SC26-4036-0
File No. S370-21

Program Product

Assembler H Version 2 Application Programming: Guide

Program Number 5668-962

Release 1.0

IBM

SC26-4036-0
File No. S370-21

Program Product

Assembler H Version 2 Application Programming: Guide

Program Number 5668-962

Release 1.0

IBM

First Edition (January 1983)

This edition applies to Version 2, Release 1, Modification 0, of Assembler H, Program Product 5668-962, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. This manual merges assembler information contained in OS/VS-VM/370 Assembler Programmer's Guide, GC33-4021, and OS Assembler H Programmer's Guide, SC26-3759.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370 and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

PREFACE

This manual tells how to use Assembler H Version 2, Release 1, Modification 0, Program Product 5668-962 (hereafter referred to as the Assembler H program or, simply, assembler).

Assembler H is an assembler language processor that performs high-speed assemblies on all IBM System/370, 303x, 3081, and 43xx processors, provided they are supported by any of the following operating systems: OS/VS2 MVS 3.8, MVS/Extended Architecture (MVS/XA), MVS/System Product (MVS/SP) V1, OS/VS1 Release 7, VM/SP, or VM/XA Migration Aid.

This manual is divided into three parts that relate to the various operating systems under which the assembler operates. The first part is common; the second part is labeled "OS/VS" and includes information relating to OS/VS2 MVS 3.8, MVS/XA, MVS/SP V1, and OS/VS1 Release 7; and the third part is labeled "CMS" and includes information relating to VM/SP and VM/XA Migration Aid.

NEW FEATURES

New features in the Assembler H Version 2 Program Product are:

- A program using System/370 Extended Architecture (S/370-XA) machine instructions may be assembled with Assembler H under MVS/Extended Architecture (MVS/XA), OS/VS2 MVS Release 3.8, OS/VS1 Release 7, MVS/SP V1, VM/XA Migration Aid, or VM/System Product (VM/SP). Programs using the Extended Architecture instruction set can be assembled on any system supported by the above operating systems; however, programs containing Extended Architecture instructions can only be executed on an Extended Architecture mode processor under MVS/XA, or with MVS/XA operating as a guest operating system under VM/XA Migration Aid.
- An AMODE attribute allows specification of the entry point of the addressing mode (24-bit, 31-bit, or any addresses that are not sensitive to addressing mode) to be associated with a control section.
- An RMODE attribute allows specification of the residence mode (in the 24-bit addressable range or anywhere) to be associated with a control section.
- New channel command word instructions: CCW1 (format 1) allows 31-bit data addresses; CCW0 (format 0) allows 24-bit data addresses.
- New machine instructions for the IBM 308X models operating in System/370 Extended Architecture mode; in addition, the System/370 set of machine instructions has been expanded. A changed installation option allows users to specify whether the System/370, Extended Architecture, or Universal (all inclusive) instruction set will be used for assemblies.
- Three new instruction types are included for the Extended Architecture object code: E, RRE, and SSE.
- An underscore character is allowed in ordinary symbols.
- Operation is now supported in the CMS (Conversational Monitor System) environment of VM/SP and VM/XA Migration Aid.

WHO THIS MANUAL IS FOR

This manual is for application programmers coding in the Assembler H language. It is intended to help you assemble, link-edit, and execute your program. It describes assembler options, how to invoke the assembler, assembler listing and output, assembler data sets, error diagnostic facilities, sample programs, programming techniques and considerations, messages, and storage estimates.

HOW TO USE THIS MANUAL

To use this manual, you should be familiar with the basic concepts and facilities of your operating system as described in OS/VS1 Planning and Use Guide, GC24-5090; OS/VS2 MVS Overview, GC28-0984; MVS/Extended Architecture Overview, GC28-1146; or VM/SP Introduction, GC19-6200. You should also have a good understanding of the assembler language as described in Assembler H Version 2 Application Programming: Language Reference, GC26-4037, and, if running under MVS/XA, you should also understand the concepts described in MVS/Extended Architecture System Programming Library: 31-Bit Addressing, GC28-1158.

And, because this is a reference manual, you should use the index or the table of contents to find the subject in which you are interested.

