro as a defined instruction mnemonic, 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 154 through 183 is a programmer macro definition. The macro instruction at statement 187 is MARK. You can use MARK and NOTE interchangeably. If required, you could remove NOTE as a macro definition in the following way:

```
MARK      OPSYN          NOTE
NOTE      OPSYN          ,
```

You could then refer to the macro only as MARK.

U Statement 202 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 at statements 211 and 212. The sequence field indicates that the model statements are statements 206 and 207.

```
BIGNAME  Ordinary, Labeled and Dependent USING Instructions                                                    Page   10
  Active Usings: a,R8
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                    215 *********************************************************************** 00164000
000000                00000 00814 W 216 pd2      com                  Named COMMON thrown in for good measure   00165000
000000                              217          ds    500f                                                     00166000
0007D0 1867                         218          lr    6,7                                                      00167000
                                    220 *********************************************************************** 00169000
                                    221 *  Use of ordinary, labeled and dependent USING Instructions         * 00170000
                                  X 222 *********************************************************************** 00171000
                 R:C  007D2         224          using *,12                                                     00173000
0007D2 4110 C022            007F4   225          la    1,area1                                                  00174000
0007D6 4120 C032            00804   226          la    2,area2                                                  00175000
                 R:1  00000         227          using first,1                 Ordinary USING                   00176000
                 R:2  00000         228 lab      using first,2                 Labeled USING                    00177000
               1 008  00000 00008   229          using second,first2           Dependent USING                  00178000
               2 008  00000 00008   230 labdep   using third,lab.first2        Labeled dependent USING          00179000
0007DA D207 1000 8098 00000 00098   231          mvc   first1,=cl8'1st'        Uses ordinary USING              00180000
0007E0 D207 2000 8098 00000 00098   232          mvc   lab.first1,=cl8'1st'    Uses labeled USING               00181000
0007E6 D203 1008 80A0 00000 000A0   233          mvc   second1,=cl4'2nd'       Uses dependent USING             00182000
0007EC D201 2008 80A4 00000 000A4   234          mvc   labdep.third1,=cl2'3d'  Uses labeled dependent USING     00183000
0007F4                              235 area1    ds    0f                      First data area                  00184000
0007F4                              236 area1a   ds    cl8                                                      00185000
0007FC                              237 area1b   ds    cl8                                                      00186000
000804                              238 area2    ds    0f                      Second data area                 00187000
000804                              239 area2a   ds    cl8                                                      00188000
00080C                              240 area2b   ds    cl8                                                      00189000
000000                00000 00010   241 first    dsect                         First dsect                      00190000
000000                              242 first1   ds    cl8                                                      00191000
000008                              243 first2   ds    cl8                                                      00192000
000000                00000 00008   244 second   dsect                         Second dsect                     00193000
000000                              245 second1  ds    cl4                                                      00194000
000004                              246 second2  ds    cl4                                                      00195000
000000                00000 000EC   247 third    dsect                         Third dsect                      00196000
000000                              248 third1   ds    cl2                                                      00197000
000002                              249 third2   ds    cl2                                                      00198000
```

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 within the section *after* the COMMON statement.

X In statement 227, an ordinary USING is established for AREA1 using the DSECT FIRST. When the fields within DSECT FIRST are referenced using symbols with the “first” qualifier, register 1 is used to resolve the address as in statement 231.

In statement 228, a labeled USING is established for AREA2 using the DSECT FIRST. Register 2 is used to resolve the address for qualified symbols within AREA2 when referred to using the qualifier “second” as in statement 232.

In statement 229, a dependent USING is established at the field FIRST2 using the DSECT SECOND. The corresponding ordinary USING for field FIRST2 is the USING on statement 227. It uses register 1 to resolve the address. The statement on line 233 specifies a field within DSECT SECOND and the assembler uses register 1 to resolve the address.

In statement 230, a labeled dependent USING is established at the field FIRST2 using the DSECT THIRD. The USING specifies the labeled USING LAB to resolve the address for field FIRST2. In statement 234, 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   11
  Active Usings: first,R1  second(X'FF8'),R1+X'8'  a,R8  pd2+X'7D2',R12  lab.first,R2  labdep.third(X'FF8'),R2+X'8'
D-Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                    251 *********************************************************************** 00200000
                                    252 *  Use of predefined absolute symbols in SETA and SETC expressions   * 00201000
                                    253 *********************************************************************** 00202000
                      00064         255 hundred  equ   100                                                      00204000
                                    256 &dividnd seta  20                                                       00205000
                                Y   257 &percent seta  &dividnd*100/40          Predefined symbol in SETA        00206000
                      00032         258 fifty    equ   50                                                       00207000
                                Z   259 &longwd  setc  (hundred)'a'            Predefined symbol in SETC        00208000
                                    260          dc    c'&longwd'                                              00209000
000004 8181818181818181                +         dc    c'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaX00209000
00000C 8181818181818181                +               aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
                                    261 &twowds  setc  (fifty)'a'.' '.(hundred/2)'B'                            00210000
                                    262          dc    c'&twowds'                                              00211000
000068 8181818181818181                +         dc    c'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa BBBX00211000
000070 8181818181818181                +               BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB'
```

Y In statement 257, the SETA statement specifies a variable symbol (&DIVIDND) as well as other arithmetic terms.

Z In statement 259 the SETC statement specifies a predefined absolute symbol (HUNDRED) as the duplication factor.

```
BIGNAME  Symbol Attribute Enhancements                                                                          Page   12
  Active Usings: first,R1  second(X'FF8'),R1+X'8'  a,R8  pd2+X'7D2',R12  lab.first,R2  labdep.third(X'FF8'),R2+X'8'
D-Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                   HLASM R6.0  2008/07/11 17.48
                                    264 *********************************************************************** 00213000
                                    265 *  Symbol Attribute enhancements                                      * 00214000
                                    266 *********************************************************************** 00215000
0000CD C1C2C3                       268 SYMBOL1  DC    C'ABC'                                                   00217000
0000D0 12345C                       269 SYMBOL2  DC    P'123.45'                                                00218000
                                    270 &VAR1    SETC  'SYMBOL1'                                                00219000
                                    271 &VAR2    SETC  'SYMBOL2'                                                00220000
0000D3 00
0000D4 4110 80A6            000A6   272          LA    1,=C'ABC'                                                00221000
0000D8 4110 80A9            000A9   273          LA    1,=P'123.45'                                             00222000
                                 1  275 &TYPE    SETC  T'=C'ABC'                                                00224000
                                    276          DC    CL1'&TYPE'                                               00225000
0000DC C3                              +         DC    CL1'C'                                                   00225000
0000DD C3                           277          DC    AL1(T'SYMBOL1)                                           00226000
                                    278          DC    AL1(T'&VAR1)                                             00227000
0000DE C3                              +         DC    AL1(T'SYMBOL1)                                           00227000
0000DF C3                           279          DC    AL1(T'=C'ABC')                                           00228000
                                 2  280 &LEN     SETA  L'=C'ABC'                                                00229000
                                    281          DC    AL1(&LEN)                                                00230000
0000E0 03                              +         DC    AL1(3)                                                   00230000
0000E1 03                           282          DC    AL1(L'SYMBOL1)                                           00231000
                                    283          DC    AL1(L'&VAR1)                                             00232000
0000E2 03                              +         DC    AL1(L'SYMBOL1)                                           00232000
0000E3 03                           284          DC    AL1(L'=C'ABC')                                           00233000
                                 3  285 &INT     SETA  I'=P'123.45'                                             00234000
                                    286          DC    AL1(&INT)                                                00235000
0000E4 03                              +         DC    AL1(3)                                                   00235000
0000E5 03                           287          DC    AL1(I'SYMBOL2)                                           00236000
                                    288          DC    AL1(I'&VAR2)                                             00237000
0000E6 03                              +         DC    AL1(I'SYMBOL2)                                           00237000
0000E7 03                           289          DC    AL1(I'=P'123.45')                                        00238000
                                 4  290 &SCALE   SETA  S'=P'123.45'                                             00239000
                                    291          DC    AL1(&SCALE)                                              00240000
0000E8 02                              +         DC    AL1(2)                                                   00240000
0000E9 02                           292          DC    AL1(S'SYMBOL2)                                           00241000
                                    293          DC    AL1(S'&VAR2)                                             00242000
0000EA 02                              +         DC    AL1(S'SYMBOL2)                                           00242000
0000EB 02                           294          DC    AL1(S'=P'123.45')                                        00243000
                                    295          end                                                            00244000
000090 00000044FFFFFFE8          5  296                =a(a5,x)
000098 F1A2A34040404040             297                =cl8'1st'
0000A0 F2958440                     298                =cl4'2nd'
0000A4 F384                         299                =cl2'3d'
0000A6 C1C2C3                       300                =C'ABC'
0000A9 12345C                       301                =P'123.45'
```

1 The Type attribute (T') is allowed for ordinary symbols, SET 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, SET 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, SET 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 Scaling attribute (S') is allowed for ordinary symbols, SET 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 can 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 213), the literals are assembled there. You control placement of literal pools with the LTORG statement. X'FFFFFFE8' is used for the contents of A(X), statement 296. The symbol X was assigned the value (4*-6) by an EQU in statement 89.

```
BIGNAME                                          Relocation Dictionary                                          Page   13
   Pos.Id   Rel.Id   Address  Type  Action                                                  HLASM R6.0  2008/07/11 17.48
  00000001 00000001 00000090   A 4     +
  00000001 00000001 000000B1   A 3     +
```

```
BIGNAME                        Ordinary Symbol and Literal Cross Reference                                      Page   14
Symbol   Length   Value     Id    R Type Asm  Program   Defn References                     HLASM R6.0  2008/07/11 17.48
a             1 00000000 00000001     J                   27   57   211   213
area1         4 000007F4 00000002     F  F               235  225
area2         4 00000804 00000002     F  F               238  226
a5            2 00000044 00000001     I                   63   67   296
a8            2 000000B0 00000001     L                   67  145M
b5            8 000000B0 00000001     W                  195   67   145M  195
constant      4 000000AC 00000001     F  F                56   54   145M
deecees       1 000000AC 00000001     J                   55  194   212
first         1 00000000 FFFFFFFF     J                  241  227U  228U
first1        8 00000000 FFFFFFFF     C  C               242  231M  232M
first2        8 00000008 FFFFFFFF     C  C               243  229U  230
lab                      00000002 A   U                  228  230U  232
labdep                   00000002 A   U                  230  234
reallylongsymbol
              4 00000084 00000001     I                  145   42    47
second        1 00000000 FFFFFFFE     J                  244  229U
second1       4 00000000 FFFFFFFE     C  C               245  233M
SYMBOL1       3 000000CD FFFFFFFD     C  C               268  277   278   282   283
SYMBOL2       3 000000D0 FFFFFFFD     P  P               269  287   288   292   293
third         1 00000000 FFFFFFFD     J                  247  230U
third1        2 00000000 FFFFFFFD     C  C               248  234M
TRANSYLVANIA
              4 00000080 00000001     I                  144   42    47
X             1 FFFFFFE8 00000001 A   U                   89  296
=a(a5,x)      4 00000090 00000001     A                  296  144
=C'ABC'       3 000000A6 00000001     C                  300  272   279   284
=cl2'3d'      2 000000A4 00000001     C                  299  234
=cl4'2nd'
              4 000000A0 00000001     C                  298  233
=cl8'1st'
              8 00000098 00000001     C                  297  231   232
=P'123.45'
              3 000000A9 00000001     P                  301  273   289   294
```

```
BIGNAME                           Unreferenced Symbols Defined in CSECTs                                        Page   15
  Defn  Symbol                                                                              HLASM R6.0  2008/07/11 17.48
    65  a7
   143  SUSQUEHANNA
```

```
BIGNAME                                   Macro and Copy Code Source Summary                                    Page   16
 Con Source                                        Volume    Members                        HLASM R6.0  2008/07/11 17.48
     PRIMARY INPUT                                           DEMO     LOCATE   NOTE
  L2 OSMACRO  MACLIB   S2                          MNT190    IHBINNRA IHBRDWRS NOTE     WRITE    WTO
```

```
BIGNAME                                         Dsect Cross Reference                                           Page   17
Dsect     Length      Id       Defn                                                         HLASM R6.0  2008/07/11 17.48
first    00000010  FFFFFFFF     241
second   00000008  FFFFFFFE     244
third    000000EC  FFFFFFFD     247
```

```
BIGNAME                                               Using Map                                                Page   18
                                                                                            HLASM R6.0  2008/07/11 17.48
  Stmt  -----Location----- Action ----------------Using----------------- Reg Max     Last Label and Using Text
          Count      Id           Type          Value    Range     Id        Disp    Stmt
    28  00000000  00000001 USING  ORDINARY    00000000 00001000 00000001   8 000AC    273 *,8
   224  000007D2  00000002 USING  ORDINARY    000007D2 00001000 00000002  12 00032    226 *,12
   227  000007DA  00000002 USING  ORDINARY    00000000 00001000 FFFFFFFF   1 00008    233 first,1
   228  000007DA  00000002 USING  LABELED     00000000 00001000 FFFFFFFF   2 00008    232 lab.first,2
   229  000007DA  00000002 USING  DEPENDENT  +00000008 00000FF8 FFFFFFFE   1              second,first2
   230  000007DA  00000002 USING  LAB+DEPND  +00000008 00000FF8 FFFFFFFD   2              labdep.third,lab.first2
```

```
                                   General Purpose Register Cross Reference                                    Page   19
 Register  References (M=modified, B=branch, U=USING, D=DROP, N=index)                      HLASM R6.0  2008/07/11 17.48
    0(0)    (no references identified)
    1(1)      63M   87M  134M  135   136N  137N  189M  190   225M  227U  272M  273M
    2(2)      63   226M  228U
    3(3)     145N
    4(4)    (no references identified)
    5(5)      54M  142M
    6(6)     134   189   218M
    7(7)     145   218
    8(8)      28U  136
    9(9)     143M
   10(A)     142   143
   11(B)    (no references identified)
   12(C)     144M  224U
   13(D)      87   144M
   14(E)      30B  139M  191M
   15(F)      29M   29   137M  138M  138   139B  190M  191B
```

```
BIGNAME                           Diagnostic Cross Reference and Assembler Summary                              Page   20
                                                                                            HLASM R6.0  2008/07/11 17.48
     No Statements Flagged in this Assembly
HIGH LEVEL ASSEMBLER, 5696-234, RELEASE 5.0
SYSTEM: CMS 15                     JOBNAME: (NOJOB)     STEPNAME: (NOSTEP)   PROCSTEP: (NOPROC)
Data Sets Allocated for this Assembly
 Con DDname   Data Set Name                            Volume  Member
  P1 SYSIN    ASMASAMP ASSEMBLE A1                        DOGBOX
  L1 SYSLIB   BROOKES  MACLIB   A1                        DOGBOX
  L2          OSMACRO  MACLIB   S2                        MNT190
  L3          OSMACRO1 MACLIB   S2                        MNT190
  L4          DMSGPI   MACLIB   S2                        MNT190
     SYSLIN   ASMASAMP TEXT     A1                        DOGBOX
     SYSPRINT ASMASAMP LISTING  A1                        DOGBOX

  98138K allocated to Buffer Pool
    244 Primary Input Records Read         2217 Library Records Read
      0 ASMAOPT Records Read                485 Primary Print Records Written
     10 Object Records Written                0 ADATA Records Written
Assembly Start Time: 17.22.51 Stop Time: 17.22.51 Processor Time: 00.00.00.0407
Return Code 000
```

Appendix E. 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 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 79 on page 290 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, 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** in Figure 79 on page 290 shows an outer macro call, **1B** in Figure 79 on page 290 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, see example **2A** in Figure 79 on page 290. 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 file at some point before the MHELP 2 statement of interest.

MHELP 4, Macro AIF Dump

Items **4A**, **4B** , **1A**, **4D** , and **4E** in Figure 79 on page 290 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** in Figure 79 on page 290 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.

If there are *k* keyword parameters, they are listed as `KPARM0001` through `KPARM000`*k* in order of appearance on the prototype statement.

If there are *p* positional parameters, they are listed as `PPARM0001` through `PPARM000`*p* in order of appearance in the macro instruction.

Item **16A** in Figure 79 on page 290 has one keyword parameter (&OFFSET) and one positional parameter. 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 NAME and VALUE fields. A length of NUL indicates the corresponding item is empty.

© Copyright IBM Corp. 1992, 2013

Item **16B** in Figure 79 shows an inner call containing zero keywords and two positional parameters.

MHELP 64, Macro Hex Dump

This option, when used 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 described in the *HLASM 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 *HLASM Language Reference* for further information.

```
                                                                                                        PAGE    3
  Active Usings: None
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

000000               00000 00000     1          csect                                                   00246000
                                     2 *        copy  lnsrch                                            00247000
                                     3          macro                                                   00248000
                                     4 &name   lnsrch &arg,&offset=stnumb-stchain                       00249000
                                     5          lclc  &label                                            00250000
                                     6 &label   setc  'A&sysndx'          Generate symbol               00251000
                                     7          aif   (t'&name eq 'O').skip                             00252000
                                     8 &label   setc  '&name'             If MACRO call has label, use it 00253000
                                     9 .skip    anop                      instead of generated symbol   00254000
                                    10 &label   la    0,&offset           Load reg. 0                   00255000
                                    11          schi  &arg,0(1)           Search                        00256000
                                    12          bc    1,&label            If max reached, continue      00257000
                                    13          mend                                                    00258000
```

Figure 79. Sample program using MHELP (part 1 of 8)

```
                                                                                                             Page    4
Active Usings: None
Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

                                   15 *        copy  schi                                                     00260000
                                   16          macro                                                          00261000
                                   17 &nm      schi  &comp,&list                                              00262000
                                   18          lcla  &cnt                                                     00263000
                                   19          lclc  &cmpadr                                                  00264000
                                   20 &cnt     seta  1                                                        00265000
                                   21 &nm      stm   1,15,4(13)                                               00266000
                                   22 .test    anop                                                           00267000
                                   23 &cmpadr  setc  '&cmpadr'.'&comp'(&cnt,1)                                00268000
                                   24          aif   ('&comp'(&cnt,1) eq '(').lpar                            00269000
                                   25 &cnt     seta  &cnt+1                                                   00270000
                                   26          aif   (&cnt lt k'&comp).test                                   00271000
                                   27 .nolnth  anop                                                           00272000
                                   28          la    3,&comp             Comparand                            00273000
                                   29          ago   .contin                                                  00274000
                                   30 .lpar    aif   ('&comp'(&cnt+1,1) eq ',').finish                        00275000
                                   31 &cnt     seta  &cnt+1                                                   00276000
                                   32          aif   (&cnt lt k'&comp).lpar                                   00277000
                                   33          ago   .nolnth                                                  00278000
                                   34 .finish  anop                                                           00279000
                                   35 &cmpadr  setc  '&cmpadr'.'&comp'(&cnt+2,k'&comp-&cnt)                   00280000
                                   36          la    3,&cmpadr           Comparand sans length                00281000
                                   37 .contin  anop                                                           00282000
                                   38          la    1,&list             List header                          00283000
                                   39          mvc   &comp,0(0)          Dummy move to get comp length        00284000
                                   40          org   *-6                 Change MVC to MVI                    00285000
                                   41          dc    x'92'               MVI Opcode                           00286000
                                   42          org   *+1                 Preserve length as immed opnd        00287000
                                   43          dc    x'd000'             Result is MVI 0(13),L                00288000
                                   44          l     15,=v(schi)                                              00289000
                                   45          balr  14,15                                                    00290000
                                   46          lm   1,15,4(13)                                                00291000
                                   47          mexit                                                          00292000
                                   48          mend                                                           00293000
```

Figure 80. Sample program using MHELP (part 2 of 8)

```
  Active Usings: None
                                                                                                               Page    5
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

000000               00000 00048    50 test     csect                                                           00295000
000000 05C0                         51          balr  12,0                                                      00296000
                 R    00002         52          using *,12                                                      00297000

                                    54          mhelp b'11111'                                                  00299000
                                    55          lnsrch listline,offset=listline-listnext                        00300000

                1A         ++//MHELP CALL TO MACRO LNSRCH      DEPTH=001   SYSNDX=0000001   STMT=00055

                16A        //MHELP ENTRY TO  LNSRCH   MODEL STMT=00000 DEPTH=001 SYSNDX=0000001 KWCNT=001
                                        ////SYSTEM PARAMETERS:
                                        //SYSVAR NAME    LNTH  VALUE (56 CHARS/LINE)
                                        //SYSNDX          004  0001
                                        //SYSECT          004  test
                                        //SYSLOC          004  test
                                        //SYSTIME         005  17.48
                                        //SYSDATE         008  07/11/08
                                        //SYSASM          020  HIGH LEVEL ASSEMBLER
                                        //SYSVER          005  1.6.0
                                        //SYSDATC         008  20080711
                                        //SYSJOB          007  (NOJOB)
                                        //SYSSTEP         008  (NOSTEP)
                                        //SYSSTYP         005  CSECT
                                        //SYSSTMT         008  00000056
                                        //SYSCLOCK        026  2008-07-11 17:48:42.914829
                                        //SYSNEST         001  1
                                        //SYSSEQF         008  00300000
                                        //SYSOPT_DBCS     001  0
                                        //SYSOPT_OPTABLE  003  UNI
                                        //SYSOPT_RENT     001  0
                                        //SYSOPT_XOBJECT  001  0
                                        //SYSTEM_ID       006  CMS 13
                                        //SYSIN_DSN       020  ASMASAMP ASSEMBLE A1
                                        //SYSIN_MEMBER    NUL
                                        //SYSIN_VOLUME    005  ADISK
                                        //SYSLIB_DSN      020  ASMASAMP ASSEMBLE A1
                                        //SYSLIB_MEMBER   NUL
                                        //SYSLIB_VOLUME   005  ADISK
                                        //SYSPRINT_DSN    020  ASMASAMP LISTING  A1
                                        //SYSPRINT_MEMBER NUL
                                        //SYSPRINT_VOLUME 005  ADISK
                                        //SYSTERM_DSN     NUL
                                        //SYSTERM_MEMBER  NUL
                                        //SYSTERM_VOLUME  NUL
                                        //SYSPUNCH_DSN    NUL
                                        //SYSPUNCH_MEMBER NUL
                                        //SYSPUNCH_VOLUME NUL
                                        //SYSLIN_DSN      020  ASMASAMP TEXT     A1
                                        //SYSLIN_MEMBER   NUL
                                        //SYSLIN_VOLUME   005  ADISK
                                        //SYSADATA_DSN    NUL
                                        //SYSADATA_MEMBER NUL
```

Figure 81. Sample program using MHELP (part 3 of 8)

```
                                                    ++//MHELP  BRANCH FROM STMT 00007 TO STMT 00009 IN MACRO LNSRCH
  Active Usings: test+X'2',R12
                                                                                                               Page    6

  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

                                        //SYSADATA_VOLUME NUL
                                        //SYSPARM         NUL
                                        //SYSM_SEV        003  000
                                        //SYSM_HSEV       003  000
                                        ////NAME; KEYWORD PARAMETERS; POSITIONAL PARAMETERS:
                                        //PARAMETER      LNTH  VALUE (54 CHARS/LINE)
                                        //NAME            NUL
                                        //KPARM0001       017  listline-listnext
                                        //PPARM0001       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

                2A         ++//MHELP  BRANCH FROM STMT 00007 TO STMT 00009 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

                16B        //MHELP ENTRY TO    SCHI   MODEL STMT=00000 DEPTH=002 SYSNDX=0000002 KWCNT=000
                                        ////SYSTEM PARAMETERS:
                                        //SYSVAR NAME    LNTH  VALUE (56 CHARS/LINE)
                                        //SYSNDX          004  0002
                                        //SYSECT          004  test
                                        //SYSLOC          004  test
                                        //SYSTIME         005  06.48
                                        //SYSDATE         008  02/02/00
                                        //SYSASM          020  HIGH LEVEL ASSEMBLER
                                        //SYSVER          005  1.4.0
                                        //SYSDATC         008  20000202
                                        //SYSJOB          007  (NOJOB)
                                        //SYSSTEP         008  (NOSTEP)
                                        //SYSSTYP         005  CSECT
                                        //SYSSTMT         008  00000058
                                        //SYSCLOCK        026  2000-02-02 06:48:42.915979
                                        //SYSNEST         001  2
                                        //SYSSEQF         008  00300000
                                        //SYSOPT_DBCS     001  0
                                        //SYSOPT_OPTABLE  003  UNI
                                        //SYSOPT_RENT     001  0
                                        //SYSOPT_XOBJECT  001  0
                                        //SYSTEM_ID       006  CMS 13
                                        //SYSIN_DSN       020  ASMASAMP ASSEMBLE A1
                                        //SYSIN_MEMBER    NUL
                                        //SYSIN_VOLUME    005  ADISK
                                        //SYSLIB_DSN      020  ASMASAMP ASSEMBLE A1
                                        //SYSLIB_MEMBER   NUL
                                        //SYSLIB_VOLUME   005  ADISK
                                        //SYSPRINT_DSN    020  ASMASAMP LISTING  A1
                                        //SYSPRINT_MEMBER NUL
```

Figure 82. Sample program using MHELP (part 4 of 8)

```
  Active Usings: test+X'2',R12
                                                                                                               Page    7

  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

                                        //SYSPRINT_VOLUME 005  ADISK
                                        //SYSTERM_DSN     NUL
                                        //SYSTERM_MEMBER  NUL
                                        //SYSTERM_VOLUME  NUL
                                        //SYSPUNCH_DSN    NUL
                                        //SYSPUNCH_MEMBER NUL
                                        //SYSPUNCH_VOLUME NUL
                                        //SYSLIN_DSN      020  ASMASAMP TEXT     A1
                                        //SYSLIN_MEMBER   NUL
                                        //SYSLIN_VOLUME   005  ADISK
                                        //SYSADATA_DSN    NUL
                                        //SYSADATA_MEMBER NUL
                                        //SYSADATA_VOLUME NUL
                                        //SYSPARM         NUL
                                        //SYSM_SEV        003  000
                                        //SYSM_HSEV       003  000
                                        ////NAME; KEYWORD PARAMETERS; POSITIONAL PARAMETERS:
                                        //PARAMETER      LNTH  VALUE (54 CHARS/LINE)
                                        //NAME            NUL
                                        //PPARM0001       008  listline
                                        //PPARM0002       004  0(1)

000006 901F D004            00004    58+         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
                                        //     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


                                        ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00022 IN MACRO SCHI


                4E         //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
```

Figure 83. Sample program using MHELP (part 5 of 8)

```
                                                                                                             Page    8
Active Usings: test+X'2',R12
Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48
                                      ++//MHELP  BRANCH FROM STMT 00026 TO STMT 00022 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 00022 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
                                      ////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 00022 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 00022 IN MACRO SCHI


                                      //MHELP AIF IN    SCHI     MODEL STMT=00024 DEPTH=002 SYSNDX=0000002 KWCNT=000
```

Figure 84. Sample program using MHELP (part 6 of 8)

```
                                                                                                               Page    9
  Active Usings: test+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

                                        ////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 00022 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

00000A 4130 C024            00026    59+         la    3,listline          Comparand                       02-00028

                                        ++//MHELP  BRANCH FROM STMT 00029 TO STMT 00037 IN MACRO SCHI

00000E 4111 0000            00000    60+         la    1,0(1)              List header                     02-00038
000012 D202 C024 0000 00026 00000    61+         mvc   listline,0(0)       Dummy move to get comp length   02-00039
000018                00018 00012    62+         org   *-6                 Change MVC to MVI               02-00040
000012 92                            63+         dc    x'92'               MVI Opcode                      02-00041
000013                00013 00014    64+         org   *+1                 Preserve length as immed opnd   02-00042
000014 D000                          65+         dc    x'd000'             Result is MVI 0(13),L           02-00043
000016 58F0 C02E            00030    66+         l     15,=v(schi)                                         02-00044
00001A 05EF                          67+         balr  14,15                                               02-00045
00001C 981F D004            00004    68+         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    69+         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
⋮
                                                                                                               Page   10
  Active Usings: test+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                  HLASM R6.0  2008/07/11 17.48

000024                               70 listnext ds    h                                                       00301000
000026                               71 listline ds    fl3'0'                                                  00302000
000030                               72          ltorg                                                         00303000
000030 00000000                      73                =v(schi)
⋮
```

Figure 85. Sample program using MHELP (part 7 of 8)

```
         New Floating-point Constants included for Verification                                                Page   11
  Active Usings: test+X'2',R12
  Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                   HLASM R6.0  2008/07/11 17.48

                                     75 **********************************************************************  00305000
                                     76 *  New FP definitions included to verify correct release of HLASM    *  00306000
                                     77 **********************************************************************  00307000
000034 00000000
000038 4093480000000000              78 BFP1     DC    DB'1234'                                                 00308000
000040 434D200000000000              79 HFP1     DC    DH'1234'                                                 00309000
000000                               80          end   test                                                     00310000
```

Figure 86. Sample program using MHELP (part 8 of 8)

Appendix F. 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 (CMS).

The following section describes the format and placement of messages issued by the assembler. “Assembly error diagnostic messages” on page 299, “Abnormal assembly termination messages” on page 341, and “ASMAHL Command Error Messages (CMS)” on page 346, list and describe each message.

Note: If you use the FLAG(*severity code*) assembler option, only error diagnostic messages with a severity of *severity code* or higher appear on the listing.

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 want 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.

© Copyright IBM Corp. 1992, 2013

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 does 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 assembles as expected or that it runs.

C—Critical
(Severity code = 16)

The condition is a critical error. It is not likely that the program runs 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:

```
ASMACMSnnne
```

where:

nnn	Is a three-character message number
e	Indicates the severity of an error. See 299.

LANGUAGE Assembler Option: Unless otherwise indicated, 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 which are not included in this section (for example, `ASMA222`).

Explanation of Message

For some messages there is more than one explanation, as 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 normally 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 processed correctly.
- 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 probably causes 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 to list (from keying errors to wrong use of the statement). 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 297.

ASMAHL messages (those messages starting with ASMACMS) have a severity code letter of E. The associated return code is shown in the System Action for each message. An ASMAHL message that causes the assembly to terminate generates a return code greater 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 z/OS) 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 a severity code of `S` prevent the assembled program from linkage editing or running. Errors with a severity code of `E`, or lower, 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 are then 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 message is:

```
** ASMA057E Undefined operation code - xxxxx
```

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, system variable, macro parameter, or value string associated with the error.

Messages reference three types of macro parameter: the *name field* parameter, *keyword* parameters, and *positional* parameters. A reference to the name field parameter is indicated by the word "NAME" appended to the message. References to keyword and positional parameters (for which there can be multiple occurrences) are in the form "KPARM*nnnn*" and "PPARM*nnnn*", where *nnnn* is the relative number of the parameter within the macro definition.

Figure 87 shows an example of a macro with messages referencing each type of variable or parameter.

```
Loc  Object Code    Addr1 Addr2  Stmt   Source Statement                                   HLASM R6.0  2008/07/11 17.48
                                    1     MACRO                                                                00001000
                                       3          4                           4                  5
                                    2 &z  parms &kw1=a,&kw2=b,&kw3=c,&kw4=d,&kw5=e,&kw6=f,&pp1,&pp2            00002000
                                    3 &c  SETC   'just a string'      1                                        00003000
                                    4 &ss SETA   &c                                                            00004000
                                    5 &sv SETA   &sysasm   2                                                   00005000
                                    6 &z1 SETA   &z   3                                                        00006000
                                    7 &k1 SETA   &kw1      4                                                   00007000
                                    8 &k5 SETA   &kw5                                                          00008000
                                    9 &n  SETA   n'&syslist                                                    00009000
                                   10 &pn     SETA   &syslist(&n)       5                                      00010000
                                   11 &p2     SETA   &pp2                        5                             00011000
                                   12     MEND                                                                 00012000
 000000             00000 00000    13 default CSECT                                                            00013000
                                   14 n   parms pp1,pp2,kw5=z,pp3,kw1=y,pp4,pp5,pp6                            00014000
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00004/C   1
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00005/SYSASM   2
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00006/NAME   3
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00007/KPARM00001      4
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00008/KPARM00005
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00010/PPARM00006      5
 ASMA102E Arithmetic term is not self-defining term; default=0 - 00011/PPARM00002
                                   15     END                                                                  00015000
```

Figure 87. Sample macro parameter messages

Notes to Figure 87:

1 SET symbol, and related message

2 System variable symbol, and related message

3 The name field parameter, and related message

4 Keyword parameters, and related messages

5 Positional parameters, and related messages

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.

Several messages might be issued for a single statement or even for a single error within a statement. This happens because each statement is normally 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 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:

```
** ASMA000E 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 assembles correctly. However, this message indicates an error in the error detection process of the assembler. Save the output and the source files from this assembly and report the problem to your IBM service representative.

Messages

ASMA001E 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
ISEQ     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

ASMA002S Generated statement too long; statement truncated - *xxxxxxxx*

Explanation: The statement generated by a macro definition is more than 3072 characters long.

System action: The statement is truncated; the leading 1728 characters are retained.

Programmer response: Shorten the statement.

Severity: 12

ASMA003E Undeclared variable symbol; default=0, null, or type=U - *xxxxxxxx*

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
```

If the assembler is unable to determine an appropriate type from the context of the symbol's use, a default type of SETC is assigned.

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

ASMA004E 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

ASMA005S 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

ASMA007S Previously defined sequence symbol -

- 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

ASMA011E Inconsistent global declarations; first is retained - *xxxxxxxx*

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 declaration compatible.

Severity: 8

ASMA012S 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

ASMA013S ACTR counter exceeded - *xxxxxxxx*

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 143 and Appendix E, “MHELP sample macro trace and dump,” on page 289, to trace macro definition logic.) If there is no error, increase the initial count on the ACTR instruction.

Severity: 12

ASMA014E 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

ASMA015W Literal bounds exceeded

Explanation: The expression containing the reference to the literal:
- resolves to an address outside the bounds of the literal
- the length of the receiving field is longer than the length of the literal

This indicates a potential error.

System action: The instruction assembles as specified.

Programmer response: Change the expression to not exceed the bounds.

Severity: 4

ASMA016W Literal used as the target of *xxxxx* **instruction**

Explanation: The target of instruction *xxxxx* is a literal. This indicates a potential error. *xxxxx* is output in uppercase.

System action: The instruction assembles as specified.

Programmer response: Specify the instruction target correctly. You can suppress this warning for an EX instruction by specifying the NOEXLITW suboption of the FLAG option.

Severity: 4

ASMA017W Undefined keyword parameter; default to positional, including keyword - *xxxxxxxx*

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 apostrophes, and adjust the macro coding appropriately.

Severity: 4

ASMA018S Duplicate keyword in macro call; last value is used - *xxxxxxxx*

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

ASMA019W Length of EQUated symbol *xxxxxxxx* **undefined; default=1**

Explanation: The value of the length attribute extracted for an EQUated symbol with an unspecified length has been set to the default: 1.

System action: The instruction assembles as specified.

Programmer response: Ensure that the length attribute of the symbol is defined.

Severity: 4

ASMA020E Illegal GBL or LCL statement - *xxxxxxxx*

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

ASMA021E Illegal SET statement - *xxxxxxxx*

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

ASMA022I START value rounded up to required boundary

Explanation: The value specified in the operand field of the START instruction has been rounded up to the required boundary.

System action: The assembly continues.

Programmer response: To stop the message occurring, specify the required boundary for the value.

Severity: 0

ASMA023E Symbolic parameter too long - *xxxxxxxx*

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 that all symbolic parameters consist of an ampersand followed by 1 to 62 alphanumeric characters, the first of which is alphabetic.

Severity: 8

ASMA024E Invalid variable symbol - *xxxxxxxx*

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

ASMA025S Invalid macro prototype operand - *xxxxxxxx*

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

ASMA026S Macro call operand too long; leading characters deleted - *xxxxxxxx*

Explanation: An operand of a macro instruction is more than the allowable length.

System action: The leading characters (for the allowable length) are deleted.

Programmer response: Split the operand into two or more operands.

Severity: 12

ASMA027S Excessive number of operands

Explanation: One of the following situations 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

ASMA028E Invalid displacement

Explanation: One of the following situations 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.
- The displacement of a long-displacement instruction cannot be resolved to lie within the range from -524288 to +524287.

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

ASMA029E Incorrect register specification - *xxxxxxxx*

Explanation: The value *xxxxxxxx* is invalid for one of the following reasons:

- *xxxxxxxx* 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 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

ASMA030E Invalid literal usage - *xxxxxxxx*

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

ASMA031E Invalid immediate or mask field

Explanation: The value of an immediate or mask operand of a machine instruction requires more bits to represent it than allowed by the instruction, or the value of the immediate operand exceeds 9 on an SRP instruction or 15 on an MC instruction.

Immediate fields used in an arithmetic context are allowed to be signed, those in a logical context are not; for example:

```
AHI  r1,-30000   is valid, but
AHI  r1,50000    is not

TMH  r1,50000    is valid, but
TMH  r1,-30000   is not
```

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 non-immediate type.

Severity: 8

ASMA032E Relocatable value or unresolved symbol found when absolute value required

Explanation: One of the following situations 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.
- The expression is not resolvable due to a dependency on another symbol that is not yet resolved.

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, or remove dependency on unresolved symbol.

Severity: 8

ASMA033I Storage alignment for *xxxxxxxx* **unfavorable**

Explanation: An address referenced by this statement might not be aligned to the optimal boundary for this instruction; for example, the data referenced by a load instruction (L) might be on a halfword boundary.

System action: The instruction assembles as written.

Programmer response: Correct the operand if it is in error. If you are using an instruction that does not require alignment, or you want to suppress alignment checking for some other reason, you can specify the NOALIGN assembler option or ACONTROL FLAG(NOALIGN). 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: 0

ASMA034E Operand *operand* **beyond active USING range by** *xxxx* **bytes**

Explanation: The address of this statement does not fall within the range of an active USING statement.

System action: The instruction assembles as zero.

Programmer response: Increase the range of the active USING.

Severity: 8

ASMA035S 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:
   - An apostrophe with an address constant.
   - A left parenthesis with a non-address constant.
   - A constant field not started with an apostrophe, left parenthesis, space, 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 space.
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

ASMA036W 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 checking for a control section defined by an RSECT instruction.

Severity: 4

ASMA037E 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

ASMA038S 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

ASMA039S 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 GOFF assembler option is specified the maximum location counter value is X'7FFFFFFF'.

System action: The assembly continues, however, the resulting code probably does 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

ASMA040S 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

ASMA041E Term expected; text is unclassifiable - *xxxxxxxx*

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 an apostrophe did not introduce a valid term; the letter is not L, C, G (DBCS option), 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. 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

ASMA042W Length attribute of symbol is unavailable; default=1 - *xxxxxxxx*

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 symbol 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 that 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: 4

ASMA043E 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

ASMA044E 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

ASMA045W 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

ASMA046E 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

ASMA047E 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 (first

Syogrammer response:

ASMA052S Record out of sequence - *xxxxxxxx*

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

ASMA053W Blank sequence field - *xxxxxxxx*

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 non-blank sequence field.

Programmer response: Put the correct sequence number in the statement or discontinue sequence checking over the blank statements with an ISEQ statement that has a blank operand.

Severity: 4

ASMA054E 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

ASMA055S 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

ASMA056W Absolute value found when relocatable value expected - *xxxxxxxx*

Explanation: An absolute expression has been used as the immediate field in a branch-relative instruction. The immediate field in a branch-relative instruction is used as signed number of halfwords relative to the current location counter. The use of an absolute expression for this value can cause unpredictable results.

System action: The instruction assembles as written.

Programmer response: Supply a relocatable expression.

Severity: 4

ASMA057E 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.
- On z/VSE the High Level Assembler only reads library macros that have a member type of `A`, or if the // OPTION SUBLIB=DF statement is used, a member type of `D`. Edited (E-Deck) macros, that have a member type of `E` or `F` can only be read by a LIBRARY exit.

System action: The statement is ignored. 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.

If the operation code shown is a z/VSE edited macro (E-Deck), High Level Assembler can only find and read it with a LIBRARY exit. You might want to use the z/VSE supplied LIBRARY exit described in VSE/ESA Guide to System Functions.

Severity: 8

ASMA058E Invalid relative address - *xxxxxxxx*

Explanation: One of the following situations has occurred:
- The relative address is an odd value, and therefore cannot be represented as a number of halfwords.

- The NOGOFF option has been specified and the relative address is not in the same control section as the instruction.
- The relative address has the same relocatability attribute as the instruction, but is outside the control section.

System action: The instruction assembles as zero.

Programmer response: Supply a valid relative address that is on a halfword boundary and within the same control section. To allow a relative address that is outside the current control section, specify the GOFF option.

Severity: 8

ASMA059C Illegal ICTL - *xxxxxxxx*

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

ASMA060S COPY code not found - *xxxxxxxx*

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

ASMA061E Symbol not name of DSECT, DXD, or external label

Explanation: The operand of a Q-type address constant is not a symbol or the name of a DSECT or DXD statement, or an external label.

System action: The constant assembles as zero.

Programmer response: Supply a valid operand.

Severity: 8

ASMA062E Illegal operand format - *xxxxxxxx*

Explanation: One of the following errors has occurred:

- 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
- ACONTROL—one or more of the operands supplied is invalid
- AINSERT—the first operand is not a valid string, or the second operand is not BACK or FRONT
- AMODE—the operand does not specify 24, 31, or ANY
- AREAD—the operand specifies an incorrect AREAD operand.
- 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

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

ASMA063E No ending apostrophe - *xxxxxxxx*

Explanation: The apostrophe 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 apostrophe.

Severity: 8

ASMA064S 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 approximately $5.4\text{x}10^{-79}$ to $7.2\text{x}10^{75}$.

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

ASMA065E Unknown type - *xxxxxxxx*

Explanation: An unknown constant type has been used in a DC or DS statement or in a literal, or the assembler option required to support the constant type has not been supplied.

System action: The operand in error and the following operands are ignored.

Programmer response: Supply a valid constant or the required assembler option. Look for an incorrect type code or incorrect syntax in the duplication factor.

Severity: 8

ASMA066W 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 64 KB (65,536 bytes).

System action: The statement assembles as written.

Programmer response: If the address constant is used to refer to a storage address less than 64 KB (65,536 bytes), the 2-byte relocatable address constant is valid. You can use the assembler option RA2 to suppress this message.

Severity: 4

ASMA067S Illegal duplication factor - *xxxxxxxx*

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

ASMA068S Length error - *xxxxxxxx*

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 is out of the permissible range
- The length modifier of a Character Unicode constant is not a multiple of 2

System action:
- A machine instruction assembles as zero
- A new floating point constant assembles as zero
- An address constant is truncated
- For other DC, DS, or DXD statements, the operand in error and the following operands are ignored
- The operand in error, and the operands following, are ignored.

Programmer response: Supply a valid length or correct the length modifier.

Severity: 12

ASMA069S Length of second operand must be less than length of first

Explanation: The length of the second operand must be less than the length of the first operand. If it is not, a specification exception is recognized.

System action: The machine instruction assembles as zero.

Programmer response: Supply a second operand with a length which is less than that of the first operand.

Severity: 12

ASMA070E Scale modifier error - *xxxxxxxx*

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

ASMA071E 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

ASMA072E 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 an 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

ASMA073E Precision lost

Explanation: The modifiers of a floating-point number either truncate the exponent or shift the fraction out of the converted constant.

System action: The constant assembles with an exponent but with a fraction of zero.

Programmer response: Change the modifier or use a longer constant type.

Severity: 8

ASMA074E Illegal syntax in expression - *xxxxxxxx*

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

ASMA075E 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; a conditional assembly expression uses zero as the result.

Programmer response: Change the expression.

Severity: 8

ASMA076E 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

ASMA077E Circular definition

Explanation: The value of a symbol in an expression is dependent on itself, either directly or indirectly, through 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

ASMA078E Operand ***op*** **expression complexly relocatable -** *expr*

Explanation: The expression specified is complexly relocatable, but an absolute or simply relocatable expression is required.

System action: The instruction assembles as zero.

Programmer response: Correct the expression.

Severity: 8

ASMA079E 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

ASMA080E 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

ASMA081E Created SET symbol exceeds 63 characters - *xxxxxxxx*

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

ASMA082E Created SET symbol is null - *xxxxxxxx*

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

ASMA083E Created SET symbol is not a valid symbol - *xxxxxxxx*

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

ASMA084S Generated name field exceeds 63 characters; discarded - *xxxxxxxx*

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

ASMA085I Generated operand field is null - *xxxxxxxx*

Explanation: The operand field of a generated machine instruction is null (empty). This message is not issued for generated macro instructions.

System action: The statement assembles as though no operand were specified.

Programmer response: Provide a non-empty operand field. If you want the statement assembled with no operand, substitute a comma rather than leave the operand blank.

Severity: 0

ASMA086S Missing MEND generated - *xxxxxxxx*

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 omitted. Otherwise, check if all the macro definition is on the library.

Severity: 12

ASMA087S Generated operation code is null - *xxxxxxxx*

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

ASMA088E Unbalanced parentheses in macro call operand - *xxxxxxxx*

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

ASMA089E Arithmetic expression contains illegal delimiter or ends prematurely - *xxxxxxxx*

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

ASMA090E Excess right parenthesis in macro call operand - *xxxxxxxx*

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

ASMA091E Character string exceeds maximum length; truncated to maximum - *xxxxxxxx*

Explanation: The value of the operand of a SETC or SETCF statement or the character relational operand of an AIF statement is longer than 1024 characters. This might occur before substrings are evaluated.

System action: The first 1024 characters are used.

Programmer response: Shorten the SETC or SETCF expression value or the operand value.

Severity: 8

ASMA092E Substring expression 1 points past string end; default=null - *xxxxxxxx*

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

ASMA093E Substring expression 1 less than 1; default=null - *xxxxxxxx*

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

ASMA094I 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

ASMA095W Substring expression 2 less than 0; default=null - *xxxxxxxx*

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

ASMA096E Unsubscripted SYSLIST; default=SYSLIST(1) - *xxxxxxxx*

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

ASMA097E Invalid attribute reference to SETA or SETB symbol; default=U or 0 - *xxxxxxxx*

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

ASMA098E Attribute reference to invalid symbol; default=U or 0 - *xxxxxxxx*

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

ASMA099W Wrong type of constant for S or I attribute reference; default=0 - *xxxxxxxx*

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

ASMA100E Subscript less than 1; default to subscript=1 - *xxxxxxxx*

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

ASMA101E Subscript less than 1; default to value=0 or null - *xxxxxxxx*

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

ASMA102E Arithmetic term is not self-defining term; default=0 - *xxxxxxxx*

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 apostrophes must be part of the SETC value.

Severity: 8

ASMA103E Multiplication overflow; default product=1 - *xxxxxxxx*

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

ASMA104W Statement processing incomplete

Explanation: This indicates that a previously flagged error has terminated processing for this statement.

System action: The assembly continues.

Programmer response: Correct previous errors.

Severity: 4

ASMA105U Arithmetic expression too complex

Explanation: An arithmetic expression in a macro definition statement caused an internal workarea 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

ASMA106E Wrong target symbol type; value left unchanged - *xxxxxxxx*

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

ASMA107E Inconsistent dimension on target symbol; subscript ignored, or 1 used - *xxxxxxxx*

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. You can declare a local SET symbol as dimensioned by using it, subscripted, in the name field of a SET statement.

Severity: 8

ASMA108E Inconsistent dimension on SET symbol reference; default = 0, null, or type=U - *xxxxxxxx*

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

ASMA109E Multiple SET operands for undimensioned SET symbol; gets last operand - *xxxxxxxx*

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

ASMA110S 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

ASMA111S Invalid AIF or SETB operand field - *xxxxxxxx*

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

ASMA112S Invalid sequence symbol - *xxxxxxxx*

Explanation: One of the following errors has occurred:
- A sequence symbol does not 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

ASMA113S 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 or records are processed normally.

Programmer response: Begin this record in the continuation indicator field.

Severity: 12

ASMA114S Invalid COPY operand - *xxxxxxxx*

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

ASMA115S COPY operand too long - *xxxxxxxx*

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

ASMA116E 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

ASMA117E Illegal subscript - *xxxxxxxx*

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

ASMA118S 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 honored and the macro definition stops. No more COPY code is read. The statements brought in before the end of the COPY code are processed.

Programmer response: Make sure that each library member to be used as COPY code contains balanced MACRO and MEND statements.

Severity: 12

ASMA119S 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

ASMA120S 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

ASMA121S 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.

on z/OS or CMS, 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.

On z/VSE, this can be done by decreasing the value you specify on the SIZE parameter of the JCL EXEC statement, or by running the assembly in a larger partition.

Severity: 12

ASMA122S Illegal operation code format

Explanation: The operation code is not followed by a space 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

ASMA123S Variable symbol too long - *xxxxxxxx*

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

ASMA124S Illegal use of symbolic parameter - *name*

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

ASMA125S Illegal macro name - macro uncallable - *xxxxxxxx*

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

ASMA126S 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 normally 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

ASMA127S 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

ASMA128S Excess right parenthesis - *xxxxxxxx*

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

ASMA129S Insufficient right parentheses - *xxxxxxxx*

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

ASMA130S Illegal attribute reference - *xxxxxxxx*

Explanation: One of the following errors has occurred:
- The symbol following an 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 or O attribute reference is not a valid variable symbol or ordinary symbol
- The apostrophe is missing from a T attribute reference

System action: The statement is ignored.

Programmer response: Supply a valid attribute reference.

Severity: 12

ASMA131S Parenthesis nesting depth exceeds 255 - *xxxxxxxx*

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

ASMA132S Invalid logical expression - *xxxxxxxx*

Explanation: A logical 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 logical expression.

Severity: 12

ASMA133S Illegal substring reference - *xxxxxxxx*

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

ASMA134S Invalid relational operator - *xxxxxxxx*

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

ASMA135S Invalid logical operator - *xxxxxxxx*

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

ASMA136S 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

ASMA137S Invalid character expression - *xxxxxxxx*

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, a valid operation code attribute (O') or a valid character expression enclosed in apostrophes.

System action: The statement is ignored.

Programmer response: Supply a valid character expression.

Severity: 12

ASMA138W Non-empty PUSH *xxxxxxx* **stack**

Explanation: The number of PUSH instructions exceeds the number of POP instructions at the end of the assembly. This indicates a potential error.

System action: The assembly continues.

Programmer response: Change your program to issue POP instructions for all PUSHes. You can suppress this warning by specifying the NOPUSH suboption of the FLAG option.

Severity: 4

ASMA139S 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

ASMA140W END record missing

Explanation: End-of-file on the source input data set occurred before an END statement was read. One of the following situations 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

ASMA141E Bad character in operation code - *xxxxxxxx*

Explanation: The operation code contains a non-alphanumeric character, that is, a character other than A to Z, 0 to 9, $, #, @, or _. Embedded spaces 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

ASMA142E Operation code not complete on first record

Explanation: The whole name and operation code, including a trailing space, is not contained on the first record (before the continue column—normally 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

ASMA143E Bad character in name field - *xxxxxxxx*

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

ASMA144E Begin-to-continue columns not blank - *xxxxxxxx*

Explanation: On a continuation record, one or more columns between the begin column (normally column 1) and the continue column (normally 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

ASMA145E Operator, right parenthesis, or end-of-expression expected - *xxxxxxxx*

Explanation: One of the following situations has occurred:

- A letter, number, equal sign, apostrophe, or undefined character occurred following a term where a right parenthesis, an operator, a comma, or a space 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

ASMA146E Self-defining term too long or value too large - *xxxxxxxx*

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

ASMA147E Symbol too long, or first character not a letter - *xxxxxxxx*

Explanation: A symbol is longer than 63 characters, or does not begin with a letter, $, #, @, or underscore (_).

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

ASMA148E Self-defining term lacks ending quote or has bad character - *xxxxxxxx*

Explanation: A hexadecimal or binary self-defining term contains a character that is not permitted or is missing the final apostrophe, 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

ASMA149E Literal length exceeds 256 characters, including = sign - *xxxxxxxx*

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

ASMA150E Symbol has non-alphanumeric character or invalid delimiter - *xxxxxxxx*

Explanation: The first character following a symbol is not a valid delimiter (plus sign, minus sign, asterisk, slash, left or right parenthesis, comma, or space).

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

ASMA151E Literal expression modifiers must be absolute and predefined - *xxxxxxxx*

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

ASMA152S External symbol too long or unacceptable character - *xxxxxxxx*

Explanation: One of the following errors has occurred:

- An external symbol is longer than eight characters, or the limit is 63 characters when the GOFF/XOBJECT option is in effect, or contains a bad character. An external symbol might be the name of a CSECT, RSECT, 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.
- A class name in a CATTR statement is longer than 16 characters, 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

ASMA153S 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

ASMA154E Operand must be absolute, predefined symbols; set to zero - *xxxxxxxx*

Explanation: The operand on a SETA, SETB, SETC, 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, DSECT, or RSECT, are not valid where an absolute, predefined value is required.

Severity: 8

ASMA155S Previous use of symbol is not this section type

Explanation: The name on a CSECT, DSECT, RSECT, COM, CATTR, 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

ASMA156S 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

ASMA157S 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, or is not an external symbol without addend.

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

ASMA158E 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

ASMA159S Operand 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
 0,16         2,16
 4,16         6,16
 8,16        10,16
12,16        14,16
```

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

ASMA160W Invalid BYTE function operand *xxxxxxxx*

Explanation: The value *xxxxxxxx* of the operand of the BYTE built-in function is outside the expected range of 0–255.