MAJOR TOPICS

The manual is divided into the following major topics:

Part 1. Common Information

"Chapter 1. Introduction" describes the organization of this manual, the purpose of the assembler, and system requirements.

"Chapter 2. Using the Assembler Listing" describes each field of the assembler listing.

"Chapter 3. Using the Assembler Diagnostic Facilities" describes the purpose and format of error messages, MNOTEs, and the MHELP macro trace facility.

Part 2. OS/VS Information

"Chapter 4. Using the Assembler" reviews the concepts of job, job step, and job control language; describes assembler input and output; tells how the operating system handles your program; describes the assembler options, the data sets used by the assembler, the number of channel programs, and return codes; and the job control language cataloged procedures supplied by IBM. The cataloged procedures can be used to assemble, link-edit or load, and execute an assembler program.

"Chapter 5. Programming Considerations" discusses various topics, such as standard entry and exit procedures for problem programs and how to invoke the assembler dynamically.

"Chapter 6. Calculating Storage Requirements" describes the priorities and use of main storage by Assembler H during an assembly.

Part 3. CMS Information

"Chapter 7. Assembler Language Programming under CMS" describes how to assemble and execute your program, how to choose and specify the options you need, and how to interpret the listing and diagnostic messages issued by the assembler.

"Chapter 8. Programming Considerations" discusses various topics, such as standard entry and exit procedures for problem programs.

"Appendix A. Sample Program" has a program example that demonstrates many of the assembler language features.

"Appendix B. Sample Macro Trace and Dump (MHELP)" lists the operation, name, and operand entries related to macro calls.

"Appendix C. Object Deck Output" describes the object module output format.

"Appendix D. Assembler H Messages" describes the Assembler H error diagnostic messages and abnormal termination messages.

"Glossary" defines the terms used in this manual.

ASSEMBLER H PUBLICATIONS

Other publications in the Assembler H library are:

Assembler H Version 2: General Information, GC26-4035, contains a brief description of Assembler H and compares Version 2, Release 1.0, features with those of Version 1, Release 5.0. Comparisons are also made between Assembler H and VS Assembler.

Assembler H Version 2: Installation, SC26-4030, contains information necessary to install the assembler program.

Assembler H Version 2 Application Programming: Language Reference, GC26-4037, describes the basic assembler language functions and specifications that are available with Assembler H.

Assembler H Version 2: Logic, LY26-3908, describes the design logic and functional characteristics of Assembler H.

Assembler Coding Form, GX28-6509, provides the means for programmers to structure their code in the proper columns.

RELATED PUBLICATIONS

The following publications provide definitive information about machine instructions:

IBM System/370 Principles of Operation, GA22-7000

IBM 4300 Processors Principles of Operation for ECPS: VSE Mode, GA22-7070

For quick reference, see:

IBM System/370 Reference Summary, GX20-1850

For information about IBCOM requirements, see:

VS FORTRAN Application Programming: Library Reference, SC26-3989

OPERATING SYSTEM PUBLICATIONS

Within the text, references are made to the following manuals.

If You Are Running under OS/VS

OS/VS1 JCL Reference, GC24-5099

OS/VS2 MVS JCL Reference, GC28-0692

MVS/Extended Architecture JCL, GC28-1148

OS/VS Linkage Editor and Loader, GC26-3813

MVS/Extended Architecture Linkage Editor and Loader, GC26-4011

OS/VS1 Supervisor Services and Macro Instructions, GC24-5103

OS/VS2 MVS Supervisor Services and Macro Instructions, GC28-0683

MVS/Extended Architecture System Programming Library: Supervisor Services and Macro Instructions, GC28-1154

OS/VS1 Utilities, GC26-3901

OS/VS2 MVS Utilities, GC26-3902

MVS/Extended Architecture Utilities, GC26-4018

MVS/Extended Architecture Conversion Notebook, GC28-1143

MVS/Extended Architecture System Programming Library: 31-Bit Addressing, GC28-1158

If You Are Running under CMS

VM/System Product System Messages and Codes, SC19-6204

VM/System Product CMS Command and Macro Reference, SC19-6209

VM/System Product CP Command Reference for General Users, SC19-6211

CMS manuals will be distributed as part of the VM/Extended Architecture Migration Aid library.