System action: The low-order eight bits of the operand's value are used.

Programmer response: Supply an arithmetic expression which returns an acceptable value.

Severity: 4

ASMA161W Only one TITLE statement may 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

ASMA162S 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

ASMA163W Operand not properly enclosed in quotes

Explanation: The operand of a PUNCH or TITLE statement does not begin with an apostrophe, or the operand of a PUNCH, MNOTE, or TITLE statement does not end with an apostrophe, or the ending quotation mark is not followed by a space.

System action: The statement is ignored.

Programmer response: Supply the missing apostrophe. Be sure that an apostrophes to be punched or printed as data is represented as two apostrophes.

Severity: 4

ASMA164W Operand is a null string - record not punched

Explanation: A PUNCH statement does not have any characters between its two apostrophes, or a single apostrophe to be punched as data is not represented by two apostrophes.

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 space enclosed in apostrophes.

Severity: 4

ASMA165W Unexpected name field

Explanation: The name field on this statement is not blank and is not a sequence symbol. The name field cannot be an ordinary symbol.

For example, this message is generated by the statement

```
X   ANOP
```

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 that the name is preceded with a period if you want it to be a sequence symbol.

Severity: 4

ASMA166S Sequence symbol too long - *xxxxxxxx*

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

ASMA167E 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.

For example, this message is generated by the statement

```
     EQU   3     Empty name field
```

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. If this message is generated for a LOCTR, when attempting to continue a location counter for an unnamed section, then first supply an appropriately named LOCTR in the unnamed section so that subsequent LOCTRs can continue by specifying the name.

Severity: 8

ASMA168C 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

ASMA169I Implicit length of symbol *symbol* **used for operand** *n*

Explanation: A length subfield was omitted from operand *n* in an SS-format machine instruction and the implicit length of *symbol* is assembled into the object code of the instruction.

System action: The instruction is assembled using an implicit length which:

- For an implicit address, is the length attribute of the first or only term in the expression representing the implicit address
- For an explicit address, is the length attribute of the first or only term in the expression representing the displacement

Programmer response: Check the instruction to ensure that the operation and operands are coded correctly. You can suppress this warning by specifying the NOIMPLEN suboption of the FLAG option.

Severity: 0

ASMA170S 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

ASMA171S Standard value too long

Explanation: The standard (default) value of a keyword parameter on a macro prototype statement is longer than 1024 characters.

System action: The parameter in error and the following parameters are ignored.

Programmer response: Shorten the standard value.

Severity: 12

ASMA172E Negative duplication factor; default=1 - *xxxxxxxx*

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 non-negative duplication factor.

Severity: 8

ASMA173S Delimiter error, expected blank - *xxxxxxxx*

Explanation: The character string *xxxxxxxx* is found 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 string is ignored and the operand processes normally. For a CNOP statement, the location counter is aligned to a halfword boundary.

Programmer response: Insert a comma, or replace the string with a space. Look for an extra operand or a missing left parenthesis.

Severity: 12

ASMA174S Delimiter error, expected blank or comma - *xxxxxxxx*

Explanation: The character string *xxxxxxxx* is found 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: Insert a comma, or replace the string with a space. Look for an extra operand or a missing left parenthesis.

Severity: 12

ASMA175S Delimiter error, expected comma - *xxxxxxxx*

Explanation: The character string *xxxxxxxx* 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: Insert a comma, or replace the string with a space. Be sure that each expression is syntactically correct and that no parentheses are omitted.

Severity: 12

ASMA176E Operand 4 must be absolute, 1-4 bytes; ignored

Explanation: If there is another message with this statement, this message is advisory. If this message appears alone, the fourth operand of an EQU statement contains one of the following errors:

- It is not an absolute term or expression whose value is from 1 to 4 bytes in size
- 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 fourth operand is ignored, and the program type of the EQU statement is set to binary zeros.

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

ASMA178S Delimiter error, expected comma or right parenthesis - *xxxxxxxx*

Explanation: The character string *xxxxxxxx* 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: Insert a comma, or replace the string with a right parenthesis. Look for a missing base field.

Severity: 12

ASMA179S Delimiter error, expected right parenthesis - *xxxxxxxx*

Explanation: The character string *xxxxxxxx* is used in a machine instruction when a right parenthesis is required.

System action: The machine instruction assembles as zero.

Programmer response: Replace the string with a right parenthesis. Look for an index field used where it is not allowed.

Severity: 12

ASMA180S 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

ASMA181S CCW operand value is outside allowable range

Explanation: One or more operands of a CCW statement are not within the following limits:
- First operand—0 to 255
- Second operand—0 to 16 777 215 (CCW, CCW0); or 0 to 2 147 483 647 (CCW1)
- Third operand—0-255 and a multiple of 8
- Fourth operand—0-65 535

System action: The CCW assembles as zero.

Programmer response: Supply valid operands.

Severity: 12

ASMA182E 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
- It contains an expression that cannot be fully evaluated

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

ASMA183E 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. Paired relocatable symbols in different LOCTRs, even though in the same CSECT, are not valid where an absolute, predefined value is required.

Severity: 8

ASMA184C 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 entry into lookahead mode (that is, before `ASMA006I Lookahead invoked`).

Severity: 16

ASMA185W 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

ASMA186E 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

ASMA187E The name field is invalid - *xxxxxxxx*

Explanation: The name field of an AMODE, RMODE, or XATTR instruction is invalid. The name field must be one of the following:
- A valid control section name
- An ENTRY name (for AMODE or XATTR with the GOFF option set)
- A valid external name (XATTR only)

If the XATTR statement uses the PSECT operand then the name field must specify either a valid control section name or ENTRY name.

System action: The instruction in error is ignored, and the name does not appear in the cross-reference listing.

Programmer response: Specify a valid name and resubmit the assembly.

Severity: 8

ASMA188E 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

ASMA189E 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

ASMA190E CATTR instruction invalid because no executable 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

ASMA191W 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

ASMA192W Lost precision - underflow to zero

Explanation: The value supplied is non-zero and is too small to be represented.

System action: The constant assembles with an exponent and fraction of zero.

Programmer response: Supply a larger value or a longer constant type.

Severity: 4

ASMA193W Lost precision - underflow to denormal

Explanation: The value supplied is non-zero and is too small to be represented in normalized form, but can be represented in denormalized form.

System action: The constant assembles with the denormalized form.

Programmer response: Supply a larger value or a longer constant type,

Severity: 4

ASMA194W Nominal value too large - overflow to MAX

Explanation: The value supplied is too large to be represented and the rounding mode of the constant indicates rounding towards zero. The value is represented as the signed maximum representable value.

System action: The constant assembles with the signed maximum value.

Programmer response: Supply a smaller value or a longer constant type.

Severity: 4

ASMA195W Nominal value too large - overflow to INF

Explanation: The value supplied is too large to be represented and the rounding mode of the constant indicates rounding away from zero. The value is represented as a signed infinity.

System action: The constant assembles with the signed special value INF.

Programmer response: Supply a smaller value or a longer constant type.

Severity: 4

ASMA196W Scaling modifier ignored for binary floating-point constant

Explanation: A scaling modifier has been included in the definition of a binary floating-point constant.

System action: The scaling modifier has been ignored.

Programmer response: Remove the scale modifier.

Severity: 4

ASMA198E Exponent modifier is not permitted for special value

Explanation: The exponent modifier is not permitted for a floating-point special value.

System action: The constant assembles as zeros.

Programmer response: Remove the exponent modifier.

Severity: 8

ASMA199E Rounding indicator invalid

Explanation: The rounding indicator for the floating-point constant is not a valid value.

System action: The operand in error and the following operands are ignored.

Programmer response: Correct the rounding indicator.

Severity: 8

ASMA201W 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 spaces in the continuation indicator field, and *not* as continuation indicators.

System action: The SO or SI in the continuation indicator field assembles as a space, 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-space 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

ASMA202W Shift-in not found at extended continuation; check data truncation - *xxxxxxxx*

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 next 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 unintentional truncation occurred.

Severity: 4

ASMA203E Unbalanced double-byte delimiters - *xxxxxxxx*

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 ASMA208E.

System action: The operand in error, and the following operands are ignored.

Programmer response: Correct the incorrect double-byte data.

Severity: 8

ASMA204E Invalid double-byte data - *xxxxxxxx*

Explanation: All data between SO and SI must be valid double-byte characters. A valid double-byte characters is defined as either X'4040' or a leitwo m20ned SI mus (charac332.7(betas)(doubT*0rangus (charaX'41'32.7(bet

haraJ-typDll be

module is generated according to the instructions described in Appendix L, “How to generate a Unicode translation table,” on page 367.

Severity: 12

ASMA212W Branch address alignment for *xxxxxxxx* **unfavorable**

Explanation: A branch address referenced by this statement might not be aligned to the required boundary for this instruction; for example, the target referenced by a Branch and Save (BAS) instruction might not be aligned on a halfword boundary.

System action: The instruction assembles as written.

Programmer response: Correct the operand if it is in error. To suppress this message, use the FLAG(NOALIGN) assembler option.

Severity: 4

ASMA213W Storage alignment for *xxxxxxxx* **unfavorable**

Explanation: An address referenced by this statement might not be aligned to the required boundary for this instruction; for example, the data referenced by a Compare and Swap (CS) instruction might not be aligned on a fullword boundary.

System action: The instruction assembles as written.

Programmer response: Correct the operand if it is in error. To suppress this message, use the FLAG(NOALIGN) assembler option, or specify a previous ACONTROL FLAG(NOALIGN) instruction.

Severity: 4

ASMA214E Invalid operand value - *&var*

Explanation: The contents of operand *&var* is invalid for one of the following reasons:

- null value
- a hexadecimal value is required in the operand of a hexadecimal conversion function
- a binary value is required in the operand of a binary conversion function
- the operand is longer than the function supports
- A decimal value is required in the operand of a decimal conversion function
- The operand is outside the range of acceptable values that the function supports

System action: The statement is ignored.

Programmer response: Supply a valid operand value.

Severity: 8

ASMA215W Relative Immediate external relocation in NOGOFF object text - *xxxxxxxx*

Explanation: The external address *xxxxxxxx* is used in a relative immediate branch instruction and the assembly is using the old object text format.

System action: A special RLD entry is created.

Programmer response: Check that the linkage editor/binder is able to process the RLD entry. On z/VM and z/VSE systems, check that the system loader and linker can support relative-external relocation types before using them.

Severity: 4

ASMA216W Quadword alignment in NOGOFF object text

Explanation: The SECTALGN option has been used to request that sections are aligned on quad word boundaries with the NOGOFF option.

System action: None - the ESD entries are created as requested.

Programmer response: Check that the linkage editor/binder is able to process the updated ESD entries.

Severity: 4

ASMA217W DFP lower clamped to zero - *xxxxxxxx*

Explanation: The Decimal Floating Point conversion routine has had to modify (or clamp) the exponent in order to fit within the format. No significant digits have been lost.

System action: The constant assembles with the correct value using the clamped exponent.

Programmer response: Modify the exponent value to remove the need for clamping.

Severity: 4

ASMA218W DFP upper clamped to zero - *xxxxxxxx*

Explanation: The Decimal Floating Point conversion routine has had to modify (or clamp) the exponent in order to fit within the format. No significant digits have been lost.

System action: The constant assembles with the correct value using the clamped exponent.

Programmer response: Modify the exponent value to remove the need for clamping.

Severity: 4

ASMA219W DFP lower clamped non zero - *xxxxxxxx*

Explanation: The Decimal Floating Point conversion routine has had to modify (or clamp) the exponent in order to fit within the format. No significant digits have been lost.

System action: The constant assembles with the correct value using the clamped exponent.

Programmer response: Modify the exponent value to remove the need for clamping.

Severity: 4

ASMA220W DFP upper clamped non zero - *xxxxxxxx*

Explanation: The Decimal Floating Point conversion routine has had to modify (or clamp) the exponent in order to fit within the format. No significant digits have been lost.

System action: The constant assembles with the correct value using the clamped exponent.

Programmer response: Modify the exponent value to remove the need for clamping.

Severity: 4

ASMA253C 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 are detected in the next assembly.

Severity: 16

ASMA254I * 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.

ASMA300W 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 suboption of the USING option by 1.

Severity: 4

ASMA301W 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 suboption of the USING option by 1.

Severity: 4

ASMA302W 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 that 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

ASMA303W Multiple address resolutions may result from this USING and the USING on 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

ASMA304W 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 at your request. You can suppress this message by reducing the value specified in the WARN suboption of the USING option by 8.

Severity: 4

ASMA305E 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

ASMA306W USING range overlaps implicit USING 0,0

Explanation: The USING range overlaps the assembler's implicit USING 0,0. This implicit USING is used to convert absolute implicit addresses in the range 0 to 4095. As a result of this USING, the assembler might not generate the expected object code.

System action: The assembly continues

Programmer response: Correct the USING statement. If you believe that it is correct, specify the FLAG(NOUSING0) option or a preceding ACONTROL FLAG(NOUSING0) instruction.

Severity: 4

ASMA307E No active USING for operand *n*

Explanation: The operand specified occurs in a section without an active USING.

System action: The instruction assembles as zero.

Programmer response: Provide a USING instruction.

Severity: 8

ASMA308E Repeated register *reg* nullifies prior USING range

Explanation: The repeated register nullifies the range specified by a prior use of that register on the same USING instruction.

System action: The statement is ignored.

Programmer response: Correct the USING instruction.

Severity: 8

ASMA309W Operand *xxxxxxxx* resolved to a displacement with no base register

Explanation: The machine instruction specifies an operand which is resolved to a baseless address when a base and displacement are expected. This might be the programmer's intent, but is generally an error.

System action: Base register zero is assembled into the object code of the instruction.

Programmer response: Check the instruction to ensure that the operation and operands are coded correctly. If you want to reference *page zero* you can specify a USING for the appropriate DSECT with a zero base register. You can suppress this warning by specifying the NOPAGE0 suboption of the FLAG option.

Severity: 4

ASMA310W Name already used in prior ALIAS or XATTR - *xxxxxxxx*

Explanation: The name specified in the ALIAS or XATTR statement has already been used in a previous ALIAS or XATTR statement.

System action: The statement is ignored.

Programmer response: Change the program so that the name is used in only one ALIAS or XATTR statement.

Severity: 4

ASMA311E 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, RSECT, 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

ASMA312E 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 using an EXTRN, CSECT, RSECT, and so on, or implicitly using 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

ASMA313E The end value specified in the USING is less than or equal to the base value

Explanation: The end value specified is less than or equal to the base value which results in a zero or negative range.

System action: The end value is ignored and the default range value is used.

Programmer response: Change the USING statement to specify an end value that is greater than the base value.

Severity: 8

ASMA314E The base and end values have differing relocation attributes

Explanation: The base and end values have differing relocation attributes; that is, they are defined in different sections.

System action: The end value is ignored and the default range value is used.

Programmer response: Change the USING statement to specify an end value that is in the same section as the base value.

Severity: 8

ASMA315E XATTR instruction invalid when NOGOFF specified

Explanation: The XATTR instruction can only be used when the GOFF option is set.

System action: The statement is ignored.

Programmer response: Either remove the XATTR statement and resubmit the assembly or ensure that the GOFF option is set and resubmit the assembly.

Severity: 8

ASMA316E Invalid PSECT symbol - *nnnnnnnn*

Explanation: The symbol *nnnnnnnn* specified in the PSECT operand is invalid for one of the following reasons:

- The name is undefined
- The name is a class name
- The name is not one of the following:
  - A valid control section name
  - A valid ENTRY name
  - A valid DXD name

System action: The PSECT operand is ignored.

Programmer response: Correct the PSECT operand and resubmit the assembly.

Severity: 8

ASMA317E Invalid ATTR symbol - *nnnnnnnn*

Explanation: The symbol *nnnnnnnn* specified in the ATTR operand is invalid for one of the following reasons:

- The name is undefined
- The name is a DSECT name
- The name is a qualifier name

System action: The ATTR operand is ignored.

Programmer response: Correct the ATTR operand and resubmit the assembly.

Severity: 8

ASMA318W Invalid message *n* **specified for SUPRWARN option. Message ignored.**

Explanation: Message *n*, specified as a suboption of option SUPRWARN, is not a valid High Level Assembler message. Message ignored.

System action: The specified message n is ignored.

Programmer response: Change the value n to a valid message, or remove it from option SUPRWARN.

Severity: 4

ASMA319W Message *n* specified for SUPRWARN option, but severity is too high. Message ignored.

Explanation: Message *n* specified as a suboption of option SUPRWARN is a valid High Level Assembler message, but the severity code of message *n* is higher than allowed by option SUPRWARN.

System action: The specified message *n* is ignored.

Programmer response: Change the value *n* to a valid message, or remove it from option SUPRWARN.

Severity: 4

ASMA320W Immediate field operand may have incorrect sign or magnitude

Explanation: The value of a signed immediate operand of a machine instruction is beyond the allowed range for the instruction, where the normal allowed range of values for a 16-bit signed immediate operand is -32768 through to 32767, and for a 32-bit signed immediate operand is -2147483648 through to 2147483647. Immediate operands used in an arithmetic context are signed, with bit 0 of the immediate field being used to hold the sign bit. This reduces by one the number of bits available to hold the absolute value of the operand; for example:

AHI r1,-30000 is valid, but

AHI r1,50000 is not because bits 1-15 of the immediate field can only hold a maximum absolute value of 32767.

System action: The required number of low-order bits of the operand's value are used.

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 non-immediate type. You can suppress this message by specifying the TYPECHECK(NOMAGNITUDE) option.

Severity: 4

ASMA321E Invalid assembler type - *xxxx*

Explanation: The fifth operand of an EQU statement contains an operand that does not match the allowed assembler type values.

System action: The fifth operand is ignored, and the assembler type of the EQU statement is set to spaces.

Programmer response: Supply a valid assembler type.

Severity: 8

ASMA322E Program type error - *xxxxxxxx*

Explanation: The program type subfield on a DC or DS assembler instruction contains one of the following errors:

- It is not an absolute term or expression whose value is from 1 to 4 bytes in size.
- 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 program type subfield is ignored, and the program type of the DC statement is set to binary zeros.

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

ASMA323W Symbol *xxxxxxxx* has incompatible type with *yyyyyyyy* register field

Explanation: A symbol was used to provide an absolute value for a register field in a machine instruction, but the assembler type assigned to the symbol does not match the expected register type for the instruction.

System action: The register value is used and the object code is generated. Execution of the code is not affected by this message.

Programmer response: Use a symbol with the correct assembler type. If the symbol was defined using the EQU assembler instruction, then the assembler type is assigned to the symbol using the fifth operand. This message can be suppressed using the TYPECHECK(NOREGISTER) assembler option.

Severity: 4

ASMA324I Symbol *xxxxxxxx* may have incompatible type with *yyyyyyyy* register field

Explanation: A symbol was used to provide an absolute value for a register field in a machine instruction, but the assembler type assigned to the symbol does not appear to match the expected register type for the instruction. Once the assembler detects the definition of a symbol with a particular assembler type, within the current piece of source code, it uses this message to highlight apparent inconsistent use of that assembler type on machine instructions.

System action: The register value is used and the object code is generated. Execution of the code is not affected by this message.

Programmer response: Use a symbol with the correct assembler type. If the symbol was defined using the

EQU assembler instruction, then the assembler type is assigned to the symbol using the fifth operand. This message can be suppressed using the TYPECHECK(NOREGISTER) assembler option.

Severity: 0

ASMA400W Error in invocation parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* is not a recognized assembler option, or is incorrectly specified.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA401N Fixed option cannot be overridden by invocation parameter - *xxxxxxxx*

Explanation: The parameter *xxxxxxxx* cannot be specified in the ASMAOPT file or as an invocation parameter because the option it is attempting to override was fixed when High Level Assembler was installed.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA402W 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 (z/OS and CMS), 121 to 133 (z/VSE), 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

ASMA404W 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 (z/OS and CMS), 125 (z/VSE), 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

ASMA409I Unable to load ASMAINFO

Explanation: The assembler attempted to load the INFO option module ASMAINFO, but the load failed.

System action: The assembly continues without listing the INFO requested.

Programmer response: Check that ASMAINFO is in a library accessible by the assembler.

Severity: 0

ASMA413C Unable to open INPUT file

Explanation: The assembler encountered an error when attempting to open the assembler input file. This is typically 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

ASMA414C Unable to open LISTING file

Explanation: The assembler encountered an error when attempting to open the assembler listing file. This is typically 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

ASMA415N Unable to open TERM file

Explanation: The assembler encountered an error when attempting to open the assembler terminal output file. This is typically 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

ASMA416C Unable to open DECK file

Explanation: The assembler encountered an error when attempting to open the assembler deck output file. This is typically 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

ASMA417C Unable to open OBJECT file

Explanation: The assembler encountered an error when attempting to open the assembler object output file. This is typically 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

ASMA418C Unable to open ADATA file

Explanation: The assembler encountered an error when attempting to open the associated data file. This is typically 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 (z/OS and CMS), or the SYSADAT file (z/VSE).

Severity: 16

ASMA419C Unable to open TRACE file

Explanation: The assembler encountered an error when attempting to open the internal trace file. This is typically 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 (z/OS and CMS), or the SYSTRAC file (z/VSE).

Severity: 16

ASMA420N Error in a *PROCESS statement parameter - ***xxxxxxxx***

Explanation: The parameter *xxxxxxxx* is not a recognized assembler option, or is incorrectly specified.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA421N Fixed option cannot be overridden by *PROCESS statement parameter - ***xxxxxxxx***

Explanation: The parameter *xxxxxxxx* cannot be specified in a *PROCESS statement parameter because the option it is attempting to override was fixed when High Level Assembler was installed.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA422N Option ***xxxxxxxx*** **is not valid in a *PROCESS statement**

Explanation: The following options cannot be specified in a *PROCESS statement:

```
ADATA|NOADATA     OBJECT|NOOBJECT
ASA|NOASA         SIZE
DECK|NODECK       SYSPARM
EXIT|NOEXIT       TERM|NOTERM
GOFF|NOGOFF       TRANSLATE|NOTRANSLATE
LANGUAGE          XOBJECT|NOXOBJECT
LINECOUNT
LIST|NOLIST
```

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA423N Option *yyyyyyyy* in a *PROCESS OVERRIDE statement conflicts with an invocation or default option. Option is not permitted in *PROCESS statement and has been ignored.

Explanation: The option *yyyyyyyy* specified in a *PROCESS OVERRIDE statement conflicts with an invocation or default option. The option is not permitted in a *PROCESS statement and has been ignored.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP is specified, the assembly continues using the invocation or default option.

Programmer response: Correct the *PROCESS OVERRIDE statement and resubmit the assembly.

Severity: 2

ASMA424W Continuation column is not blank. *PROCESS statements may not be continued.

Explanation: The continuation indicator field (normally column 72) is not blank for a *PROCESS statement. *PROCESS statements cannot 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 that can fit on the *PROCESS statement, add another *PROCESS statement to your code. You can specify a maximum of 10 *PROCESS statements.

Severity: 4

ASMA425N Option conflict in invocation parameters. *yyyyyyyy* overrides an earlier setting.

Explanation: The option *yyyyyyyy* specified as an invocation parameter in either the ASMAOPT file or the invocation parameters overrides an earlier setting of the option in either the same ASMAOPT file or the invocation parameters.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP is specified, the assembly continues using the last specified conflicting option.

Programmer response: Correct the ASMAOPT file or the invocation parameter and resubmit the assembly.

Severity: 2

ASMA426N Option conflict in *PROCESS statements. *yyyyyyyy* overrides an earlier setting.

Explanation: The option *yyyyyyyy* specified in a *PROCESS statement overrides an earlier setting of the option on the same statement or a previous *PROCESS statement.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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

ASMA427N Invocation parameter option *xxxxxxxx* ignored. This option is not valid under z/VSE.

Explanation: The option *xxxxxxxx* specified in an invocation parameter is not valid for the z/VSE operating system.

System action: If installation option PESTOP is specified, the assembly stops. If installation option NOPESTOP is specified, the assembly continues and the option is ignored.

Programmer response: Remove the option from the invocation parameter and resubmit the assembly.

Severity: 2

ASMA428N *PROCESS statement option *xxxxxxxx* ignored. This option is not valid under z/VSE.

Explanation: The option *xxxxxxxx* specified in a *PROCESS statement is not valid for the z/VSE operating system.