CONTENTS

FIGURES

PART 1. COMMON INFORMATION

CHAPTER 1. INTRODUCTION

The three chapters in this part of the manual contain information common to all the operating systems under which Assembler H executes.

This chapter describes the purpose of the IBM Assembler H Version 2, Program Product 5668-962 (hereinafter referred to as Assembler H or, simply, the assembler), and its system requirements. "Chapter 6. Calculating Storage Requirements" on page 50 and "Assembler Data Sets and Storage Requirements" on page 68 have information on the amount of main and auxiliary storage required by Assembler H for OS/VS and CMS, respectively.

"Chapter 4. Using the Assembler" on page 18 and "Chapter 7. Assembler Language Programming under CMS" on page 58 describe

- Assembler input and output
- How the operating system handles your program
- How to assemble your program
- Assembler options

for OS/VS and CMS, respectively.

PURPOSE OF THE ASSEMBLER

The purpose of Assembler H is to translate programs written in assembler language into object code.

SYSTEM REQUIREMENTS

In addition to its ability to execute under OS/VS, Assembler H can be used in the CMS environment of VM/System Product (VM/SP) and VM/Extended Architecture (VM/XA) Migration Aid. Thus, Assembler H Version 2 executes under the following operating systems:

MVS/XA
OS/VS2 MVS 3.8
OS/VS1 Release 7
MVS/SP V1
VM/XA Migration Aid
VM/SP

Note: Assembler H Version 2 cannot be used with the OS/MFT, OS/MVT, or OS/VS2 SVS operating systems.

Assembler H supports the new operation codes available with the Extended Architecture mode processor, VM/XA, and bimodal addressing of MVS/XA. It is required for installation and service of MVS/SP-JES2 Version 2 and MVS/SP-JES3 Version 2, for installation of Data Facility Product, and for installation of service and modifications to VM/XA Migration Aid.

Programs written using Assembler H can be assembled, including use of the new Extended Architecture mode processor machine instructions, on all IBM System/370, 303x, 308x, and 43xx processors supported by the above operating systems. You may require the MVS/XA macro library to assemble programs that will be executed on MVS/XA, depending on macro usage.

Execution of programs assembled with Version 2 containing Extended Architecture machine instructions can only be accomplished on processors operating in Extended Architecture mode under MVS/XA or an MVS/XA guest operating system under VM/XA Migration Aid.

Virtual Storage: Assembler H Version 2, Release 1.0, requires a minimum of 200K bytes of main storage.

Auxiliary Storage Space: Auxiliary storage space is required for the following data sets:

- System input.
- Macro instruction library—either system or private or both.
- An intermediate work file, which must be a direct-access device (3330/3333, 3340/3344, 3350, 3375, or 3380). Under VM/XA Migration Aid, the intermediate work file must be formatted as a CMS minidisk. Under VM/SP, the intermediate work file, which must be a direct-access device (3310, 3370, or one of the devices mentioned above), must also be formatted as a CMS minidisk.
- Print output.

COMPATIBILITY

The language supported by Assembler H Version 2 has functional extensions to the language supported by VS Assembler and OS Assembler H 5734-AS1 Release 5.0. Programs written for VS Assembler and OS Assembler H Release 5.0 that were successfully assembled with no warning or diagnostic messages can be assembled with Version 2, with the minor exceptions described in Appendix E, "Assembler H Version 2 Incompatibility with OS/VS Assembler."

CHAPTER 2. USING THE ASSEMBLER LISTING

This chapter tells you how to interpret the printed listing produced by the assembler. The listing is obtained only if the option LIST is in effect. Parts of the listing can be suppressed by using other options; for information on the listing options, refer to "Assembler Options for OS/VS" on page 24 or "Assembler Options for CMS" on page 63.