System action: If installation option PESTOP is specified, the assembly stops. If installation option NOPESTOP is specified, the assembly continues and the option is ignored.

Programmer response: Remove the option from the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA429W SYSPRINT LRECL should be at least 133 when GOFF/XOBJECT option is specified

Explanation: The GOFF or XOBJECT assembler option has been specified, however the logical record length of the listing file, SYSPRINT, is less than 133.

System action: If installation option PESTOP is specified, the assembly stops. If installation 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 at least 133 for SYSPRINT.

Severity: 4

ASMA430W 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

ASMA431W 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

ASMA432W Continuation statement may be in error - comma omitted from continued statement.

Explanation: The continuation record starts in the continue column (normally 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

ASMA433W 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 (normally 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

ASMA434N GOFF/XOBJECT option specified, option LIST(133) will be used

Explanation: You specified the GOFF or XOBJECT option, and the LIST suboption is 121.

System action: The assembler sets the LIST suboption to 133. If installation option PESTOP is specified, the assembly stops. If installation option NOPESTOP is specified, the assembly continues.

Programmer response: To prevent this warning message, run the assembly again specifying GOFF and LIST(133).

Severity: 2

ASMA435I 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.

For an AINSERT instruction:

n
The number of the statement within the AINSERT internal buffer. This number might not reflect the statement's relative statement number within the buffer at the point of retrieval, but does reflect the relative retrieval number. This is because it is possible to insert records into the buffer after statements have been retrieved from the buffer.

xxxxxxxx
The constant AINSERT BUFFER to indicate that the statement resulted from an AINSERT instruction.

vvvvvv
is null.

System action: See the System Action section of the error messages which immediately precede this message.

Programmer response: Refer to the Programmer Response section of the error messages which immediately precede this message.

Severity: 0

ASMA436N Attempt to override invocation parameter in a *PROCESS statement. Option *yyyyyyyy* ignored.

Explanation: The option *yyyyyyyy* specified in a *PROCESS statement conflicts with an option specified either in the ASMAOPT file or in an invocation parameter.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP is specified, the assembly continues using the option specified in the ASMAOPT file or the invocation parameters.

Programmer response: Correct the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA437N Attempt to override invocation parameter in a *PROCESS statement. Suboption *yyyyyyyy* of *xxxxxxxx* option ignored.

Explanation: The suboption *yyyyyyyy* of option *xxxxxxxx* specified on a *PROCESS statement conflicts with a suboption specified in either the ASMAOPT file or in the invocation parameters.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP is specified, the assembly continues using the suboption specified in the *PROCESS OVERRIDE statement.

Programmer response: Correct the *PROCESS statement and resubmit the assembly. the assembly.

Severity: 2

ASMA438N Attempt to override ASMAOPT parameter. Option *yyyyyyyy* ignored

Explanation: The option *yyyyyyyy* specified as an invocation parameter overrides the option specified in the ASMAOPT file (CMS or z/OS) or Librarian member (z/VSE).

System action: If installation option PESTOP is specified, the assembly stops. If installation option NOPESTOP is specified, the assembly continues using the option specified in the ASMAOPT file (z/OS and CMS) or library member (z/VSE).

Programmer response: Remove the option from the invocation parameters and resubmit the assembly.

Severity: 2

ASMA439N Attempt to override ASMAOPT parameter. Suboption *yyyyyyyy* of option *xxxxxxxx* ignored

Explanation: The suboption *xxxxxxxx* of options *yyyyyyyy* specified in an invocation parameter overrides the suboption specified in the ASMAOPT file (z/OS and CMS) or library member (z/VSE).

System action: If installation option PESTOP is specified, the assembly stops. If installation option NOPESTOP is specified, the assembly continues using the suboption specified in the ASMAOPT file (z/OS and CMS) or library member (z/VSE).

Programmer response: Remove the suboption from the invocation parameters and resubmit the assembly.

Severity: 2

ASMA440N Attempt to override OVERRIDE parameter in *PROCESS statement. Option *yyyyyyyy* ignored.

Explanation: The option *yyyyyyyy* specified on a *PROCESS statement conflicts with an option specified in a previous *PROCESS OVERRIDE statement.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP is specified, the assembly continues using the option specified in the *PROCESS OVERRIDE statement.

Programmer response: Correct the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA441N Attempt to override OVERRIDE parameter in a *PROCESS statement. Suboption *yyyyyyyy* ignored.

Explanation: The suboption *yyyyyyyy* of option *xxxxxxxx* specified on a *PROCESS statement conflicts with a suboption specified in a previous *PROCESS OVERRIDE statement.

System action: If installation option PESTOP is specified, the assembler stops. If installation option NOPESTOP i specified, the assembly continues using the suboption specified in the *PROCESS OVERRIDE statement.

Programmer response: Correct the *PROCESS statement and resubmit the assembly.

Severity: 2

ASMA442N ASMAOPT internal buffer full - some options ignored.

Explanation: The length of the options list provided by the ASMAOPT file, including the delimiting commas inserted by the assembler, exceeds 32766 bytes.

System action: The record which caused the message to be generated, together with those records following, is ignored.

Programmer response: Reduce the length of the options list provided by the ASMAOPT file.

Severity: 2

ASMA443N ASMAOPT record format invalid - options provided via ASMAOPT ignored

Explanation: The ASMAOPT DD statement or ASMAOPT FILEDEF refers to a file with a record format that is neither fixed-length nor variable-length.

System action: The ASMAOPT file is not processed and any options it contains are ignored.

Programmer response: Supply a file which is either fixed-length or variable-length record format.

Severity: 2

ASMA500E Requested alignment exceeds section alignment

Explanation: The section alignment is lower than that requested on the instruction and hence the actual alignment might not be honored.

System action: The requested alignment is ignored.

Programmer response: Change the requested alignment to be less than or equal to that of the section, or ensure that the wanted alignment is achieved during program linking and loading.

Severity: 8

ASMA700I *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

ASMA701W *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: 4

ASMA702E *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

ASMA703S *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

ASMA704C *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

ASMA710I *function-name*: *function-supplied text*

Explanation: The user supplied function *function-name* has requested the assembler to issue this message with the *function-supplied text*.

System action: None

Programmer response: Check the external function documentation for the cause of this message and for the correct response.

Severity: 0

ASMA711W *function-name*: *function-supplied text*

Explanation: The user supplied function *function-name* has requested the assembler to issue this message with the *function-supplied text*.

System action: None

Programmer response: Check the external function documentation for the cause of this message and for the correct response.

Severity: 4

ASMA712E *function-name* **:** *function-supplied text*

Explanation: The user supplied function *function-name* has requested the assembler to issue this message with the *function-supplied text*.

System action: None

Programmer response: Check the external function documentation for the cause of this message and for the correct response.

Severity: 8

ASMA713S *function-name* **:** *function-supplied text*

Explanation: The user supplied function *function-name* has requested the assembler to issue this message with the *function-supplied text*.

System action: None

Programmer response: Check the external function documentation for the cause of this message and for the correct response.

Severity: 12

ASMA714C *function-name* **:** *function-supplied text*

Explanation: The user supplied function *function-name* has requested the assembler to issue this message with the *function-supplied text*.

System action: None

Programmer response: Check the external function 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.

Note:

If SYSPRINT or SYSLST (z/VSE) is unavailable, the assembler uses the macro WTO with a route code of 11 and a descriptor code of 7, or the system defaults, to output the messages.

Messages

ASMA930U LOAD OF ASMA93 PHASE FAILED; INSUFFICIENT GETVIS STORAGE OR PHASE NOT FOUND

Explanation: The assembler attempted to load the phase ASMA93, but the load failed either because there was insufficient GETVIS storage available to complete the load, or the phase 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: Check the LIBDEF chain to ensure that the sublibrary containing High Level Assembler is correctly concatenated. If it is, consider increasing the partition size.

Severity: 20

ASMA932U 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

ASMA933U 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

ASMA934U 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

ASMA935U 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

ASMA936U 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

ASMA937U 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 that 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

ASMA938U 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 that the translation table is generated according to the instructions described in Appendix K, “How to generate a translation table,” on page 365.

Severity: 20

ASMA939U 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

ASMA940U *exit-type* **exit 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

ASMA941U *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

ASMA942U *xxxxxxxx* **IS NOT A VALID MODULE**

Explanation: The default options module ASMADOPT is not in the required format for this release of the assembler. The assembler checks to ensure that the release of the module is the same as that of the assembler.

Note: This message might be produced in uppercase English, even if you have specified a different language.

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 current ASMAOPT macro.

Severity: 20

ASMA943U 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 sources, and contact IBM for support.

Severity: 20

ASMA944U LOAD OF ASMA93 MODULE FAILED. INSUFFICIENT GETMAIN 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, or the Assembler was dynamically invoked by an APF-authorized program yet the Assembler does not reside in an APF-authorized library.

Note: This message is only produced in uppercase English.

System action: The assembly stops and no listing is produced.

Programmer response: on z/OS, ensure that the correct High Level Assembler load library is available in the standard load module search order. If the assembler is invoked by an APF-authorized program then ensure that the assembler resides in an APF-authorized library. Check that there is sufficient virtual storage available to the assembler and consider increasing the region size.

On 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

ASMA945U Unable to load code page *xxxxxxxx*

Explanation: The assembler attempted to load the Code Page module called *xxxxxxxx*, but the load failed. The name of the module is determined from the value specified in the CODEPAGE option.

System action: The assembly stops.

Programmer response: Check that you have correctly specified the Code Page module using the CODEPAGE option, and that the module is in a library accessible by the assembler.

Severity: 20

ASMA946U Module ASMA*xxxx* is not a valid code page module.

Explanation: The code page module specified in the CODEPAGE option is not valid.

System action: The assembly stops.

Programmer response: Ensure that the code page module is generated according to the instructions described in Appendix L, “How to generate a Unicode translation table,” on page 367.

Severity: 20

ASMA950U End of statement flag was expected in Macro Edited Text, but was not found - MACRO EDITOR is suspect

---

ASMA951U The MACRO GENERATOR has encountered untranslatable Macro Edited Text

---

ASMA952U Bad SET symbol name field or LCL/GBL operand - check the Macro Edited Text

---

ASMA953U Bad subscript on SET symbol - check the Macro Edited Text

---

ASMA954U Character expression followed by bad subscripts - check the Macro Edited Text

ASMA955U A right parenthesis with no matching left parenthesis was found in an expression - check the Macro Edited Text or the expression analysis work area

ASMA956U Multiple subscripts or bad SET symbol terminator - check the Macro Edited Text

ASMA957U Bad terminator on created SET symbol - check the Macro Edited Text

ASMA958U Bad terminator on parameter - check the Macro Edited Text

ASMA960U A bad internal file number has been passed to the *xxxxxxxx* internal storage management routine

ASMA961U An invalid storage request has been made, or the free storage chain pointers have been destroyed

ASMA962U A zero block address or bad block number has been passed to an internal storage management routine

ASMA963U Invalid pointer at entry to utility routine

ASMA964U Macro Edited Text Flag is not ICTL

Explanation: The assembly stops because of one of the errors described in ASMA950U through ASMA964U. This is typically 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 generally 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

ASMA966U Insufficient partition GETVIS storage to load *xxxxxxxx*; increase the partition GETVIS size

Explanation: The assembler attempted to load the named phase, but there was not enough GETVIS storage available for the phase.

System action: The assembly stops and no listing is produced.

Programmer response: Increase the amount of GETVIS storage allocated to the partition.

Severity: 20

ASMA967U Insufficient partition GETVIS storage for assembly initialization; increase the partition GETVIS size

Explanation: The assembler attempted to acquire an initial block of storage, but there is not enough GETVIS storage available.

System action: The assembly stops and no listing is produced.

Programmer response: Increase the amount of GETVIS storage allocated to the partition.

Severity: 20

ASMA970U 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 sources, and contact IBM for support.

Severity: 20

ASMA971U Insufficient storage available for Macro Editor work area

---

ASMA972U 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 normally 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

ASMA974U 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 in the region (z/OS), virtual machine (CMS), or partition GETVIS (z/VSE).

System action: The assembly stops and no listing is produced.

Programmer response: Increase the region size (z/OS), the virtual machine size (CMS), or the partition GETVIS (z/VSE) size, or reduce the size requested in the SIZE option.

Severity: 20

ASMA975U 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 200 KB or 10 times the work file blocksize, plus 20 KB, of working storage in the region (z/OS), virtual machine (CMS), or partition GETVIS (z/VSE) to proceed.

System action: The assembly stops and no listing is produced.

Programmer response: Increase the region size (z/OS), virtual machine size (CMS), or the partition GETVIS (z/VSE) size, 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

ASMA976U 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 input sources, and contact IBM for support.

Severity: 20

ASMA990U 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

ASMA998U 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 is typically 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 normally 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 sources, and contact IBM for support.

Severity: 20

ASMA999U 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 typically caused by a machine or an operating system error. The assembly can generally 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 generally 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 all input or library data sets have the same device class (all DASD or all tape). Also check that file attributes such as DSORG, RECFM, LRECL, and BLKSIZE have been correctly specified.

If the I/O error is on SYSUT1, check that SYSUT1 is allocated to a single volume—the assembler does not support a multivolume work file.

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 sources, and contact IBM for support. Also, if the program assembles correctly, submit a copy of the listing and input sources of the correct assembly.

Severity: 20

ASMAHL Command Error Messages (CMS)

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 eight 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 that 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 6 FORMAT

Explanation: The module *xxxxxxxx* is not in the required format for Release 6.

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 that 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 6. If you cannot resolve the problem, contact your High Level Assembler maintenance programmer, or your IBM service representative.

Appendix G. User interface macros

The macros identified in this appendix are provided as programming interfaces by High Level Assembler.

Attention: Do not use any High Level Assembler macros, other than those identified in this appendix, as programming interfaces.

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
On z/OS and CMS, generates the Filter Management Table used by the sample ADATA user exit ASMAXADT.

ASMAXITP
Maps the parameter list passed to the assembler user exits.

© Copyright IBM Corp. 1992, 2013

Appendix H. Sample ADATA user exits (z/OS and CMS)

ASMAXADT, ASMAXADC, and ASMAXADR are sample ADATA exits supplied with High Level Assembler.

Sample ASMAXADT user exit to filter records

This 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 do not 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 79 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 88 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 88. 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-shareable.

© Copyright IBM Corp. 1992, 2013

You can specify an initial character string as part of the filter operand that is passed to the filter routine during initialization. Figure 89 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 89. 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 90 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 90. 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 91 on page 353 shows the six fullwords in the parameter list.

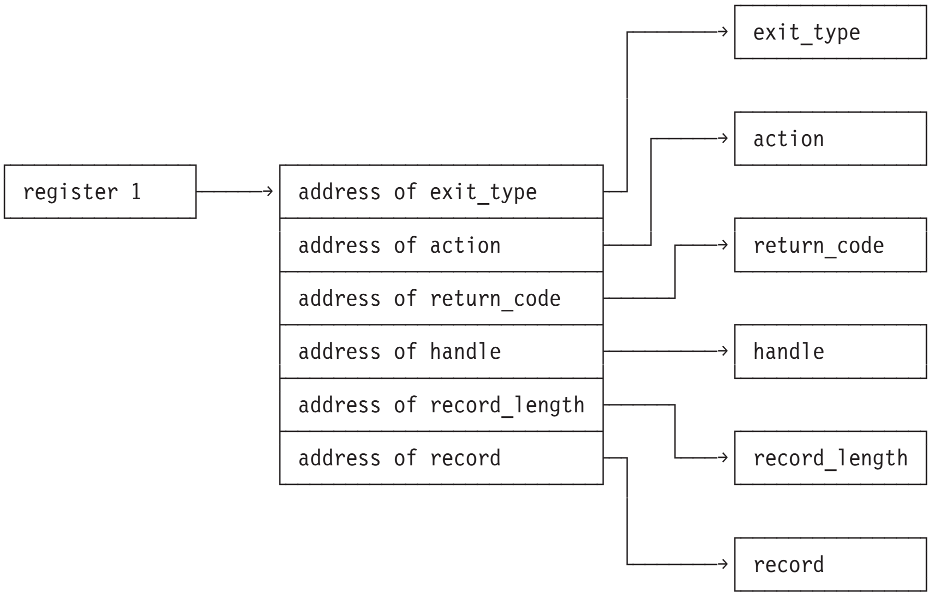


Figure 91. Filter module parameter list format

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) 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, although there might be initial character string data supplied.

The exit accepts the following return codes:

0 The open action was successful. The exit 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) 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) 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) 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:

- `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 text might be included in the ASMA940U message:

- `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.
- `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.
- `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 might be a failure in communication between the assembler and the exit.
- `SYSADATA exit initialized, but was entered for OPEN`

  The exit has been initialized, but was unexpectedly entered with an OPEN request (`action=0`). There might be a failure in communication between the assembler and the exit.
- `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 might be a failure in communication between the assembler and the exit.
- `SYSADATA exit expecting input record, zero buffer length found`

  The exit was expecting an input record, but the record length was zero. There might be a failure in communication between the assembler and the exit.
- `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.
- `All SYSADATA filter modules failed to open`

  All 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, as shown in Figure 92.

```
ASMAXFMT Title 'ADATA Exit Filter Management Table'
         ASMAXFMB (ASMAXFLU,'DUMP')
         END
```

Figure 92. 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 one of these formats:

-

Columns	**Contents**
1	Record type: “P”=Primary Input, “L”=Library
2	Space
3–10	Date, in *YYYYMMDD* format (spaces for type “L”)
11–14	Time, in *HHMM* format (spaces for type “L”)
15–58	Data set name
59–66	Member name
67–72	Volume ID where file was found
73-80	Sequencing information

-

Columns	**Contents**
1	Record type: “M”=Inline MACRO
2	Space
3–15	“PRIMARY INPUT”
16–17	Space
18–80	Macro name (can be up to 63 characters)

Figure 93 on page 356 shows a sample data set containing records written by ASMAXFLU:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8

* ASMAXFLU Filter Header Record
*Dump 10000203 00000000 00000008 00000000 00000000 00000000 00000000 00000000
*Dump 10000103 00000000 0000000A B0CC4378 E50BF900 00250000 00000000 00000000
*Dump 10000003 00000000 00000077 F1F9F9F8 F0F7F2F4 F1F9F3F3 F5F6F9F6 60F2F3F4
P 199807241933AINSERT3 ASSEMBLE A1                                ADISK 00000001
*Dump 10001003 00000000 00000068 A0CF0049 84BC0601 0F008000 40404040 40404040
*Dump 10002003 00000000 00000028 00000000 00000001 00000000 00000000 00000000
*Dump 10003003 00000000 0000009C 00000001 00000000 00000001 00000000 00010000
*Dump 10003003 00000000 0000009C 00000002 00000000 00000002 00000000 00010000
*Dump 10003003 00000000 0000009C 00000003 00000000 00000003 00000000 00010000
*Dump 10003003 00000000 0000009C 00000004 00000000 00000004 00000000 00010000
*Dump 10003003 00000000 0000009C 00000005 00000000 00000005 00000000 00010000
*Dump 10003003 00000000 0000009C 00000006 00000000 00000006 00000000 00010000
*Dump 10004203 00000000 0000002A 00000001 0000000F 01D10000 00010001 00000000
*Dump 10004403 00000000 0000001D 00080000 000F0001 40000000 00000000 C1C9D5E3
*Dump 10006003 00000000 0000003B 0002000D 0000FFFF 00060000 0000D7D9 C9D4C1D9
M PRIMARY INPUT  AINSERT_TEST_MACRO
M PRIMARY INPUT  MAC1
*Dump 10006003 00000000 00000039 00020014 00050001 00060000 0000E3C5 E2E34040
L             TEST     MACLIB   A1                        XIT1    ADISK
L             TEST     MACLIB   A1                        XIT3    ADISK
*Dump 10006003 00000000 00000033 00010014 00050002 00060000 0000C4E2 C5C3E340
L             DSECT    MACLIB   A1                        XIT2    ADISK
*Dump 10006203 00000000 000000A6 00000000 00000003 D7400000 00000000 0000C1C9
*Dump 10006203 00000000 000000A6 00000000 0000001C D7400000 00000000 0000D4C1
*Dump 10006203 00000000 000000A6 00000002 00000000 D3400000 00000000 0000E7C9
*Dump 10006203 00000000 000000A6 00000003 00000000 D3400000 00000000 0000E7C9
*Dump 10006203 00000000 000000A6 00000002 00000000 D3400000 00000000 0000E7C9
*Dump 10009003 00000000 000000B0 00000DF3 000000C8 00000012 00000000 00000000
*Dump 10000203 00000000 00000008 00010000 0000003D 00000000 00000000 00000000
* ASMAXFLU Filter Trailer Record
```

Figure 93. 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_MACROS_NUM in the X'0060' ADATA record.
- `ASMAXFLU library record missing member data?`

  Check the value of ADMXREF_MACRO_NAME_LEN in the X'0060' ADATA record.
- `ASMAXFLU Job-ID record has no file names?`

  Check the value of ADJID_FILES_NUM 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 “Preparing the filter modules” on page 352 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 do not 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 90 on page 352, which shows you how to generate an alternative filter management table.

Sample ASMAXADC user exit to control record output

This sample ADATA exit uses parameters specified on the assembler EXIT option to determine if it, or the assembler, is to perform output processing for the associated data records, and which record types are to be kept or discarded. The exit accepts records from the assembler, from either a PROCESS request or a WRITE request, and it decides whether the record should be discarded, output, or modified and then output. The exit then decides if an additional record should be inserted, and if so it builds the new record.

For a PROCESS request the exit sets the appropriate flags and returns to allow the assembler to perform the output processing. For a WRITE request the exit performs the output processing itself.

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. ASMAXADC, as supplied, has the following attributes: reusable, reenterable, amode(31), rmode(any).

Refer to Chapter 4, “Providing user exits,” on page 79 for further information about coding and preparing user exits.

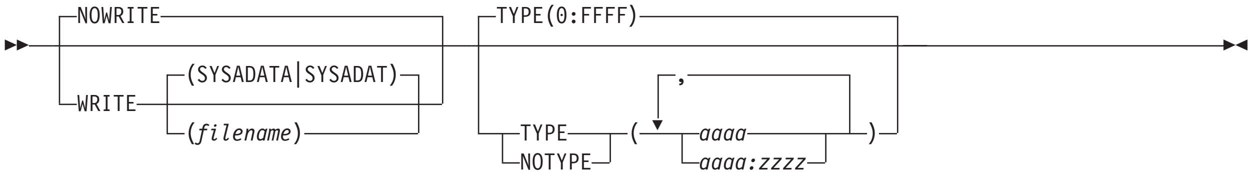
Invoking the exit

To invoke the exit, specify the EXIT assembler option as follows:

```
EXIT(ADEXIT(ASMAXADC(parameter-string)))
```

where *parameter-string* controls what action the exit performs.

Parameter string



`Default`
NOWRITE,TYPE(0:FFFF)

`Abbreviations`
W, NOW, T, N

The abbreviations shown here are the minimum number of characters allowed. You can, for example, specify TYP or NOTY.

`WRITE(`*`filename`*`)`
Specifies that the exit performs output processing for the associated data records, instead of the assembler, and optionally uses the alternative filename *filename* instead of the default SYSADATA file (z/OS and CMS) or SYSADAT file (z/VSE).

`NOWRITE`
Specifies that the assembler performs output processing for the associated data records as per normal processing.

`TYPE(`*`aaaa:zzzz`*`)`
Specifies that these associated data record types are output as per normal processing. Record types are specified as one to four character hex values, in the form of a list of single record types *aaaa*, or record type ranges *aaaa:zzzz*

`NOTYPE(`*`aaaa:zzzz`*`)`
Specifies that these associated data record types are discarded. Record types are specified the same as for TYPE.

Note: The *parameter-string* is processed left to right, so where a conflict occurs for a particular record type the last occurrence takes precedence.

The default processing is to allow all associated data record types to be output by the assembler, as per normal processing. This means, for example, that a *parameter-string* of `NOWRITE,TYPE(0:OTYPE(0030)` has the same result as `NOTYPE(0030)` alone.

Messages

When the exit detects an error it directs the assembler to issue message ASMA940U and the assembly stops. The following text might be included in the ASMA940U message:

- `SYSADATA exit not coded at same level of interface definition (3) as Assembler`

  The exit uses version 3 of the exit definition, but the assembler uses a different version.
- `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.
- `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 might be a failure in communication between the assembler and the exit.
- `SYSADATA exit initialized, but was entered for OPEN`

The exit has been initialized, but was unexpectedly entered with an OPEN request (`action=0`). There might be a failure in communication between the assembler and the exit.

- `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 might be a failure in communication between the assembler and the exit.

- `SYSADATA exit - Supplied parameters contain a syntax error`

  While parsing the supplied parameter string the exit has detected a syntax error. This is probably caused by an incorrect specification of parameters on the `ADEXIT(ASMAXADC(`*`parameter-string`*`))` suboption of the EXIT assembler option.

- `SYSADATA exit expecting input record, zero buffer length found`

  The exit was expecting an input record, but the record length was zero. There might be a failure in communication between the assembler and the exit.

Sample ASMAXADR user exit to reformat records

This sample ADATA exit uses parameters specified on the assembler EXIT option, to determine which associated data record types are to be reformatted from the High Level Assembler Release 5 format back to the Release 4 format. The exit accepts records from the assembler, from a PROCESS request, and it decides whether the record should be reformatted.

High Level Assembler Release 5 has restructured the associated data records. This sample exit is provided as a migration aid to allow existing associated data processing utilities to continue to function during the transition from Release 4 to Release 5.

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. ASMAXADR, as supplied, has the following attributes: reusable, reenterable, amode(31), rmode(any).

Refer to Chapter 4, "Providing user exits," on page 79 for further information about coding and preparing user exits.

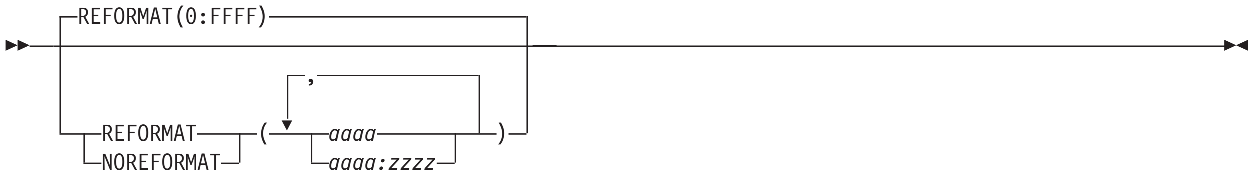
Invoking the exit

To invoke the exit, specify the EXIT assembler option as follows:

```
EXIT(ADEXIT(ASMAXADR(parameter-string)))
```

where *parameter-string* controls what action the exit performs.

Parameter string



Default

REFORMAT(0:FFFF)

Abbreviations
R, N The abbreviations shown here are the minimum number of characters allowed. You can, for example, specify REF or NORE.

REFORMAT(*aaaa:zzzz*)
Specifies that these associated data record types are reformatted from the High Level Assembler Release 5 format back to the Release 4 format. Record types are specified as one to four character hex values, in the form of a list of single record types *aaaa*, or record type ranges *aaaa:zzzz*

NOREFORMAT(*aaaa:zzzz*)
Specifies that these associated data record types are not altered. They remain in the High Level Assembler Release 5 format. Record types are specified the same as for REFORMAT.

Note: The *parameter-string* is processed left to right, so where a conflict occurs for a particular record type the last occurrence takes precedence.

The default processing is to allow all associated data record types to be reformatted. This means, for example, that a *parameter-string* of `REFORMAT(0:FFFF),NOREFORMAT(0030)` has the same result as `NOREFORMAT(0030)` alone.

Messages

When the exit detects an error it directs the assembler to issue message ASMA940U and the assembly stops. The following text might be included in the ASMA940U message:

- `SYSADATA exit not coded at same level of interface definition (3) as Assembler`

  The exit uses version 3 of the exit definition, but the assembler uses a different version.
- `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.
- `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 might be a failure in communication between the assembler and the exit.
- `SYSADATA exit initialized, but was entered for OPEN`

  The exit has been initialized, but was unexpectedly entered with an OPEN request (`action=0`). There might be a failure in communication between the assembler and the exit.
- `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 might be a failure in communication between the assembler and the exit.
- `SYSADATA exit - Supplied parameters contain a syntax error`

  While parsing the supplied parameter string the exit has detected a syntax error. This is probably caused by an incorrect specification of parameters on the `ADEXIT(ASMAXADR(`*`parameter-string`*`))` suboption of the EXIT assembler option.
- `SYSADATA exit expecting input record, zero buffer length found`

  The exit was expecting an input record, but the record length was zero. There might be a failure in communication between the assembler and the exit.

Appendix I. Sample LISTING user exit (z/OS and CMS)

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 79 for further information about coding and preparing user exits.

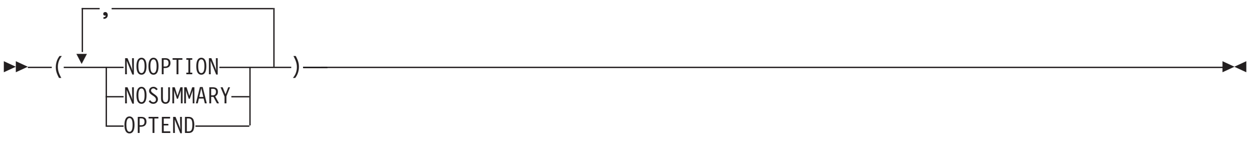
Invoking the exit

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



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.

© Copyright IBM Corp. 1992, 2013

Messages

ASMAXPRT might issue message ASMA701W as follows:

- `** ASMA701W 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.

- `** ASMA701W 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.

- `** ASMA701W 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 then assemble and link-edit it.

  This error might cause an incomplete Options Summary page.

Appendix J. Sample SOURCE user exit (z/OS and CMS)

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 79 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))
```

© Copyright IBM Corp. 1992, 2013

Appendix K. 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 72.