The Assembler H listing consists of up to five sections, ordered as follows:

- External symbol dictionary (ESD)
- Source and object program
- Relocation dictionary (RLD)
- Symbol and literal cross-reference
- Diagnostic cross-reference and assembler summary

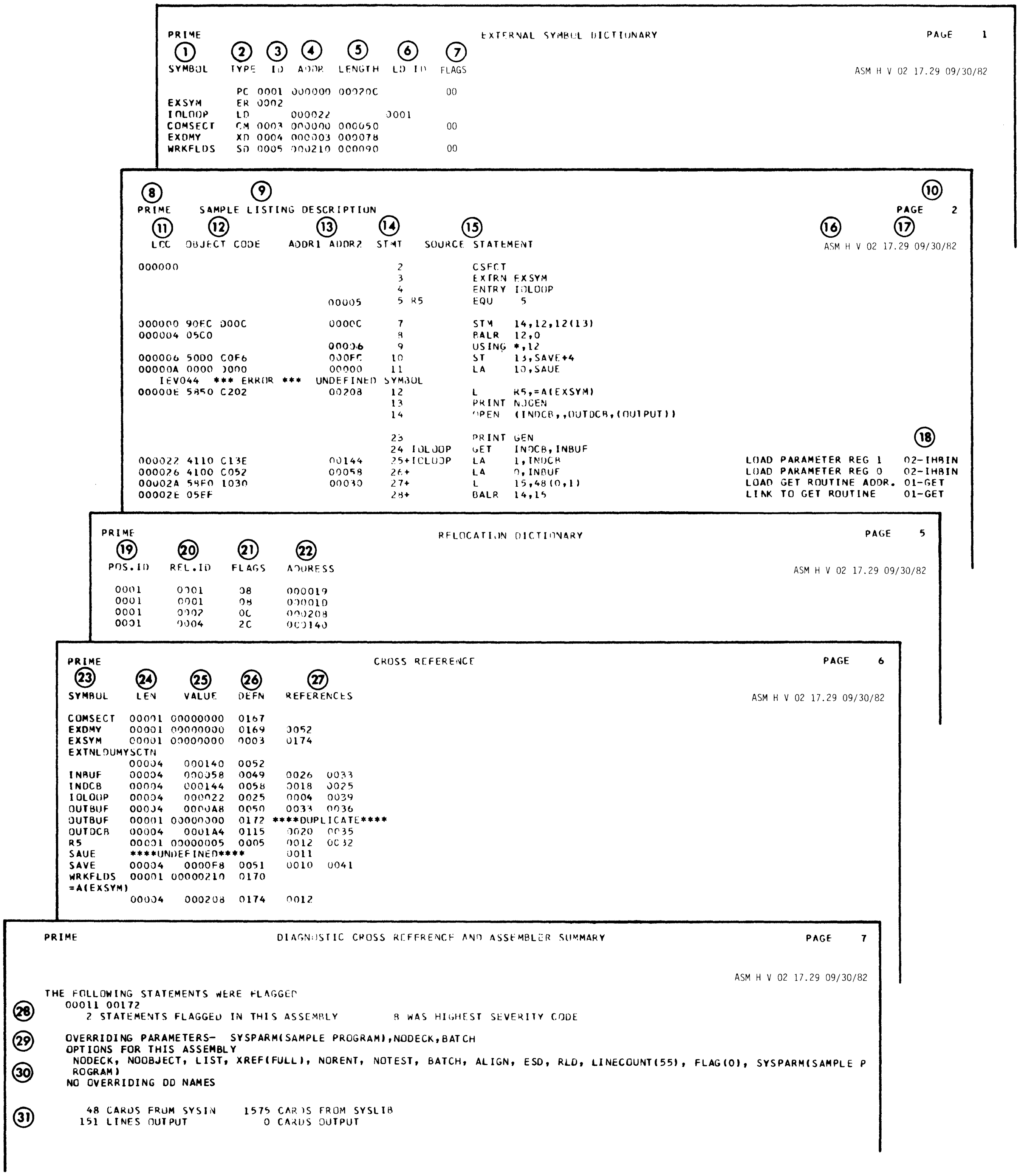
Figure 1 on page 5 shows each section of the listing. Each item marked with a circled number is explained in the following text. (See "Glossary" for definitions of terms.)

EXTERNAL SYMBOL DICTIONARY (ESD)

This section of the listing contains the external symbol dictionary information passed to the linkage editor or loader in the object module.