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 from code page 500 into ASCII characters from code page 367 is shown in Figure 94 on page 366. Specify the TRANSLATE(U1) assembler option to use this translation table.

© Copyright IBM Corp. 1992, 2013

```
* Translate from EBCDIC Code Page 500 to ASCII Code Page 368.
* Untranslated characters are set to X'00'.
&LT      SETC  'ASMALTU1'
&LT      CSECT
         DC    256X'00'
         ORG   &LT+64
         DC    X'20'            EBCDIC: X'40' space
         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'603A2340273D'  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+159
         DC    X'A4'            EBCDIC: X'A4' euro
         ORG   &LT+161
         DC    X'7E7374757677'  EBCDIC: ~stuvw
         ORG   &LT+167
         DC    X'78797A'        EBCDIC: xyz
         ORG   &LT+186
         DC    X'5B5D'          EBCDIC: []
         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 94. Sample Translation Table

Appendix L. How to generate a Unicode translation table

High Level Assembler supports the definition of EBCDIC SBCS data constants which are converted to Unicode data constants. A Unicode character (CU-type) data constant (DC) is converted to a Unicode DBCS constant using a code page module. This module is identified using the CODEPAGE assembler option (see "CODEPAGE" on page 41). The code page module must be named ASMA*xxxx*, where *xxxx* is the value supplied to the CODEPAGE assembler option.

Preparing the Unicode translation table

The user-supplied Unicode 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 1 to 8 of the module.
- The address of the translation table must be in bytes 65 to 68 of the module.
- The two code pages handled by this translation table must be in bytes 69 to 72 of the module. The number of the "from" code page must be in bytes 69 and 70, and the number of the "to" code page must be in bytes 71 and 72.

A sample Unicode translation table is shown in Figure 94 on page 366. Specify the CODEPAGE(X'FFFF') assembler option to use this translation table.

The first five DCs in this module are an eye-catcher, which is information that is easy to read from a dump. The eye-catcher includes useful information such as the date and time the module was built, and the PTF level.

© Copyright IBM Corp. 1992, 2013

```
ASMAFFFF CSECT
ASMAFFFF AMODE 31
ASMAFFFF RMODE ANY
         DC    C'ASMAFFFF'       Module name
         DC    CL8'&SYSPARM '    PTF
         DC    CL8'&SYSDATC'     Date
         DC    CL8'&SYSTIME'     Time
         DC    CL32'Module Description'
         DC    A(UNICODE)        Address of UNICODE translation table
         DC    X'47C'            From code page number
         DC    X'44B0'           To code page number
         DC    CL60              Reserved
UNICODE  DS    0H
         DC    X'0000'           EBCDIC hexadecimal value 00
         DC    X'0001'           EBCDIC hexadecimal value 01
         DC    X'0002'           EBCDIC hexadecimal value 02
         DC    X'0003'           EBCDIC hexadecimal value 03
         DC    X'009C'           EBCDIC hexadecimal value 04
         DC    X'0009'           EBCDIC hexadecimal value 05
         DC    X'0086'           EBCDIC hexadecimal value 06
         DC    X'007F'           EBCDIC hexadecimal value 07
         DC    X'0097'           EBCDIC hexadecimal value 08
         DC    X'008D'           EBCDIC hexadecimal value 09
         DC    X'008E'           EBCDIC hexadecimal value 0A
         DC    X'000B'           EBCDIC hexadecimal value 0B
         DC    X'000C'           EBCDIC hexadecimal value 0C
         DC    X'000D'           EBCDIC hexadecimal value 0D
         DC    X'000E'           EBCDIC hexadecimal value 0E
         DC    X'000F'           EBCDIC hexadecimal value 0F
         DC    X'0010'           EBCDIC hexadecimal value 10
         DC    X'0011'           EBCDIC hexadecimal value 11
         DC    X'0012'           EBCDIC hexadecimal value 12
         DC    X'0013'           EBCDIC hexadecimal value 13
         DC    X'009D'           EBCDIC hexadecimal value 14
         DC    X'0085'           EBCDIC hexadecimal value 15
         DC    X'0008'           EBCDIC hexadecimal value 16
         DC    X'0087'           EBCDIC hexadecimal value 17
         DC    X'0018'           EBCDIC hexadecimal value 18
         DC    X'0019'           EBCDIC hexadecimal value 19
         DC    X'0092'           EBCDIC hexadecimal value 1A
         DC    X'008F'           EBCDIC hexadecimal value 1B
         DC    X'001C'           EBCDIC hexadecimal value 1C
         DC    X'001D'           EBCDIC hexadecimal value 1D
         DC    X'001E'           EBCDIC hexadecimal value 1E
         DC    X'001F'           EBCDIC hexadecimal value 1F
         DC    X'0080'           EBCDIC hexadecimal value 20
         DC    X'0081'           EBCDIC hexadecimal value 21
         DC    X'0082'           EBCDIC hexadecimal value 22
         DC    X'0083'           EBCDIC hexadecimal value 23
         DC    X'0084'           EBCDIC hexadecimal value 24
         DC    X'000A'           EBCDIC hexadecimal value 25
         DC    X'0017'           EBCDIC hexadecimal value 26
         DC    X'001B'           EBCDIC hexadecimal value 27
         DC    X'0088'           EBCDIC hexadecimal value 28
         DC    X'0089'           EBCDIC hexadecimal value 29
         DC    X'008A'           EBCDIC hexadecimal value 2A
         DC    X'008B'           EBCDIC hexadecimal value 2B
         DC    X'008C'           EBCDIC hexadecimal value 2C
         DC    X'0005'           EBCDIC hexadecimal value 2D
         DC    X'0006'           EBCDIC hexadecimal value 2E
         DC    X'0007'           EBCDIC hexadecimal value 2F
         DC    X'0090'           EBCDIC hexadecimal value 30
         DC    X'0091'           EBCDIC hexadecimal value 31
         DC    X'0016'           EBCDIC hexadecimal value 32
         DC    X'0093'           EBCDIC hexadecimal value 33
         DC    X'0094'           EBCDIC hexadecimal value 34
         DC    X'0095'           EBCDIC hexadecimal value 35
         DC    X'0096'           EBCDIC hexadecimal value 36
         DC    X'0004'           EBCDIC hexadecimal value 37
```

Figure 95. Sample Unicode translation table (part 1 of 4)

```
DC    X'0098'           EBCDIC hexadecimal value 38
DC    X'0099'           EBCDIC hexadecimal value 39
DC    X'009A'           EBCDIC hexadecimal value 3A
DC    X'009B'           EBCDIC hexadecimal value 3B
DC    X'0014'           EBCDIC hexadecimal value 3C
DC    X'0015'           EBCDIC hexadecimal value 3D
DC    X'009E'           EBCDIC hexadecimal value 3E
DC    X'001A'           EBCDIC hexadecimal value 3F
DC    X'0020'           EBCDIC hexadecimal value 40
DC    X'00A0'           EBCDIC hexadecimal value 41
DC    X'00E2'           EBCDIC hexadecimal value 42
DC    X'00E4'           EBCDIC hexadecimal value 43
DC    X'00E0'           EBCDIC hexadecimal value 44
DC    X'00E1'           EBCDIC hexadecimal value 45
DC    X'00E3'           EBCDIC hexadecimal value 46
DC    X'00E5'           EBCDIC hexadecimal value 47
DC    X'00E7'           EBCDIC hexadecimal value 48
DC    X'00F1'           EBCDIC hexadecimal value 49
DC    X'005B'           EBCDIC hexadecimal value 4A
DC    X'002E'           EBCDIC hexadecimal value 4B
DC    X'003C'           EBCDIC hexadecimal value 4C
DC    X'0028'           EBCDIC hexadecimal value 4D
DC    X'002B'           EBCDIC hexadecimal value 4E
DC    X'0021'           EBCDIC hexadecimal value 4F
DC    X'0026'           EBCDIC hexadecimal value 50
DC    X'00E9'           EBCDIC hexadecimal value 51
DC    X'00EA'           EBCDIC hexadecimal value 52
DC    X'00EB'           EBCDIC hexadecimal value 53
DC    X'00E8'           EBCDIC hexadecimal value 54
DC    X'00ED'           EBCDIC hexadecimal value 55
DC    X'00EE'           EBCDIC hexadecimal value 56
DC    X'00EF'           EBCDIC hexadecimal value 57
DC    X'00EC'           EBCDIC hexadecimal value 58
DC    X'00DF'           EBCDIC hexadecimal value 59
DC    X'005D'           EBCDIC hexadecimal value 5A
DC    X'0024'           EBCDIC hexadecimal value 5B
DC    X'002A'           EBCDIC hexadecimal value 5C
DC    X'0029'           EBCDIC hexadecimal value 5D
DC    X'003B'           EBCDIC hexadecimal value 5E
DC    X'005E'           EBCDIC hexadecimal value 5F
DC    X'002D'           EBCDIC hexadecimal value 60
DC    X'002F'           EBCDIC hexadecimal value 61
DC    X'00C2'           EBCDIC hexadecimal value 62
DC    X'00C4'           EBCDIC hexadecimal value 63
DC    X'00C0'           EBCDIC hexadecimal value 64
DC    X'00C1'           EBCDIC hexadecimal value 65
DC    X'00C3'           EBCDIC hexadecimal value 66
DC    X'00C5'           EBCDIC hexadecimal value 67
DC    X'00C7'           EBCDIC hexadecimal value 68
DC    X'00D1'           EBCDIC hexadecimal value 69
DC    X'00A6'           EBCDIC hexadecimal value 6A
DC    X'002C'           EBCDIC hexadecimal value 6B


D7    X'002D'           CBCDIC hexadecimal value 5C
D7    X'002F'           EBCDIC hexadecimal value 5B
D7    X'00C2'           EBCDIC hexadecimal value 5D
D7    X'00C4'           EBCDIC hexadecimal value 53
D7    X'00C0'           EBCDIC hexadecimal value 5F
D7    X'00C1'           EBCDIC hexadecimal value 5E

D7    X'00C7'           6BCDIC hexadecimal value 63
D7    X'00D1'           3BCDIC hexadecimal value 5B
D7    X'00A6'           EBCDIC hexadecimal value 65
D7    X'002C'           4BCDIC hexadecimal value 63
```

```
         DC    X'0027'           EBCDIC hexadecimal value 7D
         DC    X'003D'           EBCDIC hexadecimal value 7E
         DC    X'0022'           EBCDIC hexadecimal value 7F
         DC    X'00D8'           EBCDIC hexadecimal value 80
         DC    X'0061'           EBCDIC hexadecimal value 81
         DC    X'0062'           EBCDIC hexadecimal value 82
         DC    X'0063'           EBCDIC hexadecimal value 83
         DC    X'0064'           EBCDIC hexadecimal value 84
         DC    X'0065'           EBCDIC hexadecimal value 85
         DC    X'0066'           EBCDIC hexadecimal value 86
         DC    X'0067'           EBCDIC hexadecimal value 87
         DC    X'0068'           EBCDIC hexadecimal value 88
         DC    X'0069'           EBCDIC hexadecimal value 89
         DC    X'00AB'           EBCDIC hexadecimal value 8A
         DC    X'00BB'           EBCDIC hexadecimal value 8B
         DC    X'00F0'           EBCDIC hexadecimal value 8C
         DC    X'00FD'           EBCDIC hexadecimal value 8D
         DC    X'00FE'           EBCDIC hexadecimal value 8E
         DC    X'00B1'           EBCDIC hexadecimal value 8F
         DC    X'00B0'           EBCDIC hexadecimal value 90
         DC    X'006A'           EBCDIC hexadecimal value 91
         DC    X'006B'           EBCDIC hexadecimal value 92
         DC    X'006C'           EBCDIC hexadecimal value 93
         DC    X'006D'           EBCDIC hexadecimal value 94
         DC    X'006E'           EBCDIC hexadecimal value 95
         DC    X'006F'           EBCDIC hexadecimal value 96
         DC    X'0070'           EBCDIC hexadecimal value 97
         DC    X'0071'           EBCDIC hexadecimal value 98
         DC    X'0072'           EBCDIC hexadecimal value 99
         DC    X'00AA'           EBCDIC hexadecimal value 9A
         DC    X'00BA'           EBCDIC hexadecimal value 9B
         DC    X'00E6'           EBCDIC hexadecimal value 9C
         DC    X'00B8'           EBCDIC hexadecimal value 9D
         DC    X'00C6'           EBCDIC hexadecimal value 9E
         DC    X'20AC'           EBCDIC hexadecimal value 9F
         DC    X'00B5'           EBCDIC hexadecimal value A0
         DC    X'007E'           EBCDIC hexadecimal value A1
         DC    X'0073'           EBCDIC hexadecimal value A2
         DC    X'0074'           EBCDIC hexadecimal value A3
         DC    X'0075'           EBCDIC hexadecimal value A4
         DC    X'0076'           EBCDIC hexadecimal value A5
         DC    X'0077'           EBCDIC hexadecimal value A6
         DC    X'0078'           EBCDIC hexadecimal value A7
         DC    X'0079'           EBCDIC hexadecimal value A8
         DC    X'007A'           EBCDIC hexadecimal value A9
         DC    X'00A1'           EBCDIC hexadecimal value AA
         DC    X'00BF'           EBCDIC hexadecimal value AB
         DC    X'00D0'           EBCDIC hexadecimal value AC
         DC    X'00DD'           EBCDIC hexadecimal value AD
         DC    X'00DE'           EBCDIC hexadecimal value AE
         DC    X'00AE'           EBCDIC hexadecimal value AF
         DC    X'00A2'           EBCDIC hexadecimal value B0
         DC    X'00A3'           EBCDIC hexadecimal value B1
         DC    X'00A5'           EBCDIC hexadecimal value B2
         DC    X'00B7'           EBCDIC hexadecimal value B3
         DC    X'00A9'           EBCDIC hexadecimal value B4
         DC    X'00A7'           EBCDIC hexadecimal value B5
         DC    X'00B6'           EBCDIC hexadecimal value B6
         DC    X'00BC'           EBCDIC hexadecimal value B7
         DC    X'00BD'           EBCDIC hexadecimal value B8
         DC    X'00BE'           EBCDIC hexadecimal value B9
         DC    X'00AC'           EBCDIC hexadecimal value BA
         DC    X'007C'           EBCDIC hexadecimal value BB
         DC    X'00AF'           EBCDIC hexadecimal value BC
         DC    X'00A8'           EBCDIC hexadecimal value BD
         DC    X'00B4'           EBCDIC hexadecimal value BE
         DC    X'00D7'           EBCDIC hexadecimal value BF
         DC    X'007B'           EBCDIC hexadecimal value C0
         DC    X'0041'           EBCDIC hexadecimal value C1
```

Figure 97. Sample Unicode translation table (part 3 of 4)

```
         DC    X'0042'          EBCDIC hexadecimal value C2
         DC    X'0043'          EBCDIC hexadecimal value C3
         DC    X'0044'          EBCDIC hexadecimal value C4
         DC    X'0045'          EBCDIC hexadecimal value C5
         DC    X'0046'          EBCDIC hexadecimal value C6
         DC    X'0047'          EBCDIC hexadecimal value C7
         DC    X'0048'          EBCDIC hexadecimal value C8
         DC    X'0049'          EBCDIC hexadecimal value C9
         DC    X'00AD'          EBCDIC hexadecimal value CA
         DC    X'00F4'          EBCDIC hexadecimal value CB
         DC    X'00F6'          EBCDIC hexadecimal value CC
         DC    X'00F2'          EBCDIC hexadecimal value CD
         DC    X'00F3'          EBCDIC hexadecimal value CE
         DC    X'00F5'          EBCDIC hexadecimal value CF
         DC    X'007D'          EBCDIC hexadecimal value D0
         DC    X'004A'          EBCDIC hexadecimal value D1
         DC    X'004B'          EBCDIC hexadecimal value D2
         DC    X'004C'          EBCDIC hexadecimal value D3
         DC    X'004D'          EBCDIC hexadecimal value D4
         DC    X'004E'          EBCDIC hexadecimal value D5
         DC    X'004F'          EBCDIC hexadecimal value D6
         DC    X'0050'          EBCDIC hexadecimal value D7
         DC    X'0051'          EBCDIC hexadecimal value D8
         DC    X'0052'          EBCDIC hexadecimal value D9
         DC    X'00B9'          EBCDIC hexadecimal value DA
         DC    X'00FB'          EBCDIC hexadecimal value DB
         DC    X'00FC'          EBCDIC hexadecimal value DC
         DC    X'00F9'          EBCDIC hexadecimal value DD
         DC    X'00FA'          EBCDIC hexadecimal value DE
         DC    X'00FF'          EBCDIC hexadecimal value DF
         DC    X'005C'          EBCDIC hexadecimal value E0
         DC    X'00F7'          EBCDIC hexadecimal value E1
         DC    X'0053'          EBCDIC hexadecimal value E2
         DC    X'0054'          EBCDIC hexadecimal value E3
         DC    X'0055'          EBCDIC hexadecimal value E4
         DC    X'0056'          EBCDIC hexadecimal value E5
         DC    X'0057'          EBCDIC hexadecimal value E6
         DC    X'0058'          EBCDIC hexadecimal value E7
         DC    X'0059'          EBCDIC hexadecimal value E8
         DC    X'005A'          EBCDIC hexadecimal value E9
         DC    X'00B2'          EBCDIC hexadecimal value EA
         DC    X'00D4'          EBCDIC hexadecimal value EB
         DC    X'00D6'          EBCDIC hexadecimal value EC
         DC    X'00D2'          EBCDIC hexadecimal value ED
         DC    X'00D3'          EBCDIC hexadecimal value EE
         DC    X'00D5'          EBCDIC hexadecimal value EF
         DC    X'0030'          EBCDIC hexadecimal value F0
         DC    X'0031'          EBCDIC hexadecimal value F1
         DC    X'0032'          EBCDIC hexadecimal value F2
         DC    X'0033'          EBCDIC hexadecimal value F3
         DC    X'0034'          EBCDIC hexadecimal value F4
         DC    X'0035'          EBCDIC hexadecimal value F5
         DC    X'0036'          EBCDIC hexadecimal value F6
         DC    X'0037'          EBCDIC hexadecimal value F7
         DC    X'0038'          EBCDIC hexadecimal value F8
         DC    X'0039'          EBCDIC hexadecimal value F9
         DC    X'00B3'          EBCDIC hexadecimal value FA
         DC    X'00DB'          EBCDIC hexadecimal value FB
         DC    X'00DC'          EBCDIC hexadecimal value FC
         DC    X'00D9'          EBCDIC hexadecimal value FD
         DC    X'00DA'          EBCDIC hexadecimal value FE
         DC    X'009F'          EBCDIC hexadecimal value FF
         END
```

Figure 98. Sample Unicode translation table (part 4 of 4)

Appendix M. TYPECHECK assembler option

You can use the TYPECHECK option to control whether the assembler performs type checking of machine instruction operands.

TYPECHECK has suboptions to enable or disable different type checking behavior:

TYPECHECK(MAGNITUDE|NOMAGNITUDE)
Specifies that the assembler performs (or does not perform) magnitude validation of signed immediate-data fields of machine instruction operands.

TYPECHECK(REGISTER|NOREGISTER)
Specifies that the assembler performs (or does not perform) type checking of register fields of machine instruction operands.

Note: For details about the syntax for the TYPECHECK option, see “TYPECHECK” on page 72.

For fine control, the TYPECHECK option is also supported on the PROCESS, ACONTROL, PUSH, and POP assembler instructions.

Extensions to the DC, DS, and EQU assembler instructions

The symbol table allows each symbol to have a program type and an assembler type assigned.

The DC and DS assembler instructions allow you to specify a program type in the fourth subfield, following type_extension.

The subfield has the format P(*program_type*), where *program_type* is a 32 bit self-defining term. For example:

```
Honda     DC   HP(C'Car')'13'
Nissan    DC   HP(C'Car')'32'
Kenworth  DC   FP(C'Truk')'128'
Mack      DC   FDP(C'Truk')'101'
```

The assembler assigns a default assembler type to the symbol comprising the type_user subfield and, if specified, the type_extension subfield.

The EQU assembler instruction allows you to specify a program type operand and an assembler type operand.

You use the fourth operand (program type) to specify a 32-bit self-defining term. This value is assigned as the symbol's program type. For example:

```
Payment    EQU   13,,,C'Rate'
Bonus      EQU   42,,,12345
```

You use the fifth operand (assembler type) to specify an assembler type keyword, which is restricted to a specific set of keywords. For a list of valid keywords for the EQU instruction, see “EQU instruction” in the *HLASM Language Reference*. The value (1 to 4 bytes) is assigned as the symbol's assembler type.

For example:

```
R9         EQU   9,,,,FPR
R10        EQU   10,,,,GR
R11        EQU   11,,,,GR
```

© Copyright IBM Corp. 1992, 2013

The SYSATTRP built-in function allows you to query the program type for a symbol. The SYSATTRA built-in function allows you to query the assembler type for a symbol.

For details about the DC, DS, and EQU instructions, or the SYSATTRP and SYSATTRA built-in functions, see the *HLASM Language Reference*.

Figure 99 shows the behavior using T' and built-in functions to retrieve the original type attribute, the program type, and the assembler type for a DC symbol. Also shown is the extended DC instruction allowing the assigning of the program type to the defined symbol.

```
          Macro
 &Lab     Show_Types &Symbol
 &Original_Type   SetC T'&Symbol
 &Program_Type    SetC SYSATTRP('&Symbol')
 &Assembler_Type  SetC SYSATTRA('&Symbol')
          MNote 0,'Type Attribute via T'' is ''&Original_Type.''.'
          MNote 0,'Program Type via function is ''&Program_Type.''.'
          MNote 0,'Assembler Type via function is ''&Assembler_Type.''.
          MEnd
 *
 Show1    Show_Types Increment
+     0,Type Attribute via T' is 'F'.
+     0,Program Type via function is 'Mony'.
+     0,Assembler Type via function is 'F'.
 Show2    Show_Types PayRate
+     0,Type Attribute via T' is 'F'.
+     0,Program Type via function is 'Mony'.
+     0,Assembler Type via function is 'FD'.
 *
 Increment   DC   FP(C'Mony')'3'
 PayRate     DC   FDP(C'Mony')'42'
```

Figure 99. Behavior to assign and retrieve a symbol's types

Figure 100 on page 375 shows the behavior using T' and built-in functions to retrieve the original type attribute, the program type, and the assembler type for an EQU symbol. Also shown is the EQU instruction allowing the assigning of assembler types to symbols used to represent registers.