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

The ESD entries describe the control sections, external references, and entry points in the assembled program. There are seven types of ESD entries (SD, LD, ER, PC, CM, XD, and WX). They are shown in Figure 2 on page 6 with their associated fields. The numbers refer to the corresponding headings in the sample listing shown in Figure 1 on page 5. For each of the different types of ESD entries, the Xs indicate which of the fields will have values.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

```
PRIME                                       EXTERNAL SYMBOL DICTIONARY                              PAGE    1
SYMBOL    TYPE  ID  ADDR   LENGTH  LD ID  FLAGS                                    ASM H V 02 17.29 09/30/82

          PC 0001 000000 00020C           00
EXSYM     ER 0002
IOLOOP    LD      000022          0001
COMSECT   CM 0003 000000 000050           00
EXDMY     XD 0004 000003 000078
WRKFLDS   SD 0005 000210 000090           00
```

```
PRIME    SAMPLE LISTING DESCRIPTION                                                         PAGE    2
  LOC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM H V 02 17.29 09/30/82

000000                                 2          CSECT
                                       3          EXTRN EXSYM
                                       4          ENTRY IOLOOP
                             00005     5 R5       EQU    5

000000 90EC D00C             0000C     7          STM   14,12,12(13)
000004 05C0                            8          BALR  12,0
                             00006     9          USING *,12
000006 50D0 C0F6             000FC    10          ST    13,SAVE+4
00000A 0000 0000             00000    11          LA    10,SAUE
   IEV044  *** ERROR ***  UNDEFINED SYMBOL
00000E 5850 C202             00208    12          L     R5,=A(EXSYM)
                                      13          PRINT NOGEN
                                      14          OPEN  (INDCB,,OUTDCB,(OUTPUT))

                                      23          PRINT GEN
                                      24 IOLOOP   GET   INDCB,INBUF
000022 4110 C13E             00144    25+IOLOOP   LA    1,INDCB                 LOAD PARAMETER REG 1   02-IHBIN
000026 4100 C052             00058    26+         LA    0,INBUF                 LOAD PARAMETER REG 0   02-IHBIN
00002A 58F0 1030             00030    27+         L     15,48(0,1)              LOAD GET ROUTINE ADDR. 01-GET
00002E 05EF                           28+         BALR  14,15                   LINK TO GET ROUTINE    01-GET
```

```
PRIME                                   RELOCATION DICTIONARY                              PAGE    5
 POS.ID   REL.ID   FLAGS   ADDRESS                                        ASM H V 02 17.29 09/30/82

  0001     0001     08     000019
  0001     0001     08     00001D
  0001     0002     0C     000208
  0001     0004     2C     000140
```

```
PRIME                                CROSS REFERENCE                                PAGE    6
SYMBOL     LEN    VALUE   DEFN   REFERENCES                            ASM H V 02 17.29 09/30/82

COMSECT   00001 00000000  0167
EXDMY     00001 00000000  0169   0052
EXSYM     00001 00000000  0003   0174
EXTNLDUMYSCTN
          00004   000140  0052
INBUF     00004   000058  0049   0026  0033
INDCB     00004   000144  0058   0018  0025
IOLOOP    00004   000022  0025   0004  0039
OUTBUF    00004   0000A8  0050   0033  0036
OUTBUF    00001 00000000  0172 ****DUPLICATE****
OUTDCB    00004   0001A4  0115   0020  0035
R5        00001 00000005  0005   0012  0032
SAUE      ****UNDEFINED****      0011
SAVE      00004   0000F8  0051   0010  0041
WRKFLDS   00001 00000210  0170
=A(EXSYM)
          00004   000208  0174   0012
```

```
PRIME                        DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                  PAGE    7

                                                                                ASM H V 02 17.29 09/30/82
THE FOLLOWING STATEMENTS WERE FLAGGED
    00011 00172
       2 STATEMENTS FLAGGED IN THIS ASSEMBLY          8 WAS HIGHEST SEVERITY CODE

    OVERRIDING PARAMETERS-  SYSPARM(SAMPLE PROGRAM),NODECK,BATCH
    OPTIONS FOR THIS ASSEMBLY
     NODECK, NOOBJECT, LIST, XREF(FULL), NORENT, NOTEST, BATCH, ALIGN, ESD, RLD, LINECOUNT(55), FLAG(0), SYSPARM(SAMPLE P
     ROGRAM)
    NO OVERRIDING DD NAMES

       48 CARDS FROM SYSIN      1575 CARDS FROM SYSLIB
      151 LINES OUTPUT             0 CARDS OUTPUT
```