```
         Macro
&Lab     Show_Types &Symbol
&Original_Type   SetC T'&Symbol
&Program_Type       SetC SYSATTRP('&Symbol')
&Assembler_Type  SetC SYSATTRA('&Symbol')
         MNote 0,'Type Attribute via T'' is ''&Original_Type.''.'
         MNote 0,'Program Type via function is ''&Program_Type.''.'
         MNote 0,'Assembler Type via function is ''&Assembler_Type.''.
         MEnd
*
Show1    Show_Types R0
+     0,Type Attribute via T' is 'U'.
+     0,Program Type via function is 'Work'.
+     0,Assembler Type via function is 'GR'.
Show2    Show_Types R1
+     0,Type Attribute via T' is 'U'.
+     0,Program Type via function is ''.
+     0,Assembler Type via function is 'GR32'.
Show3    Show_Types A12
+     0,Type Attribute via T' is 'U'.
+     0,Program Type via function is ''.
+     0,Assembler Type via function is 'AR'.
Show4    Show_Types FP4
+     0,Type Attribute via T' is 'U'.
+     0,Program Type via function is 'Spam'.
+     0,Assembler Type via function is 'FPR'.
*
R0          EQU  0,,,C'Work',GR
R1          EQU  1,,,,GR32
A12         EQU  12,,,,AR
FP4         EQU  4,,,C'Rate',FPR
```

Figure 100. Behavior to assign and retrieve a symbol's register types

Type checking behavior for REGISTER

Type checking for REGISTER cause the assembler to perform type checking of register fields of machine instruction operands.

As described previously in "Extensions to the DC, DS, and EQU assembler instructions" on page 373, you can use the EQU assembler instruction to specify assembler types to be assigned to a symbol. The assembler types apply to registers. The REGISTER suboption (the default) controls whether the assembler uses these assembler types. To disable the checking, use the NOREGISTER suboption.

When it is resolving register fields of machine instructions, the assembler uses the machine instruction format, expected operand format, and expected operand length, to check for acceptable types on any symbols specified as register operands or register parts of operands.

The assembler maintains flags which track when one or more instances of a specific register assembler type has been encountered on an EQU instruction in the source code. The assembler uses these flags to decide on the depth of register type checking for that piece of source code.

The description and examples in the remainder of this section assume that checking is active, and describe assembler behavior when machine instruction register fields of specific types are being evaluated.

Access Register type checking

The following examples use the Load Access Multiple (LAM) instruction and are only concerned with the access register fields. The first and second operands are register fields requiring a resolved absolute value of 0 through to 15. This value specifies an Access Register (AR).

Each unresolved access register field is an expression composed of one or more terms. The assembler checks only the first term:

- If the term is not a symbol, no more checking is performed.
- If the assembler type of the symbol is AR, no more checking is performed
- If the assembler type of the symbol is assigned but is not AR, the assembler issues a warning message (severity 4) about a type checking conflict.
- If the assembler type of the symbol is not assigned, and the flag shows that at least one instance of an EQU statement with AR has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is not assigned,and the flag shows that no instances of an EQU statement with AR have been encountered, no more checking is performed.

Figure 101 shows an example of Access Register checking, with warning messages about incompatible symbol types, and an informational message about a symbol not assigned an assembler type due to the existence of an EQU statement with AR in the source code.

```
00000000 9AEC D00C               0000000C     30          LAM   14,12,12(13)
00000004 9AEC D00C               0000000C     31          LAM   A14,A12,12(R13)
00000008 9AEC D00C               0000000C     32          LAM   R14,R12,12(R13)
** ASMA323W Symbol 'R14' has incompatible type with access register field
** ASMA323W Symbol 'R12' has incompatible type with access register field
0000000C 9AEC D00C               0000000C     33          LAM   AR14,A12,12(R13)
** ASMA324I Symbol 'AR14' may have incompatible type with access register field
                                              34 *
                        0000000C              35 A12      EQU   12,,,,AR
                        0000000E              36 A14      EQU   14,,,,AR
                                              37 *
                        0000000C              38 R12      EQU   12,,,,GR
                        0000000D              39 R13      EQU   13,,,,GR
                        0000000E              40 R14      EQU   14,,,,GR
                                              41 *
                        0000000C              42 AR12     EQU   12
                        0000000E              43 AR14     EQU   14
```

Figure 101. Access Register type checking with AR activated

Figure 102 on page 377 shows an example of Access Register checking, with warning messages of incompatible symbol types, and tolerance of symbols not assigned an assembler type due to the lack of an EQU statement with AR in the source code.

```
00000000 9AEC D00C               0000000C     30          LAM   14,12,12(13)
00000004 9AEC D00C               0000000C     31          LAM   A14,A12,12(R13)
00000008 9AEC D00C               0000000C     32          LAM   R14,R12,12(R13)
** ASMA323W Symbol R14 has incompatible type with access register field
** ASMA323W Symbol R12 has incompatible type with access register field
0000000C 9AEC D00C               0000000C     33          LAM   AR14,A12,12(R13)
                                              34 *
                        0000000C              35 A12      EQU   12
                        0000000E              36 A14      EQU   14
                                              37 *
                        0000000C              38 R12      EQU   12,,,,GR
                        0000000D              39 R13      EQU   13,,,,GR
                        0000000E              40 R14      EQU   14,,,,GR
                                              41 *
                        0000000C              42 AR12     EQU   12
                        0000000E              43 AR14     EQU   14
```

Figure 102. Access Register type checking with AR inactive

General Register type checking

The following examples use two instructions and are only concerned with the general register fields:

- The Load (L) instruction in which the first operand is a register field requiring a resolved absolute value of 0 through to 15. This value specifies a General Register (GR) which is treated as a 32-bit General Register (GR32).
- The Load (LG) instruction in which the first operand is a register field requiring a resolved absolute value of 0 through to 15. This value specifies a General Register (GR) which is treated as a 64-bit General Register (GR64).

Assembler type GR can normally be used for both the L and LG instructions, unless symbols have been defined with types of GR32 or GR64. Once use is made of the 32-bit or 64-bit types for general registers, then the assembler becomes more restrictive in its checking. This could be helpful when you are programming for a mix of hardware architectures, or converting code from 32-bit to 64-bit hardware.

Each unresolved general register field is an expression composed of one or more terms. The assembler checks only the first term as follows for the L instruction:

- If the term is not a symbol, no more checking is performed.
- If the assembler type of the symbol is GR32, no more checking is performed
- If the assembler type of the symbol is GR, and the flag shows that at least one instance of an EQU statement with GR32 has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is assigned but is not GR or GR32, the assembler issues a warning message (severity 4) about a type checking conflict.
- If the assembler type of the symbol is not assigned, and the flags show that at least one instance of an EQU with GR or GR32 has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is not assigned,and if the flags show that no instances of an EQU with GR or GR32 have been encountered, no more checking is performed.

Each unresolved general register field is an expression composed of one or more terms. The assembler checks only the first term as follows for the LG instruction:

- If the term is not a symbol, no more checking is performed.
- If the assembler type of the symbol is GR64, no more checking is performed.

- If the assembler type of the symbol is GR, and the flag shows that at least one instance of an EQU with GR64 has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is assigned, but is not GR or GR64, the assembler issues a warning message (severity 4) about a type checking conflict.
- If the assembler type of the symbol is not assigned, and if the flags show that at least one instance of an EQU with GR or GR64 has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is not assigned,and if the flags show that no instances of an EQU with GR or GR64 have been encountered, no more checking is performed.

Figure 103 shows an example of General Register checking, with a warning message about an incompatible symbol type, and an informational message about a symbol not assigned an assembler type due to the existence of an EQU statement with GR in the source code.

```
00000000 5824 C000               00000000     30          L     2,0(4,12)
00000004 5824 C000               00000000     31          L     R2,0(R4,R12)
00000008 5824 C000               00000000     32          L     A2,0(R4,R12)
** ASMA323W Symbol A2 has incompatible type with general register field
0000000C 5824 C000               00000000     33          L     REG2,0(R4,R12)
** ASMA324I Symbol REG2 may have incompatible type with general register field
00000010 E324 C000 0004          00000000     34          LG    R2,0(R4,R12)
                                              35 *
                        00000002              36 R2       EQU   2,,,,GR
                        00000004              37 R4       EQU   4,,,,GR
                        0000000C              38 R12      EQU   12,,,,GR
                                              39 *
                        00000002              40 A2       EQU   2,,,,AR
                                              41 *
                        00000002              42 REG2     EQU   2
```

Figure 103. General Register type checking with GR activated

Figure 104 shows an example of General Register checking, with a warning message about an incompatible symbol type, and tolerance of symbols not assigned an assembler type due to the lack of an EQU statement with GR in the source code.

```
00000000 5824 C000               00000000     30          L     2,0(4,12)
00000004 5824 C000               00000000     31          L     R2,0(R4,R12)
00000008 5824 C000               00000000     32          L     A2,0(R4,R12)
** ASMA323W Symbol A2 has incompatible type with general register field
0000000C 5824 C000               00000000     33          L     REG2,0(R4,R12)
00000010 E334 C000 0004          00000000     34          LG    R3,0(R4,R12)
                                              35 *
                        00000002              36 R2       EQU   2
                        00000003              37 R3       EQU   3
                        00000004              38 R4       EQU   4
                        0000000C              39 R12      EQU   12
                                              40 *
                        00000002              41 A2       EQU   2,,,,AR
                                              42 *
                        00000002              43 REG2     EQU   2
```

Figure 104. General Register type checking with GR inactive

Figure 105 on page 379 shows an example of General Register checking, with an informational message about a symbol with a GR assembler type due to the existence of an EQU statement with GR32 in the source code, and a warning message about an incompatible symbol type.

```
00000000 5824 C000              00000000     31         L     R2,0(R4,R12)
** ASMA324I Symbol R2 may have incompatible type with general register field
00000004 E334 C000 0004         00000000     32         LG    R3,0(R4,R12)
** ASMA323W Symbol R3 has incompatible type with general register field
0000000A E324 C000 0004         00000000     33         LG    R2,0(R4,R12)
                                             34 *
                        00000002             35 R2      EQU   2,,,,GR
                        00000003             36 R3      EQU   3,,,,GR32
                        00000004             37 R4      EQU   4,,,,GR
                        0000000C             38 R12     EQU   12,,,,GR
```

Figure 105. General Register type checking with GR32 activated

Figure 106 shows an example of General Register checking, with informational messages about symbols with a GR assembler type due to the existence of both an EQU statement with GR32 and an EQU statement with GR64 in the source code, and a warning message about an incompatible symbol type.

```
00000000 5824 C000              00000000     31         L     R2,0(R4,R12)
** ASMA324I Symbol R2 may have incompatible type with general register field
00000004 E334 C000 0004         00000000     32         LG    R3,0(R4,R12)
** ASMA323W Symbol R3 has incompatible type with general register field
0000000A E324 C000 0004         00000000     33         LG    R2,0(R4,R12)
** ASMA324I Symbol R2 may have incompatible type with general register field
                                             34 *
                        00000002             35 R2      EQU   2,,,,GR
                        00000003             36 R3      EQU   3,,,,GR32
                        00000004             37 R4      EQU   4,,,,GR
                        00000005             38 R5      EQU   5,,,,GR64
                        0000000C             39 R12     EQU   12,,,,GR
```

Figure 106. General Register type checking with GR32 and GR64 activated

Control Register type checking

The following examples use two instructions and are only concerned with the control register fields:

- The Load Control (LCTL) instruction, in which the first and second operands are register fields requiring a resolved absolute value of 0 through to 15. This value specifies a Control Register (CR) which is treated as a 32-bit Control Register.
- The Load Control (LCTLG) instruction, in which the first and second operands are register fields requiring a resolved absolute value of 0 through to 15. This value specifies a Control Register (CR) which is treated as a 64-bit Control Register.

Each unresolved control register field is an expression composed of one or more terms. The assembler checks only the first term:

- If the term is not a symbol, no more checking is performed.
- If the assembler type of the symbol is CR, no more checking is performed.
- If the assembler type of the symbol is assigned but is not CR, the assembler issues a warning message (severity 4) about a type checking conflict.
- If the assembler type of the symbol is not assigned, and if the flags show that at least one instance of an EQU with CR has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is not assigned,and if the flags show that no instances of an EQU with CR have been encountered, no more checking is performed.

Figure 107 shows an example of Control Register checking, with a warning message about an incompatible symbol type, and an informational message about a symbol not assigned an assembler type due to the existence of an EQU statement with CR in the source code.

```
00000000 B7EC A00C               0000000C     30          LCTL  14,12,12(10)
00000004 B7EC A00C               0000000C     31          LCTL  C14,C12,12(R10)
00000008 B72C A00C               0000000C     32          LCTL  A2,C12,12(R10)
** ASMA323W Symbol A2 has incompatible type with control register field
0000000C B7E1 A00C               0000000C     33          LCTL  C14,CON1,12(R10)
** ASMA324I Symbol CON1 may have incompatible type with control register field
00000010 EBEC A00C 002F          0000000C     34          LCTLG C14,C12,12(R10)
                                              35 *
                        0000000C              36 C12      EQU   12,,,,CR
                        0000000E              37 C14      EQU   14,,,,CR
                                              38 *
                        0000000A              39 R10      EQU   10,,,,GR
                        00000002              40 A2       EQU   2,,,,AR
                        00000001              41 CON1     EQU   1
```

Figure 107. Control Register type checking with CR activated

Figure 108 shows an example of Control Register checking, with a warning message about an incompatible symbol type, and tolerance of symbols not assigned an assembler type due to the lack of an EQU statement with CR in the source code.

```
00000000 B7EC A00C               0000000C     30          LCTL  14,12,12(10)
00000004 B7EC A00C               0000000C     31          LCTL  C14,C12,12(R10)
00000008 B72C A00C               0000000C     32          LCTL  A2,C12,12(R10)
** ASMA323W Symbol A2 has incompatible type with control register field
0000000C B7E1 A00C               0000000C     33          LCTL  C14,CON1,12(R10)
00000010 EBEC A00C 002F          0000000C     34          LCTLG C14,C12,12(R10)
                                              35 *
                        0000000C              36 C12      EQU   12
                        0000000E              37 C14      EQU   14
                                              38 *
                        0000000A              39 R10      EQU   10,,,,GR
                        00000002              40 A2       EQU   2,,,,AR
                        00000001              41 CON1     EQU   1
```

Figure 108. Control Register type checking with CR inactive

Floating-Point Register type checking

The following examples use two instructions and are only concerned with the floating-point register fields:

- The first operand of the Load Short (LE) instruction is a register field requiring a resolved absolute value of 0 through to 15. This value specifies a Floating-Point Register (FPR).
- The first operand of the Load Long (LD) instruction is a register field requiring a resolved absolute value of 0 through to 15. This value specifies a Floating-Point Register (FPR).

Each unresolved floating-point register field is an expression composed of one or more terms. The assembler checks only the first term:

- If the term is not a symbol, no more checking is performed.
- If the assembler type of the symbol is FPR, no more checking is performed.
- If the assembler type of the symbol is assigned but is not FPR, the assembler issues a warning message (severity 4) about a type checking conflict.

- If the assembler type of the symbol is not assigned, and if the flag shows that at least one instance of an EQU with FPR has been encountered, the assembler issues an informational message (severity 0) about a possible type checking conflict.
- If the assembler type of the symbol is not assigned,and if the flag shows that no instances of an EQU with FPR have been encountered, no more checking is performed.

Figure 109 shows an example of Floating-Point Register checking, with a warning message about an incompatible symbol type, and an informational message about a symbol not assigned an assembler type due to the existence of an EQU statement with FPR in the source code.

```
00000000 7845 C00C               0000000C     30          LE    4,12(5,12)
00000004 7845 C00C               0000000C     31          LE    FP4,12(R5,R12)
00000008 6825 C00C               0000000C     32          LD    A2,12(R5,R12)
** ASMA323W Symbol A2 has incompatible type with floating-point register field
0000000C 6865 C00C               0000000C     33          LD    FP6,12(R5,R12)
** ASMA324I Symbol FP6 may have incompatible type with floating-point register field
                                              34 *
                        00000004              35 FP4      EQU   4,,,,FPR
                        00000006              36 FP6      EQU   6
                                              37 *
                        00000005              38 R5       EQU   5,,,,GR
                        0000000C              39 R12      EQU   12,,,,GR
                        00000002              40 A2       EQU   2,,,,AR
```

Figure 109. Floating-Point Register type checking with FPR activated

Figure 110 shows an example of Floating-Point Register checking, with a warning message about an incompatible symbol type, and tolerance of symbols not assigned an assembler type due to the lack of an EQU statement with FPR in the source code.

```
00000000 7845 C00C               0000000C     30          LE    4,12(5,12)
00000004 7845 C00C               0000000C     31          LE    FP4,12(R5,R12)
00000008 6825 C00C               0000000C     32          LD    A2,12(R5,R12)
** ASMA323W Symbol A2 has incompatible type with floating-point register field
0000000C 6865 C00C               0000000C     33          LD    FP6,12(R5,R12)
                                              34 *
                        00000004              35 FP4      EQU   4
                        00000006              36 FP6      EQU   6
                                              37 *
                        00000005              38 R5       EQU   5,,,,GR
                        0000000C              39 R12      EQU   12,,,,GR
                        00000002              40 A2       EQU   2,,,,AR
```

Figure 110. Floating-Point Register type checking with FPR inactive

Type checking behavior for MAGNITUDE

Type checking for MAGNITUDE causes the assembler to perform magnitude validation of signed immediate-data fields of machine instruction operands. To disable the checking, use the NOMAGNITUDE suboption.

For each violation, a warning message (severity 4) is issued and object code is created.

For a 16-bit signed immediate-data field, the normal allowed range of values is -32768 through to 32767.

For a 32-bit signed immediate-data field, the normal allowed range of values is -2147483648 through to 2147483647.

Figure 111 shows the assembler behavior, with the default of MAGNITUDE, which is to issue a warning message and to generate the object code.

```
00000000 A72A 8000               00008000     12          AHI     R2,32768
** ASMA320W Immediate field operand may have incorrect sign or magnitude
00000004 A72A 7FFF               00007FFF     13          AHI     R2,32767
00000008 A72A 0001               00000001     14          AHI     R2,1
0000000C A72A 0000               00000000     15          AHI     R2,0
00000010 A72A FFFF               FFFFFFFF     16          AHI     R2,-1
00000014 A72A 8000               FFFF8000     17          AHI     R2,-32768
00000018 A72A 7FFF               FFFF7FFF     18          AHI     R2,-32769
** ASMA320W Immediate field operand may have incorrect sign or magnitude
0000001C A72A FFFF               0000FFFF     19          AHI     R2,X'FFFF'
** ASMA320W Immediate field operand may have incorrect sign or magnitude
```

Figure 111. MAGNITUDE behavior

Note: When generating object code, the assembler takes bits 16 to 31 of *Addr2* for use as the immediate-data field. Bits 0 to 15 are ignored.

Figure 112 shows the assembler behavior, with NOMAGNITUDE, which is to issue no messages and to generate the object code.

```
00000000 A72A 8000               00008000     12          AHI     R2,32768
00000004 A72A 7FFF               00007FFF     13          AHI     R2,32767
00000008 A72A 0001               00000001     14          AHI     R2,1
0000000C A72A 0000               00000000     15          AHI     R2,0
00000010 A72A FFFF               FFFFFFFF     16          AHI     R2,-1
00000014 A72A 8000               FFFF8000     17          AHI     R2,-32768
00000018 A72A 7FFF               FFFF7FFF     18          AHI     R2,-32769
0000001C A72A FFFF               0000FFFF     19          AHI     R2,X'FFFF'
```

Figure 112. NOMAGNITUDE behavior

Note: When generating object code, the assembler takes bits 16 to 31 of *Addr2* for use as the immediate-data field. Bits 0 to 15 are ignored.

Appendix N. HLASM Services Interface

This section describes in detail the workings of the HLASM Services Interface.

Communication and work areas

The interface between High Level Assembler and its I/O exits establishes a “coroutine” interaction: both the assembler and the exit routine must cooperate, with neither being fully in control of the other. All interactions take place through the I/O exit parameter list. This list is described in more detail in “Exit parameter list” on page 82.

The actions to be taken by the exit and the assembler are determined by the values of the request type (when the assembler calls the exit) and the return and reason code (when the exit returns to the assembler)

The I/O exit interface uses standard OS linkage conventions, and the parameter list follows standard OS parameter-passing conventions. (In fact, this interface is designed to allow exit routines to be written in most high-level languages.) The high-order bit of the last pointer in the I/O exit parameter list is set to 1.

When the HLASM Services Interface is present, the seventh parameter of the I/O exit parameter list (the HLASM Services Interface pointer) points to the HLASM Services Interface block, illustrated in Figure 113. The high-order bit of this parameter pointer is set to 1.

C'HSIB'	HLASM Services Interface Block Identifier
HSI block version	Version number of this block
HSI block length	Length of this block
↑ Service entry point	A(Services entry point in HLASM)
↑ Work area	A(work area provided by HLASM)
Number of argument words	Number of argument words for the service
Number of value words	Number of returned—value words for the service
Request type	Type of request
Return code	Return code from the HLASM service
Reserved (3)	Reserved (3 words)

The fields in the HLASM Services Interface Block are set by different means:

- The first five fields (fields 1 to 5) are initialized by HLASM before invoking an exit.
- The next three fields (fields 6 to 8) are set by the service requester.
- Field 9 is set by the HLASM Services Interface depending on the status of the request.
- Field 10 and on are set by the service requester or by the HLASM Services Interface.

The fields in the HLASM Services Interface Block are:

1. HLASM Services Interface block identifier: the EBCDIC characters `'HSIB'`.
2. HLASM Services Interface block version. Currently, this is set to 1.
3. HLASM Services Interface block length.
4. The address of the entry point in HLASM that services requests made by way of the HLASM Services Interface block.
5. The address of a **32**-byte doubleword-aligned work area provided by HLASM for the use of the service requester.

   These first five fields are initialized by HLASM before invoking an exit or external function. The next three fields are set by the service requester.
6. The number of argument words. Their number and contents are described below for each supported service. The maximum number of argument words for this version of the HLASM Services Interface block is 10.
7. The number of returned-value words. Their number and contents are described below for each supported service. The maximum number of returned-value words for this version of the HLASM Services Interface block is 10. This value is set by HLASM.
8. The type of service request:
   a. Get storage service (see “Get storage service” on page 385)
   b. Return storage service (see “Return storage service” on page 385)
   c. Time and date service (see “Time and date service” on page 385)
   d. Write-to-terminal service (see “Write to terminal service” on page 386)

   Each service is described below.
9. The return code provided by the HLASM Services Interface service, indicating the status of the request. The values of the return code are:

0	Service completed successfully
4	Service-dependent value
8	Service-dependent value
20	Bad or unsupported service request type
24	Wrong number of arguments for this service type
28	Invalid argument for this service type
32	Returned-values list not long enough for this service type
10. The remaining words in the list are the argument words (set by the service requester) and the returned-value words (set by the HLASM Services Interface).

Invoking the HLASM Services Interface

Use the following steps to invoke a service:

1. General Purpose Register 1 must point to the HLASM Services Interface block.
2. Set the service request type, the number of argument words, and the argument words in the HLASM Services Interface block, as described for the service you are requesting.
3. General Purpose Register 15 is the entry point to all HLASM Services Interface services. Its address is placed in the HLASM Services Interface block by HLASM. The AMODE (high order) bit set on the service entry-point address in the HLASM Services Interface Block.
4. General Purpose Register 14 contains the return address to the service requester. The assembler resets the caller's AMODE mode back to what was provided at entry.

5. All registers are restored by HLASM on return.

The supported services and their service-interface descriptions are shown below.

Get storage service

Table 39. Request to get storage

Field	**Contents**
Request type	1 (get storage)
Number of argument words: 2	**Word 1** Requested storage length **Word 2** Requested storage location: **1** Below 16 MB line **2** Below or above 16 MB line **3** Above 16 MB line
Number of returned-value words: 1	Word 1: Starting address of obtained storage
Return code	**0** Request satisfied **4** Storage not available **8** Invalid argument 1 or 2

Return storage service

Table 40. Request to return storage

Field	**Contents**
Request type	2 (return storage)
Number of argument words: 2	**Word 1** Returned storage length **Word 2** Returned storage starting address
Number of returned-value words: 0	
Return code	**0** Request satisfied **4** Request failed

Time and date service

Table 41. Request for time and date

Field	**Contents**
Request type	3 (time and date)
Number of argument words: 1	**Word 1** Result type **1** TIME DEC format
Number of returned-value words	2 **Word 1** time in unsigned packed decimal **Word 2** date in signed packed decimal
Return code	**0** Request satisfied **4** Request failed

Write to terminal service

Table 42. Write a message to the terminal or to SYSTERM

Field	Contents
Request type	4 (write to terminal)
Number of argument words: 1	**Word 1** Address of the 2-byte message length, followed by a message string of at most 120 characters. **Word 2** Target of the message **1** Write to terminal **2** Write to SYSTERM data set if available, to terminal otherwise
Number of returned-value words: 0	
Return code	**0** Request satisfied **4** Request failed

Mapping the communication and work areas

The I/O exit and service-request parameter lists are mapped by DSECTs generated by the ASMAXITP macro, which is supplied by High Level Assembler with the macros used for installing and customizing the assembler.

Appendix O. High Level Assembler for Linux on zSeries

This appendix provides information relevant to the Linux on zSeries implementation of High Level Assembler.

For information about system requirements, machine requirements, and storage requirements, see "Planning for installing High Level Assembler on zLinux" in the *HLASM Installation and Customization Guide*.

For installation details, see "Installing High Level Assembler on zLinux" in the *HLASM Installation and Customization Guide*.

For information about starting the assembler, see "Starting High Level Assembler on zLinux" in the *HLASM Installation and Customization Guide*.

For usage notes and limitations, see "Usage and limitations of High Level Assembler on zLinux" in the *HLASM Installation and Customization Guide*.

Options

There are some differences between the options available for the assembler on Linux and other versions of assembler.

Sources of assembler options

- The ASMAOPT file is not available.
- The default options module is not available, so any reference to installation default options do not apply to the Linux assembler.

For more information, see Chapter 3, "Controlling your assembly with options," on page 35.

Assembler options

There are options that are either not available, unique, or different for the Linux assembler.

NOADATA
Is the default for the ADATA option. The SYSADATA file is not available, so do not change this option.

EXIT This is not available to users on Linux (see option "ELF32").

NODECK
Default. Do not change.

OBJECT
This is the default. If you specify NOOBJECT then no object file is written to the object path.

NOGOFF
If you are using option ELF32 then do not change.

SIZE The default value is 32 MB.

ELF32 Linux only option.

The interface module elides the ELF32 option and replaces it with EX(OBX(ASMAXT2E)), and adds the OBJ option if it is not already present. See "ELF32 option" on page 388 for more details.

© Copyright IBM Corp. 1992, 2013

Chapter 4, “Providing user exits,” on page 79 is not applicable. No user exits are generally available.

Chapter 5, “Providing external functions,” on page 135 is not applicable. No external functions are available.

ELF32 option

This is a Linux only option. It is provided to allow creation of ELF32 files directly.

If the ELF32 option is specified, then changes are made to the options passed to the assembler to invoke the exit.

The following considerations apply to generated ELF32 files:

1. All section lengths are rounded up to a doubleword.
2. ELF32 name lengths are set to the section length for SD, CM, and PC items, and to 4 for LD items. The name of the section is not affected.
3. Zero-length sections are assigned length 8.
4. RSects are assigned to ELF32 section `'.text.RSECT'`, and are assigned the executable attribute.
5. CSects are assigned to ELF32 section `'.data.CSECT'`, and are also assigned the data and executable attributes.
6. Common (CM) sections are assigned ELF32 section names `'.bss.<name>'`, where `<name>` is the name of the CM section.
7. High Level Assembler for Linux on zSeries converts only OBJ object files to ELF32 format.
8. When generating ELF32 object files, the following limitations apply:
   a. A minimum of 25 KB working storage must be available
   b. The GOFF option must not be specified.
   c. Do not specify external names starting with an underscore (_), as they might conflict with Linux-generated names during the linking and loading process.
   d. Entry points (LD items) must not be specified in common sections.
   e. Q-type and CXD-type address constants are not supported.
   f. External dummy sections (DXD items, and dummy control sections referenced in Q-type address constants) are not supported.
   g. Address constants of lengths other than 4 are not supported.
   h. Only AMODE(31) and RMODE(ANY) are supported.
   i. The following internal tables are currently limited to:
      - 50 control section (SD, CM, PC) names
      - 150 external symbols (SD, CM, PC, ER. WX)
      - 100 entry point (LD) names
      - 150 address constant items (RLD) residing in RSects
      - 150 address constant items (RLD) residing in CSects
      - 50 ELF32 string-table items
      - 100 ELF32 sections
      - 250 ELF32 symbols (including ELF32 section names)

      These limits are easily increased by recompiling ASMAXT2E.
   j. Double relocations, although correctly converted, do not work. The Linux `ld` command does not add the previous contents of an adcon field when performing relocations.
   k. Zero-length Private Code (PC) sections are ignored.
   l. Negative relocations (for example, `A(0-*)`) are not supported.

ASMAXT2E messages

ASMAXT2E generates the following messages. Each message is prefixed with an assembler indication of the form `ASMA70ns`, where `n` is 0, 1, 2, 3, or 4, and `s` is one of the standard severity-indication letters `I, W, E, S, T`. This indication is followed by the word “`OBJECT: `”. Then, the message text is preceded by “`ASMAXT2E: `”.

Exit not coded at same level (2,3) as Assembler

Explanation: The ASMAXT2E module expects a different I/O exit interface from that supplied by High Level Assembler.

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 16

Exit called for other than PUNCH or OBJECT

Explanation: The ASMAXT2E module was invoked as an exit of type other than PUNCH or OBJECT.

System action: Processing is terminated.

User response: This is caused by incorrect assembler invocation options. Correct the options string and reassemble.

Severity: 16

Exit not initialized, and not entered for OPEN

Explanation: The ASMAXT2E module was expected to be open and initialized, but was not

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 16

Insufficient working storage for exit

Explanation: The ASMAXT2E module requires more working storage than is available.

System action: Processing is terminated.

User response: Specify a larger storage area, or specify a smaller SIZE option.

Severity: 16

Invalid action or operation type requested

Explanation: The ASMAXT2E module was invoked as an I/O exit with an invalid action or operation type.

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 16

Expecting input record, zero buffer length

Explanation: The ASMAXT2E module expected to receive an object file record to process, but none was present.

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 16

Close request for wrong exit type

Explanation: The ASMAXT2E module received a CLOSE request, but for the wrong exit type.

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 16

Invalid request-list options value

Explanation: The ASMAXT2E module was invoked to process a record, but an options string was provided.

System action: Processing is terminated.

User response: This is an internal error. Report the problem to IBM service.

Severity: 12

Invalid parm-string length

Explanation: The ASMAXT2E module was provided with a parameter string, but its length was invalid.

System action: Processing is terminated.

User response: Check that the assembler options specifying the I/O exit are valid.

Severity: 12

Too many ESD IDs

Explanation: The program being processed has too many ESD items with distinct ESD IDs.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of independently relocatable items.

Severity: 12

Too many RLD items

Explanation: The program being processed has too many relocatable address constants.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of address constants.

Severity: 12

Too many OBJ SD/CM/PC sections

Explanation: The program being processed has too many control sections.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of control sections.

Severity: 12

Too many ELF sections

Explanation: The program being processed requires more ELF sections than can be provided.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of independent items, or contact IBM service to request that ASMAXT2E be enhanced.

Severity: 12

String table overflow

Explanation: The program being processed requires more space in the ELF string table than can be provided.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of or length of external names, or contact IBM service to request that ASMAXT2E be enhanced.

Severity: 12

Too many LD items

Explanation: The program being processed contains too many ENTRY names.

System action: Processing is terminated.

User response: Restructure the program to reduce the number ENTRY statements, or contact IBM service to request that ASMAXT2E be enhanced.

Severity: 12

Too many ELF symbols

Explanation: The program being processed requires more ELF symbols than can be processed.

System action: Processing is terminated.

User response: Restructure the program to reduce the number of names, or contact IBM service to request that ASMAXT2E be enhanced.

Severity: 12

Insufficient storage for OBJ TXT records

Explanation: The program being processed requires more workspace than is available to process machine language instructions and data.

System action: Processing is terminated.

User response: Specify a larger storage area, or specify a smaller SIZE option.

Severity: 12

AMODE/RMODE 24 or 64 not supported in section 'xxxxxxxx'

Explanation: The program being processed contains a control section *xxxxxxxx* with an unsupported AMODE or RMODE.

System action: The section is discarded, and processing continues. No ELF object file is produced.

User response: Modify the program to specify valid AMODE and RMODE.

Severity: 8

External Dummy (XD) item 'xxxxxxxx' not supported

Explanation: The program being processed contains a dummy external control section (XD) *xxxxxxxx*.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Modify the program to remove the XD item.

Severity: 8

Adcon at xxxxxxxx in section with ID xxxx not 4 bytes long

Explanation: The program being processed contains an address constant at address *xxxxxxxx* in a control section with ESDID *xxxx* that is not 4 bytes long.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Modify the program to correct the address constant.

Severity: 8

Adcon at xxxxxxxx in section with ID xxxx not type A or V

Explanation: The program being processed contains an address constant at address *xxxxxxxx* in a control section with ESDID *xxxx* of type Q, or DXD, or CXD.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Modify the program to remove the address constant.

Severity: 8

Adcon at xxxxxxxx in section with ID xxxx requires unsupported negative relocation

Explanation: The program being processed contains an address constant at address *xxxxxxxx* in a control section with ESDID *xxxx* which requires negative relocation.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Modify the program to remove the address constant.

Severity: 8

ELF file not produced due to previous errors

Explanation: Previous errors have suppressed the production of an ELF object file.

User response: Correct the errors.

Severity: 8

Section length on END record not supported

Explanation: The OBJ END record contains the length of a control section.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: The object file might be old, possibly produced by a language translator that put the section length on the OBJ END record. (This condition does not arise with object files produced by High Level Assembler.)

Severity: 8

Invalid END-record entry point request

Explanation: The OBJ END record contains a request for a particular entry point in the program.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Remove the entry point request from the END record.

Severity: 8

LD item 'xxxxxxxx' in unsupported section is ignored

Explanation: The ENTRY name *xxxxxxxx* is in a control section that was previously discarded.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Correct the original error causing the section to be rejected.

Severity: 8

RLD item at address 'xxxxxxxx' in unsupported section with ID 'xxxx' is ignored

Explanation: The RLD item at address *xxxxxxxx* in a control section with ESDID *xxxx* is in a control section that was previously discarded.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Correct the original error causing the section to be rejected.

Severity: 8

RLD item at address 'xxxxxxxx' in unsupported section with ID 'xxxx' references unsupported section

Explanation: The RLD item at address *xxxxxxxx* in a control section with ESDID *xxxx* references a position in a control section that was previously discarded.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Correct the original error causing the section to be rejected.

Severity: 8

Section 'xxxxxxxx' ignored, addresses exceed X'FFFFFF'

Explanation: The control section named *xxxxxxxx* contains addresses either close to or exceeding X'FFFFFF'.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Reduce the size of the section, change its starting address, or (if the source program contains multiple control sections) reassemble with the NOTHREAD option.

Severity: 8

RLD items at 'xxxxxxxx' in section with ID 'xxxx' cause double relocation

Explanation: The RLD items at address *xxxxxxxx* in a control section with ESDID *xxxx* require more than a single relocation at that address.

System action: The items are discarded, and processing continues. No ELF object file is produced.

User response: Change the program so that only a single relocation is required at a given address.

Severity: 8

No TXT records in OBJ file

Explanation: The program generates no machine language instructions or data.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Verify that the program is correctly coded. (A DSECT statement might be misplaced.)

Severity: 8

Entry point in zero-length PC section rejected

Explanation: An entry point was specified in a zero-length Private Code section.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Verify that the program is correctly coded.

Severity: 8

LD item 'xxxxxxxx' in section with ID 'xxxx' lies outside its owning section

Explanation: An entry point named *xxxxxxxx* was specified that lies outside the bounds of its owning control section with ESDID *xxxx*.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Verify that the program is correctly coded.

Severity: 8

LD item 'xxxxxxxx' in COM section with ID 'xxxx' not supported

Explanation: An entry point named *xxxxxxxx* was specified in a common (COM) control section with ESDID *xxxx*.

System action: The item is discarded, and processing continues. No ELF object file is produced.

User response: Correct the program to remove the ENTRY point. (ENTRY points in COM sections are not supported. You might be able to reference items not at the origin of a COM section by specifying a constant offset in a referencing address constant.)

Severity: 8

Invalid or duplicate parm-string character ignored

Explanation: An invalid or duplicated character appears in the parameter string provided to ASMAXT2E.

System action: The item is ignored, and processing continues.

User response: Correct the options invoking ASMAXT2E to specify valid characters.

Severity: 0

nnnnnnnn SYM records ignored

Explanation: The OBJ file contained *nnnnnnnn* SYM records, produced when the TEST option is provided to High Level Assembler.

System action: The records are ignored, and processing continues.

User response: Correct the options invoking High Level Assembler.

Severity: 0

nnnnnnnn non-OBJect record(s) ignored

Explanation: The OBJ file contained *nnnnnnnn* records that are not object-module records.

System action: The records are ignored, and processing continues.

User response: The program might contain PUNCH or REPRO statements. Verify that they are not required.

Severity: 0

Zero-length Private Code (PC) section ignored

Explanation: The program contains a zero-length Private Code (PC) section.

System action: The section is ignored, and processing continues.

User response: Correct the source program.

Severity: 0

xxxxxxxx object records processed, xxxx usable control section(s)

Explanation: The program produced *xxxxxxxx* object records, and *xxxx* control sections were converted to ELF format.

Severity: 0

ELF file length X'xxxxxxxx'

Explanation: The generated ELF file is *X'xxxxxxxx'* bytes long.

Severity: 0

Appendix P. Transactional Memory exit ASMAXTXP

This appendix provides information regarding the use of the LISTING exit ASMAXTXP supplied with the High Level Assembler

Function

The LISTING exit ASMAXTXP is used to verify the instructions used within a constrained transaction. ASMAXTXP performs the following checks for constrained transactions:

1. The transaction must not exceed more than 32 instructions.
2. All instructions within the transaction must be within 256 bytes contiguous bytes of storage.
3. In addition to all regularly-restricted instructions (those restricted in a non-constrained transaction), the following instructions are also restricted:
   a. Any instruction not defined in Chapter 7 of the z/Architecture Principles of Operation.
   b. The only branching instructions are relative branches with a positive relative immediate operand.
   c. Instructions that cause a classic serialization operation. Exceptions to this rule include TRANSACTION END and any instructions that cause specific-operand serialization.
   d. All SS and SSE format instructions.
   e. A list of individual instructions that may execute in the hardware coprocessor.

Other constraints such as operand alignment and number of locations accessed are not possible to check during assembly.

Invoking the exit

To invoke the exit, specify the EXIT assembler option as follows: `EXIT(PRTEXIT(ASMAXTXP))`

The sample program ASMASTXP found in SASMSAM1 can be run with the above assembler option as an example of how ASMAXTXP processes constrained transactions.

Messages

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: HLASM R6.0 or higher required

Explanation: This message is issued because HLASM R6.0 or higher is required to run this exit.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Could not acquire memory.

Explanation: An attempt to acquire memory failed and the exit could not continue.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Invalid exit type or request type passed to exit.

Explanation: ASMAXTXP only processes valid LISTING EXIT requests; this message is issued if the listing exit request is invalid.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: TBEGINC base register nonzero.

Explanation: The TBEGINC instruction would cause a specification exception if bits in the base register field are non-zero.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: TBEGINC second operand is invalid.

Explanation: The immediate mask value used in the TBEGINC instruction does not conform to the instruction's specification.

© Copyright IBM Corp. 1992, 2013

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Transaction exceeds total byte limit.

Explanation: The transaction length is longer than 256 bytes allowed.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Transaction exceeds maximum instruction count (excluding) TBEGIN and TEND.

Explanation: The number of instructions between the TBEGIN and TEND instructions exceeds 32 instructions.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: TBEGINC allow-AR-modification control is 0.

Explanation: If the allow-AR-modification flag is 0, the exit cannot guarantee that instructions used within a transaction will not modify access registers.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Instruction is restricted.

Explanation: This message is issued if the instruction is restricted for example: a privileged instruction, modifying ARs / FRs and so on.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Relative branch target is zero or negative.

Explanation: A relative branch instruction may branch backwards in the program. This is invalid in a constrained transaction.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Relative branch target exceeds size limit.

Explanation: The relative branch will cause program flow to leave the constrained transaction without a TEND instruction being issued.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Relative branch mask is 0.

Explanation: The relative branch instruction will never execute because its mask is 0. ASMAXTXP flags this as a warning.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: STCMH with M3 field = 0 and R1 field = 6 or 7.

Explanation: The use of the STCMH instruction becomes restricted when the M3 field value is 0 and the R1 field is 6 or 7.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Warning: DC/DS within transaction.

Explanation: A DC or DS assembler instruction has been detected during assembly within a constrained transaction. This only occurs if the duplication factor on the instruction is non-zero.

ASMA701W @ ** ASMA701W LISTING: ASMAXTXP: Code section ended inside transaction.

Explanation: An END statement has been encountered within a constrained transaction that has not been terminated with a TEND instruction.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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 IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

© Copyright IBM Corp. 1992, 2013

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at http://www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Bibliography

High Level Assembler Documents

HLASM General Information, GC26-4943
HLASM Installation and Customization Guide, SC26-3494
HLASM Language Reference, SC26-4940
HLASM Programmer's Guide, SC26-4941

Toolkit Feature document

HLASM Toolkit Feature User's Guide, GC26-8710
HLASM Toolkit Feature Debug Reference Summary, GC26-8712
HLASM Toolkit Feature Interactive Debug Facility User's Guide, GC26-8709
HLASM Toolkit Feature Installation and Customization Guide, GC26-8711

Related documents (Architecture)

z/Architecture Principles of Operation, SA22-7832

Related documents for z/OS

z/OS:
z/OS MVS JCL Reference, SA23-1385
z/OS MVS JCL User's Guide, SA23-1386
z/OS MVS Programming: Assembler Services Guide, SA23-1368
z/OS MVS Programming: Assembler Services Reference, Volume 1 (ABE-HSP), SA23-1369
z/OS MVS Programming: Assembler Services Reference, Volume 2 (IAR-XCT), SA23-1370
z/OS MVS Programming: Authorized Assembler Services Guide, SA23-1371
z/OS MVS Programming: Authorized Assembler Services Reference, Volumes 1 - 4, SA23-1372 - SA23-1375
z/OS MVS Program Management: User's Guide and Reference, SA23-1393
z/OS MVS System Codes, SA38-0665
z/OS MVS System Commands, SA38-0666
z/OS MVS System Messages, Volumes 1 - 10, SA38-0668 - SA38-0677
z/OS Communications Server: SNA Programming, SC27-3674
UNIX System Services:
z/OS UNIX System Services User's Guide, SA23-2279
DFSMS/MVS:
z/OS DFSMS Program Management, SC27-1130
z/OS DFSMSdfp Utilities, SC23-6864
TSO/E (z/OS):
z/OS TSO/E Command Reference, SA32-0975
SMP/E (z/OS):
SMP/E for z/OS Messages, Codes, and Diagnosis, GA32-0883
SMP/E for z/OS Reference, SA23-2276
SMP/E for z/OS User's Guide, SA23-2277

© Copyright IBM Corp. 1992, 2013

Related documents for z/VM

z/VM: VMSES/E Introduction and Reference, GC24-6243
z/VM: Service Guide, GC24-6247
z/VM: CMS Commands and Utilities Reference, SC24-6166
z/VM: CMS File Pool Planning, Administration, and Operation, SC24-6167
z/VM: CP Planning and Administration, SC24-6178
z/VM: Saved Segments Planning and Administration, SC24-6229
z/VM: Other Components Messages and Codes, GC24-6207
z/VM: CMS and REXX/VM Messages and Codes, GC24-6161
z/VM: CP System Messages and Codes, GC24-6177
z/VM: CMS Application Development Guide, SC24-6162
z/VM: CMS Application Development Guide for Assembler, SC24-6163
z/VM: CMS User's Guide, SC24-6173
z/VM: XEDIT User's Guide, SC24-6245
z/VM: XEDIT Commands and Macros Reference, SC24-6244
z/VM: CP Commands and Utilities Reference, SC24-6175

Related documents for z/VSE

z/VSE: Guide to System Functions, SC33-8312
z/VSE: Administration, SC34-2627
z/VSE: Installation, SC34-2631
z/VSE: Planning, SC34-2635
z/VSE: System Control Statements, SC34-2637
z/VSE: Messages and Codes, Vol.1 , SC34-2632