Figure 1. Assembler H Listing

(1) SYMBOL	(2) TYPE	(3) ID	(4) ADDR	(5) LENGTH	(6) LD ID	(7) FLAGS
X	SD	X	X	X	-	X
X	LD	-	X	-	X	-
X	ER	X	-	-	-	-
-	PC	X	X	X	-	X
X	CM	X	X	X	-	X
X	XD	X	X	X	-	-
X	WX	X	-	-	-	-

Figure 2. Types of ESD Entries

(1) The name of every external dummy section, control section, entry point, and external symbol.

(2) The type designator for the entry, as shown in the table. The type designators are defined as:

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

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

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

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

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

XD External dummy section. The symbol appeared in the name field of a DXD statement or a Q-type address constant. (The external dummy section is also called a pseudo register in the appropriate Linkage Editor and Loader.)

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

(3) The external symbol dictionary identification number (ESDID). The number is a unique 4-digit hexadecimal number identifying the entry. It is used in combination with the LD entry of the ESD and in the relocation dictionary for referencing the ESD.

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

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

(6) For an LD-type entry, the ESDID of the control section in which the symbol was defined.

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

```
Bit 5:     0  = RMODE is 24
           1  = RMODE is ANY
Bits 6-7:  00 = AMODE is 24
           01 = AMODE is 24
           10 = AMODE is 31
           11 = AMODE is ANY
```

SOURCE AND OBJECT PROGRAM

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 punching errors, and, together with the diagnostics and statistics, to locate and correct errors detected by the assembler. By using this section with the 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.

(8) The 1- to 8-character deck identification, if any. It is obtained from the name field of the first named TITLE statement. The assembler prints the deck identification and date (item 16) on every page of the listing.

(9) The information taken from the operand field of a TITLE statement.

(10) The listing page number.

(11) The assembled address (in hexadecimal notation) of the object code.

- For ORG statements, the location-counter value before the ORG is placed in the location column and the location counter value after the ORG is placed in the object code field.
- If the END statement contains an operand, the operand value (transfer address) appears in the location field (LOC).
- In the case of LOCTR, COM, CSECT, 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 a USING statement, the location field contains the value of the first operand. It is 4 bytes long.
- For LTORG statements, the location field contains the location assigned to the literal pool.
- For an EQU statement, the location field contains the value assigned. It is 4 bytes long.

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

(13) 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 SS instruction.

The field headed ADDR2 contains the effective address of the last operand of any instruction referencing storage.

Both address fields contain 6 digits; however, if the high-order digit is a 0, it is not printed.

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

(15) 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 always printed.

- Macro definitions obtained from SYSLIB are not listed, unless the macro definition is included in the source program by means of a COPY statement.

- The statements generated as the result of a macro call follow the macro call in the listing, unless PRINT NOGEN is in effect.

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

- All error diagnostic messages appear in line except those suppressed by the FLAG option. "Chapter 3. Using the Assembler Diagnostic Facilities" 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 the following example:

```
Source Statements:    &C   SETC        'ABCDEFGHIJK'
                      &C   LA          1,4
Generated Statement:  ABCDEFGHIJK LA   1,4
```

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.

(16) The version identifier of Assembler H.

(17) The current date (date run is made).

(18) 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.

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), 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.

RELOCATION DICTIONARY (RLD)

This section of the listing contains the relocation dictionary information passed to the linkage editor 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.

(19) 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.

(20) The external symbol dictionary ID number assigned to the ESD entry for the control section in which the referenced symbol is defined.

(21) The 2-digit hexadecimal number represented by the characters in this field is interpreted as follows:

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

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

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

(22) The assembled address of the field where the address constant is stored.

SYMBOL AND LITERAL CROSS-REFERENCE

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

(23) The symbols or literals.