z/VSE: Messages and Codes, Vol.2, SC34-2633
z/VSE: Messages and Codes, Vol.3, SC34-2634
REXX/VSE Reference, SC33-6642
REXX/VSE User's Guide, SC33-6641

Glossary

This glossary defines terms 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 (typically distinguished by reference to the words “assembler” or “assembly”)
- 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.

ADATA
(See **associated data**.) An assembler option causing it to produce associated data.

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) can be used. 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.

associated data
Additional information produced by the assembler containing detailed machine-readable information about the assembly.

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

© Copyright IBM Corp. 1992, 2013

responsible for linking and editing programs, to create either record format load modules or program objects. The z/OS binder is a functional replacement for the z/OS 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.

class A cross-section of program object data with uniform format, content, function, and behavioral attributes.

code page
An assignment of graphic characters and control function meanings to all code points.

code point
A 1-byte code representing one of 256 potential characters.

COMMON
A control section having a length attribute but no machine language text, for which space is reserved in the executable program.

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 space (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.

element
The unit of program object data uniquely identified by a section name and a class name.

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 systems beginning with the IBM 3081. 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.

GOFF Generalized Object File Format.

hierarchical file system
In z/OS UNIX System Services, 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. z/OS 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 *z/OS UNIX System Services User's Guide*.

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 (such as GET and LINK) 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 can 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 might be specified.

***operating system**
Software that controls the running of computer programs and which can provide scheduling, debugging, input and 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 (such as type and length) of the symbol.

Part Reference
A named subdivision of a MERGE class in a program object. A Pseudo-Register (external dummy section) or an external data item, having length and alignment attributes. Space in the loaded program is reserved for Parts (which can contain machine language text), but not for Commons or Pseudo-Registers.

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 like the directory and members of partitioned data sets.

phase The output of a single z/VSE linkage editor run. A phase is in a format suitable for loading into virtual storage

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 (z/OS), or Librarian library (z/VSE), 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
A new form of executable program supporting one or more independently relocatable loadable segments. 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 MB 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 can be used to influence the running of succeeding programs or, in the case of an abnormal end of task, might be printed for programmer analysis.

section
(1) A cross-section of program object data with a single name, consisting of elements belonging to one or more classes. (2) A control section.

segment
The aggregate of all section contributions to a given class, loaded as a single entity

into storage, and having its own relocation base address.

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 Machine language instructions and data.

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.

trimodal program execution
A function of z/Architecture that allows a program to run in 24-bit, 31-bit, or 64-bit address mode. The addressing mode is under program control.

translate table
A table used to replace one or more characters with alternative characters.

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'.

Index

Special characters

*PROCESS OVERRIDE assembler options 36
*PROCESS statements
  OVERRIDE 37
  precedence of assembler options 35
&SYSNDX system variable symbol, controlling its value using MHELP 290

Numerics

121-character format, source and object listing 12
121, LIST assembler suboption 55
133-character format, source and object listing 12
133, LIST assembler suboption 55
370, OPTABLE assembler suboption 59

A

abnormal assembly termination 146
ABOVE, suboption of SIZE 67
accessing the assembler (CMS) 183
ACONTROL statement
  COMPAT/NOCOMPAT option 42
  FLAG option 49
  LIBMAC/NOLIBMAC option 54
  RA2/NORA2 option 64
  TC/NOTC option 73
active USINGs
  in the assembler listing 16
  UHEAD, PCONTROL assembler suboption 61
ADATA
  assembler option 39, 227
  definition 401
  exit processing 110
  GOFF assembler suboption 51, 227
  WORKFILE assembler suboption 75
  XOBJECT assembler suboption 76
adding macro definitions to libraries
  CMS 195
  z/OS 171
  z/VSE 213
addressing mode 168
ADEXIT
  EXIT assembler suboption 46
  installation option 48
ALIGN
  assembler option 39, 204
  FLAG assembler suboption 48
ALIGNWARN installation option 51
alternate ddnames 152
AMODE
  binder option 166
  in ESD section of listing 12
  processing option 168
architecture level in ADATA records 239
ASA assembler option 40
ASA assembler option (CMS) 188
ASA assembler option (z/OS) 157
ASCII translation table 72
ASMAC
  cataloged procedure for assembly 150
ASMACG
  cataloged procedure for assembly and run, using the loader 176
ASMACL
  cataloged procedure for assembly and link 174
ASMACLG
  cataloged procedure for assembly, link, and run 176
ASMADATA macro 227, 349
ASMAEFNP macro 136, 349
ASMAHL command
  by FILEDEF for SYSIN 188
  error messages 346
  with filename of source 188
  with the EXIT option 189
ASMAHL command-error messages 297
ASMAOPT
  data set 155
  macro 36
  options file/library member 36
ASMAOPT file 187
ASMAOPT option file
  specifying on CMS 189
  specifying on z/OS 157
  specifying on z/VSE 208
ASMAPROF, default profile member name 63
ASMATXT2E messages 389
ASMAXADC (ADATA user exit) 357
ASMAXADR (ADATA user exit) 359
ASMAXADT (ADATA user exit) 351
ASMAXFMB macro 349, 351
ASMAXINV (SOURCE user exit) 363
ASMAXITP macro 82, 349, 386
ASMAXPRT (LISTING user exit) 361
assembler
  sample program 277
assembler cataloged procedures
  cataloged procedure for assembly 172
assembler data sets (z/OS)
  characteristics of 155
  list of 154
assembler diagnostics
  abnormal assembly termination 146
  cross reference 19
  error messages 143
  facilities 143
  macro trace facility (MHELP) 146
  MNOTE statements 144
  multicultural support 53
  reference information for statements in error 146
  register cross reference 28
  suppression of error messages and MNOTE statements 146
assembler files (CMS)
  characteristics of 187
  list of 186
assembler files (z/VSE)
  characteristics of 206
  list of 205
Assembler H Version 2 compatibility 2

Assembler information block 91
assembler language support 2
assembler listing
  *PROCESS statements 6
  121-character format 12
  133-character format 12
  CMS options that affect 6
  diagnostic cross reference and assembler summary 29
  DSECT cross reference 26
  external symbol dictionary 9
  general purpose register cross reference 28
  macro and copy code cross reference 23
  macro and copy code source summary 22
  option summary 6
  options that affect 5
  parts of 5
  relocation dictionary 18
  source and object 12, 13
  source and object, 121 character format 12
  source and object, 133 character format 17
  symbol and literal cross reference 19
  unreferenced symbols defined in CSECTs 22
  USING map 27
assembler macros
  on CMS 195
  on z/OS 171
assembler option
  OBJECT 52
assembler options 387
  *PROCESS OVERRIDE 36
  *PROCESS statements 37
  ADATA 39, 227
  ALIGN 39, 204
  ASA 40, 157, 188
  ASMAOPT 36
  at invocation 36
  BATCH 40
  CODEPAGE 41
  coding rules 37
  COMPAT 42
  conflicting 35
  DBCS 43
  DECK 43, 204
  default 37
  DISK (CMS) 63
  DXREF 44
  ERASE (CMS) 45
  ESD 45
  EXIT 46, 80
  external source 36
  external sources 35
  fixed defaults 36
  FLAG 48
  FOLD 51
  GOFF 51
  in a parameter list 37
  INFO 52
  JCL options 204
  LANGUAGE 53
  LIBMAC 54
  LINECOUNT 54
  LIST 55, 204
  MACHINE 56
  MXREF 58
  NOADATA 39, 227
  NOALIGN 39, 204
  NOASA 40, 157, 188
assembler options *(continued)*
  NOBATCH 40
  NOCOMPAT 42
  NODBCS 43
  NODECK 43, 204
  NODXREF 44
  NOERASE (CMS) 45
  NOESD 45
  NOEXIT 46
  NOFOLD 51
  NOGOFF 51
  NOINFO 52
  NOLIBMAC 54
  NOLIST 55, 204
  NOMXREF 58
  NOOBJECT 59, 204
  NOPCONTROL 61
  NOPRINT (CMS) 63
  NOPROFILE 63
  NORA2 64
  NORENT 65
  NORLD 65, 204
  NORXREF 66
  NOSEG 66
  NOSUPRWARN 69
  NOTERM 70, 204
  NOTEST 71
  NOTHREAD 71
  NOTRANSLATE 72
  NOTYPECHECK 72
  NOUSING 73
  NOWORKFILE 75
  NOXOBJECT 76
  NOXREF 76, 204
  OBJECT 59, 204
  on CMS 185
  OPTABLE 59
  override 36
  OVERRIDE on *PROCESS statement 37
  overriding defaults 37, 178
  PCONTROL 61
  PESTOP 63
  precedence 35
  PRINT (CMS) 63
  processing 166
  PROFILE 63
  RA2 64
  RENT 65
  restricted options 37
  RLD 65, 204
  RXREF 66
  sample of use 279
  SECTALGN 66
  SEG 66
  SIZE 67
  source 35
  SUPRWARN 69
  SYSPARM 69
  TERM 33, 70, 204
  TEST 71
  THREAD 71
  TRANSLATE 72
  TYPECHECK 72
  types of 35
  USING 73
  WORKFILE 75
  XOBJECT 76

assembler options *(continued)*
  XREF 76, 204
Assembler options
  overriding 178
  overriding statements in cataloged procedures 178
assembler statistics 32
assembler summary 29
assembler type 19, 21
assembling your program 199
assembly abnormal-termination messages 297
assembly error diagnostic messages 299
assembly error-diagnostic messages 297
associated data records
  ADATA compilation unit start/end record - X'0002' 242
  ADATA identification record - X'0001' 242
  architecture level 227, 239
  ASMADATA macro 227
  common header section 239
  DC extension record - X'0035' 263
  DC/DS record - X'0034' 261
  external symbol dictionary record - X'0020' 256
  job identification record - X'0000' 240
  library member and macro cross reference record - X'0062' 269
  library record - X'0060' 268
  machine instruction record - X'0036' 263
  options file information - X'000B' 247
  options record - X'0010' 247
  output file information record - X'000Aa' 243
  register cross reference record - X'0045' 267
  relocation dictionary record - X'0040' 264
  source analysis record - X'0030' 257
  statistics record - X'0090' 271
  symbol and literal cross reference record - X'0044' 266
  symbol record - X'0042' 264
  user-supplied information record - X'0070' 270
  USING map record - X'0080' 270
associated data, definition 401
ATTACH macro instruction (z/OS) 152
automatic call library
  definition 401
  secondary data set 164

B

BATCH assembler option 40
batch assembling 203
  CMS 184
  z/OS 154
batch facility on CMS 194
binder
  generalized object format 76
  options (z/OS) 166
  sample JCL 163
  z/OS
    input 164
    primary 164
    secondary 164
BLKSIZE for assembler files 207
  CMS 188
  z/OS 157
books xv

C

CALL binder option 166
CALL macro instruction 203
CALL macro instruction (z/OS) 152
calling the assembler from a problem program 203
calling the assembler from program (z/OS) 152
CASE, COMPAT assembler suboption 42
cataloged procedures
  for assembling (ASMAC) 172
  for assembling and linking (ASMACL) 174
  for assembling and running using the loader (ASMACG) 176
  for assembling, linking, and run (ASMACLG) 176
  invoking 172
  overriding 178
CATTR instruction, interaction with GOFF 52
CDLOAD macro instruction 203
characteristics of assembler data sets (z/OS) 155
characteristics of assembler files (CMS) 187
characteristics of assembler files (z/VSE) 206
class, definition 402
CLOSE calls to user exit 84
CMS
  ASMAHL command error messages 346
  assembler macros supported by 195
  assembling on 184
  batch facility 194
  diagnostic messages 346
  logical saved segments 66
  relationship to assembler 183
  running on 193
CMS assembler options 185
CMS diagnostic messages 191
CMSBATCH command 194
CODEPAGE assembler option 41
coding rules for assembler options 37
collection kits xvi
COM instruction
  interaction with GOFF 52
COMMON, definition 402
COMPAT assembler option 42
compatibility with earlier assemblers 2
concatenation of sublibraries 207
concatenation of SYSLIB data sets 158, 189, 190
COND parameter 159, 178
conditional assembly external functions 135
CONT, FLAG assembler suboption 48
CONTWARN installation option 51
COPY code libraries
  specifying in CMS 189
  specifying in z/OS 158
create, phase 209
cross reference
  binder (z/OS) 166
  data variables (z/OS) 166
  examples 5, 289
  linkage editor 211
cross reference list
  DSECT 26
CSECT instruction
  interaction with GOFF 52
Customization book xv

D

data
  variables, cross reference 166
data file
  specifying on CMS 190
data sets, assembler (z/OS)
  characteristics of 155
  list of 154
DATA, PCONTROL assembler suboption 61
DBCS assembler option 43
DD statements, overriding in cataloged procedures 178
ddnamelist 152
ddnames
  alternate 152
  SYSADATA 159, 191
  SYSIN 157, 189
  SYSLIB 158, 165, 189, 190
  SYSLIN 159, 165, 190
  SYSLMOD 165
  SYSPRINT 158, 165, 190
  SYSPUNCH 159, 190
  SYSTERM 158, 165, 190
  SYSUT1 165
  user-specified 165
DE, LANGUAGE assembler suboption 53
DECK assembler option 43, 204
default options 37
default profile member name 63
diagnostic cross reference and assembler summary 29
diagnostic messages written by CMS 191, 346
diagnostic messages, assembly error 299
DISK assembler option (CMS) 63
documents
  High Level Assembler xv, 399
  HLASM Toolkit 399
  machine instructions 399
  z/OS 399
  z/VM 399, 400
  z/VSE 400
DOS, OPTABLE assembler suboption 59
DOS/VSE assembler compatibility 2
DSECT cross reference listing format 26
DVD collection kits xvi
DXREF assembler option 44
dynamic invocation of assembler 203
dynamic invocation of assembler (z/OS) 152

E

E-Decks, processing 213
element, definition 402
ELF32 assembler option 387
ELF32 object files
  creating 388
ELF32 option
  implementation notes 388
EN, LANGUAGE assembler suboption 53
END OF MEMBER calls to user exit 84
END Record format 222
entry point restatement 182
ERASE assembler option (CMS) 45
erasing files (CMS) 45
error messages
  abnormal assembly termination messages 341
  ASMAHL command error messages (CMS) 346
  assembly error diagnostic messages 143, 299
error messages *(continued)*
  reference information 146
  suppression of 48, 146
error, binder (z/OS) 169
error, link-edit 211
ES, LANGUAGE assembler suboption 53
ESA
  OPTABLE assembler suboption 59
ESD assembler option 45
ESD Record format 219
ESDID
  in Ordinary Symbol and Literal Cross Reference 21
  in USING Map 27
examples
  cataloged procedures coding 178
  register saving and restoring coding 214
  register saving and restoring coding (CMS) 196
  register saving and restoring coding (z/OS) 180
EXEC statements, overriding in cataloged procedures 178
EXIT assembler option 46, 80, 387
EXIT option with the ASMAHL command 189
exit parameter list 82
exit types 79
exit-specific information block 92
EXITCTL assembler instruction 80
exits
  user-supplied 79
EXLITW, FLAG assembler suboption 48
external functions 135, 388
external symbol dictionary (ESD)
  entry types 9
  examples 10
  listing format 9

F

file name with ASMAHL command 188
FILEDEF with the ASMAHL command 188
files, assembler (CMS)
  characteristics of 187
  list of 186
files, assembler (z/VSE)
  characteristics of 206
  list of 205
files, linkage editor
  SYSIPT 210
  SYSLNK 210
  SYSLOG 210
  SYSLST 210
  SYSRDR 210
filter management table (ADATA user exit) 351
filter modules (ADATA user exit) 351
FIND COPY calls to user exit 84
FIND MACRO calls to user exit 84
FLAG assembler option 48
FOLD assembler option 51
format notation, description xvi
FULL
  MXREF assembler suboption 58
  XREF assembler suboption 76
function calls, external 135

G

GEN, PCONTROL assembler suboption 61
general purpose register cross reference 28

generalized object format data set 76
generate a translation table 365
generate a Unicode translation table 367
GENMOD command (CMS) 193
GOFF
  assembler option 51
  definition 403
graphic constants 43

H

hierarchical file system
  definition 403
  object data sets 156
High Level Assembler
  documents xv
High Level Assembler messages 297
High Level Assembler option summary 6
HLASM Services Interface pointer 92

I

I/O Exit Usage Statistics
  in the listing 31
ICCF, assembling on 200
identification-sequence field 14
IF statement 208
IJSYS01 210
IJSYS03 200, 206
IMPLEN, FLAG assembler suboption 48
INCLUDE control statement 165, 210
INEXIT
  EXIT assembler suboption 46
  installation option 48
INFO assembler option 52
input, binder (z/OS) 164
input, linkage editor 209
installation and customization
  book information xv
invoking cataloged procedures 172
invoking the assembler from a problem program 203
invoking the assembler from a program (z/OS) 152

J

JP, LANGUAGE assembler suboption 53

L

LANGUAGE assembler option 53
Language Reference xv
LET binder option 166
LIBEXIT
  EXIT assembler suboption 46
  installation option 48
LIBMAC assembler option 54
Library 206, 207
LIBRARY control statement 165
LIBRARY exit processing 98
license inquiry 397
Licensed Program Specifications xv
LIMIT installation option 75
LIMIT, USING assembler suboption 73
LINECOUNT assembler option 54
LINK macro instruction (z/OS) 152
linkage conventions for external functions 136
linkage editor
  control statements 211
  errors 211
  files 210
  INCLUDE statement 210
  input 209
  libraries 210
  output 211
  sample 209
  z/OS
    AMODE option 166
    CALL option 166
    errors 169
    LET option 166
    MAP option 166
    PRINT option 166
    RMODE option 166
    TSO LINK command 168
linkage, object program 181
linker
  definition 403
Linux
  overview 387
Linux User's Guide xvi
LIST assembler option 55, 204
LIST binder option 166
LIST, OPTABLE assembler suboption 59
listing control instructions, printing of 14
LISTING exit processing 103
listing format 10
LITTYPE, COMPAT assembler suboption 42
LKED command (CMS) 194
LOAD command (CMS) 193
load module
  on TSO 163
  sample 163
load module modification 182
loader 163
loading user exits 81
LOADLIB (CMS) 194
logical saved segments (CMS) 66
LRECL for assembler files 207
  CMS 188
  z/OS 157

M

MACHINE assembler option 56
machine instructions 217
  documents 399
macro and copy code cross reference listing format 23
macro and copy code cross reference with LIBMAC option 23
macro and copy code source summary listing format 22
macro code libraries
  specifying in CMS 189
  specifying in z/OS 158
macro definition libraries, additions to 213
  CMS 195
  z/OS 171
macro operand
  quoted in MACROCASE 42
macro-generated statements
  format of 14
  in diagnostic messages 300
MACROCASE
  clarification of quoted macro operand 42

MACROCASE, COMPAT assembler suboption 42
macros
  ASMADATA 227, 349
  ASMAEFNP 136, 349
  ASMAXFMB 349
  ASMAXITP 82, 349
macros, error messages in 143
macros, external function calls 135
MAGNITUDE, TYPECHECK assembler suboption 72
manuals xv
map
  link-edit option 211
MAP
  binder option (z/OS) 166
  processing option 166
MAP installation option 75
MAP, USING assembler suboption 73
MAX
  LIST assembler option 55
  SIZE assembler option 67
MCALL, PCONTROL assembler suboption 61
message code format 297
messages 389
MHELP
  description 146
  global suppression—operand=32 289
  macro AIF dump—operand=4 289
  macro branch trace—operand=2 289
  macro call trace—operand=1 289
  macro entry dump—operand=16 289
  macro exit dump—operand=8 289
  macro hex dump—operand=64 289
  MHELP control on &SYSNDX 290
  MHELP suppression—operand=128 289
  sample program and listing 289
MHELP instruction
  format 146
migration considerations 2
MNOTE statements 144
MSOURCE, PCONTROL assembler suboption 61
multicultural support 53
MVS/ESA
  assembling on 149
MXREF assembler option 58

N

NOADATA assembler option 387
NOADEXIT, EXIT assembler suboption 46
NOALIGN
  assembler option 39, 204
  FLAG assembler suboption 48
NOASA assembler option 40
NOASA assembler option (CMS) 188
NOASA assembler option (z/OS) 157
NOBATCH assembler option 40
NOCALL link-edit option 166
NOCASE, COMPAT assembler suboption 42
NOCOMPAT assembler option 42
NOCONT, FLAG assembler suboption 48
NODATA, PCONTROL assembler suboption 61
NODBCS assembler option 43
NODECK assembler option 43, 204, 387
NODXREF assembler option 44
NOERASE assembler option (CMS) 45
NOESD assembler option 45
NOEXIT assembler option 46
NOEXLITW, FLAG assembler suboption 48
NOFOLD assembler option 51
NOGEN, PCONTROL assembler suboption 61
NOGOFF assembler option 51, 387
NOIMPLEN, FLAG assembler suboption 48
NOINEXIT, EXIT assembler suboption 46
NOINFO assembler option 52
NOLET binder option 166
NOLIBEXIT, EXIT assembler suboption 46
NOLIBMAC assembler option 54
NOLIMIT, USING assembler suboption 73
NOLIST assembler option 55, 204
NOLIST, OPTABLE assembler suboption 59
NOLITTYPE, COMPAT assembler suboption 42
NOMACROCASE, COMPAT assembler suboption 42
NOMAGNITUDE, TYPECHECK assembler suboption 72
NOMAP
  binder option 166
  USING assembler suboption 73
NOMCALL, PCONTROL assembler suboption 61
NOMSOURCE, PCONTROL assembler suboption 61
NOMXREF assembler option 58
NOOBJECT assembler option 59, 204, 387
NOOBJEXIT, EXIT assembler suboption 46
NOPAGE0, FLAG assembler suboption 48
NOPCONTROL assembler option 61
NOPRINT assembler option (CMS) 63
NOPRINT binder option 166
NOPROFILE assembler option 63
NOPRTEXIT, EXIT assembler suboption 46
NOPUSH, FLAG assembler suboption 48
NORA2 assembler option 64
NORECORD, FLAG assembler suboption 48
NOREGISTER, TYPECHECK assembler suboption 72
NORENT assembler option 65
NORLD assembler option 65, 204
NORXREF assembler option 66
NOSEG assembler option 66
NOSUBSTR, FLAG assembler suboption 48
NOSUPRWARN assembler option 69
NOSYSLIST, COMPAT assembler suboption 42
notation, description xvi
NOTERM assembler option 70, 204
NOTEST assembler option 71
NOTHREAD assembler option 71
NOTRANSLATE assembler option 72
NOTRMEXIT, EXIT assembler suboption 46
NOTYPECHECK assembler option 72
NOUHEAD, PCONTROL assembler suboption 61
NOUSING assembler option 73
NOUSING0, FLAG assembler suboption 48
NOWARN, USING assembler suboption 73
NOWORKFILE assembler option 75
NOXOBJECT assembler option 76
NOXREF assembler option 76, 204

O

object
  extended format 76
  file format 52, 219, 403
  modules 165, 210, 217
  program linkage 181
  program migration 3
OBJECT assembler option 52, 59, 204, 387
OBJECT exit processing 107

OBJEXIT
  EXIT assembler suboption 46
  installation option 48
OFF, PCONTROL assembler suboption 61
ON statement 208
ON, PCONTROL assembler suboption 61
OPEN calls to user exit 84
OPTABLE assembler option 59
option file
  specifying on CMS 189
  specifying on z/OS 157
  specifying on z/VSE 208
option summary listing format 6
options
  binder (z/OS) 166
  ELF32 387
  EXIT 387
  NOADATA 387
  NODECK 387
  NOGOFF 387
  NOOBJECT 387
  not available 387
  OBJECT 387
  SIZE 387
  sources 387
options file
  overriding *PROCESS OVERRIDE option 36
options library member
  overriding *PROCESS OVERRIDE option 36
ordinary symbol and literal cross reference 19, 76
organization of this document xiii
OSRUN command (CMS) 194
output format listing 10
output, linkage editor 211
OVERRIDE on *PROCESS statement 37
overriding ddname 153
overriding default options 37, 178
overriding statements in cataloged procedures 178

P

PAGE0, FLAG assembler suboption 48
PARM field 35
Part Reference, definition 404
partitioned data set definition 404
PCONTROL assembler option 61
PDSE definition 404
PESTOP assembler option 63
phase
  create 209
  sample 209
portability
  machine instructions 217
  object modules 217
  system macros 217
precedence of assembler options 35
primary binder data set 164
PRINT assembler option (CMS) 63
PRINT binder option 166
PROCESS calls to user exit 84
PROCESS COPY calls to user exit 84
PROCESS MACRO calls to user exit 84
processing E-Decks 213
processor time for the assembly 32
PROFILE assembler option 63
PROFMEM, default profile member name 63
program execution 169
program fetch definition 405
program library definition 405
program management binder definition 405
program module definition 405
program module modification 182
program object, definition 405
program termination 214
  CMS 196
  z/OS 180
program type 19, 21
Programmer's Guide xvi
PRTEXIT
  assembler suboption 46
  installation option 48
publications xv
  DVD xvi
PUNCH exit processing 107
PUSH
  FLAG assembler suboption 48
  level, in the assembler listing 16

R

RA2 assembler option 64
railroad track format, how to read xvi
range, in USING Map 27
READ calls to user exit 84
RECFM for assembler files 207
  CMS 188
  z/OS 157
RECORD, FLAG assembler suboption 48
RECORDINFO installation option 51
reference information for statements in error 146
register cross reference 28
REGISTER, TYPECHECK assembler suboption 72
registers, saving and restoring
  CMS 195
  z/OS 180, 182
  z/VSE 214
REINIT calls to user exit 84
relocation dictionary
  examples 10, 283
  listing format 18
RENT assembler option 65
request information list
  ASMAXITP mapping macro 386
residency mode 168
restoring registers
  CMS 195
  z/OS 180
  z/VSE 214
return codes 159, 208
RETURN macro instruction
  CMS 195
  z/OS 180
  z/VSE 214
RLD assembler option 65, 204
RLD Record format 221
RMODE
  binder option 166
  in ESD section of listing 12
  processing option 168
RSECT instruction
  interaction with GOFF 52
running
  CMS 193
  TSO 169

running *(continued)*
  using batch facility 194
  using LKED and OSRUN commands 194
  using LOAD and START commands 193
  using the GENMOD command 193
  z/OS 169
running programs on CMS 193
running your program 212
RXREF assembler option 66

S

S370, MACHINE assembler suboption 56
S370ESA, MACHINE assembler suboption 56
S370XA, MACHINE assembler suboption 56
S390, MACHINE assembler suboption 56
S390E, MACHINE assembler suboption 56
sample ADATA user exits 351
sample LISTING user exit 361
sample program to call the assembler dynamically 203
  z/OS 153
sample programs and listings
  assembler language features, using 277
  assembler listing description 6
  diagnostic error messages 146
  MHELP 289
sample SOURCE user exit 363
SAVE macro instruction
  CMS 195
  z/OS 180
  z/VSE 214
saving registers
  CMS 195
  z/OS 180
  z/VSE 214
secondary data set 164
SECTALGN assembler option 66
section, definition 405
SEG assembler option 66
segment, definition 406
sequence number 14
SHORT, XREF assembler suboption 76
SIZE assembler option 67, 387
softcopy publications xvi
source and object assembler listing format 12
source and object listing 12
source and object listing, 121 character format 12
source and object listing, 133 character format 17
SOURCE exit processing 95
source program migration 2
SOURCE, MXREF assembler suboption 58
stacked items xvii
START command (CMS) 193
start time of assembly 32
Static Assembler information block 91
Static Assembler Information Pointer 91
statistics, assembler 32
stop time of assembly 32
SUBLIB JCL option 207
SUBSTR, FLAG assembler suboption 48
suppression of error messages and MNOTE statements 146
SUPRWARN assembler option 69
SYM Record format 223
syntax notation, description xvi
SYSADAT 206, 208
SYSADATA data file
  specifying on CMS 190
SYSADATA data set 155, 159
SYSADATA file 187, 191
SYSIN data set 155, 157
SYSIN file 187, 189
SYSIPT 206, 207, 210
SYSLIB data set 155, 158, 165, 189
SYSLIB file 187, 190
SYSLIN data set 155, 159, 165
SYSLIN file 187, 190
SYSLIST, COMPAT assembler suboption 42
SYSLMOD data set 165
SYSLNK 200, 206, 208, 210
SYSLOG 206, 207, 210
SYSLST 206, 207, 210
SYSPARM assembler option 69
SYSPCH 206, 208
SYSPRINT data set 155, 158, 165
SYSPRINT file 187, 190
SYSPUNCH data set 155, 159
SYSPUNCH file 187, 190
SYSRDR 210
system macros 217
system variable symbols
  in diagnostic messages 300
  MHELP numbering system 289
  setting data set information 94
system-determined blocksize 157
SYSTERM data set 155, 158, 165
SYSTERM file 187, 190
SYSTERM output 32
SYSUT1 208
SYSUT1 data set 165
SYSUT1 file 187
SYSUT1 utility data file
  specifying on CMS 191
SYSUT1 utility data set
  specifying on CMS 159

T

TERM assembler option 33, 70, 204
TERM exit processing 112
terminal output 32
termination
  abnormal assembly 146
  program 214
    CMS 196
    z/OS 180
TEST assembler option 71
text, definition 406
THREAD assembler option 71
Toolkit Customization book xvi
Toolkit installation and customization book information xvi
TRANSLATE assembler option 72
translation table generation 365
TRMEXIT
  EXIT assembler suboption 46
  installation option 48
TSO
  assembling on 151
  LINK command 168
  link-edit options 168
  running 169
TXT Record format 221
TYPECHECK assembler option
  description 72, 373

U
UE, LANGUAGE assembler suboption 53
UHEAD, PCONTROL assembler suboption 61
UNI, OPTABLE assembler suboption 59
Unicode translation table generation 367
unreferenced symbols defined in CSECTs 76
unreferenced symbols defined in CSECTs listing format 22
UNREFS, XREF assembler suboption 76
user exit
  ADATA exit processing 110
  addressing mode (AMODE) 81
  ASMAXADC (ADATA user exit) 357
  ASMAXADR (ADATA user exit) 359
  ASMAXADT (ADATA user exit) 351
  ASMAXFMB (ADATA user exit) 351
  ASMAXFMT (ADATA user exit) 351
  ASMAXINV (SOURCE user exit) 363
  ASMAXPRT (LISTING user exit) 361
  calling 81
  coding example 114
  error handling 92
  EXIT assembler option 46
  exit parameter list 82
  exit-specific information block 82, 92
  failure handling 92
  filter management table (ADATA user exit) 351
  filter modules (ADATA user exit) 351
  LIBRARY exit processing 98
  linkage conventions 81
  LISTING exit processing 103
  loading 81
  locating 81
  OBJECT exit processing 107
  PUNCH exit processing 107
  reason codes 87
  residency mode (RMODE) 81
  return codes 86
  sample ADATA user exit 351, 357, 359
  sample LISTING user exit 361
  sample SOURCE user exit 363
  samples 114
  SOURCE exit processing 95
  specifying 80
  TERM exit processing 112
  types 79
  user error handling 92
user exits 388
user-specified data set 165
USING assembler option 73
USING map listing format 27
using the assembler 199
  CMS 183
  TSO 151
  z/OS 149
  z/VSE 199
using the assembler (ICCF) 200
USING0, FLAG assembler suboption 48
USINGs, active
  in the assembler listing 16
  UHEAD, PCONTROL assembler suboption 61
utility data file
  specifying on CMS 191
utility data file SYSUT1
  specifying on z/VSE 208
utility data set 155
  specifying on CMS 159
utility file 187, 206

W
WARN
  installation option 75
  USING assembler suboption 73
WORKFILE assembler option 75
WRITE calls to user exit 84

X
XA, OPTABLE assembler suboption 59
XCTL macro instruction (z/OS) 152
XOBJECT assembler option 76
XREF assembler option 76, 204
XREF binder option (z/OS) 166
XREF, MXREF assembler suboption 58

Z
z/OS
  assembler macros supported by 171
  running 169
z/OS documents 399
z/VM documents 399, 400
z/VSE
  JCL options 204
  relationship to assembler 199
  running your program 212
z/VSE documents 400
ZOP, OPTABLE assembler suboption 59
ZS-2, MACHINE assembler suboption 56
ZS-3, MACHINE assembler suboption 56
ZS-4, MACHINE assembler suboption 56
ZS, MACHINE assembler suboption 56
ZS3, OPTABLE assembler suboption 59
ZS4, OPTABLE assembler suboption 59
ZSERIES-2, MACHINE assembler suboption 56
ZSERIES-3, MACHINE assembler suboption 56
ZSERIES-4, MACHINE assembler suboption 56
ZSERIES, MACHINE assembler suboption 56

SC26-4941-06