(24) The length, in bytes (in decimal notation), of the field represented by the symbol. The length of a literal is always 1.

(25) Either the address that the symbol or literal represents, or a value to which the symbol is equated. The value is 3 bytes long, except for the following, which are 4 bytes long: CSECT, DSECT, START, COM, DXD, EQU, LOCTR, EXTRN, WXTRN, and a duplicate symbol.

(26) The number of the statement in which the symbol or literal was defined.

(27) The statement numbers of statements in which the symbol or literal appears as an operand. In the case of a duplicate symbol or literal, the assembler fills this column with the message:

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

The following notes apply to the cross-reference section:

- The statement numbers in fields 26 and 27 may have 4, 5, or 6 print positions. The number of print positions for the statement number will be chosen based on the highest statement number assigned for the assembly. For example, if 21056 is the highest statement number used in an assembly, all statement numbers in the cross-reference listing will have 5 print positions.

- Symbols appearing in V-type address constants do not appear in the cross-reference listing.

- 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.

- A PRINT OFF listing control instruction does not affect the production of the cross-reference section of the listing.

- In the case of an undefined symbol, the assembler fills fields 24, 25, and 26 with the message:

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

DIAGNOSTIC CROSS-REFERENCE AND ASSEMBLER SUMMARY

The diagnostic messages issued by the assembler are fully documented in Appendix D, "Assembler H Messages."

(28) The statement number of each statement flagged with an error message or MNOTE appears in this list. The number of statements flagged and the highest nonzero severity code encountered are also printed. The highest severity code is equal to the assembler return code.

If no errors are encountered, the following statement is printed:

```
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
```

See "Chapter 3. Using the Assembler Diagnostic Facilities" for a complete discussion of how error messages and MNOTEs are handled.

(29) A list of the options in effect for this assembly is printed. The options specified in the PARM field to override the assembler default options are also printed.

(30) If the assembler has been called by a problem program (see "Invoking the Assembler Dynamically" on page 48) 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
```

(31) The assembler prints the number of records read from SYSIN and SYSLIB and the number of records written on SYSPUNCH. The assembler also prints the number of lines written on SYSPRINT. This is a count of the actual number of 121-byte records generated by the assembler; it may be less than the total number of printed and blank lines appearing in the listing if the SPACE n assembler instruction is used. For a SPACE n that does not cause an eject, the assembler inserts n blank lines in the listing by generating n/3 blank 121-byte records, rounded to the next lower integer if a fraction results. For example, for a SPACE 2, no blank records are generated. The assembler does not generate a blank record to force a page eject.

CHAPTER 3. USING THE ASSEMBLER DIAGNOSTIC FACILITIES

The diagnostic facilities for Assembler H include diagnostic messages for assembly errors, diagnostic or explanatory messages issued by the source program or by macro definitions (MNOTEs), a macro trace and dump facility (MHELP), and messages and dumps issued by the assembler in case it terminates abnormally.

This chapter briefly describes these facilities. The assembly error diagnostic messages and abnormal assembly termination messages are described in detail in Appendix D, "Assembler H Messages."

ASSEMBLY ERROR DIAGNOSTIC MESSAGES

Assembler H prints most error messages in the listing immediately following the statement in error. It also prints the total number of flagged statements and their line numbers in the diagnostic cross-reference section at the end of the listing.

The messages do not follow the statement in error when:

- Errors are detected during editing of macro definitions read from a library. A message for such an error appears after the first call in the source program to that macro definition. You can, however, bring the macro definition into the source program with a COPY statement. The editing error messages will then be attached to the statements in error.

- Errors are detected by the lookahead function of the assembler. (Lookahead scans, for attribute references, statements after the one being assembled.) Messages for these errors appear after the statements in which they occur. The messages may also appear at the point at which lookahead was called.

- Errors are detected on conditional assembler statements during macro generation or MHELP testing. Such a message follows the most recently generated statement or MHELP output statement.

A typical error diagnostic message is:

```
  IEV057   ***ERROR***  UNDEFINED OPERATION CODE—xxxxx
```

The term ***ERROR*** is part of the message if the severity code is 8 or greater. The term **WARNING** is part of the message if the severity code is 0 or 4.

A copy of a segment of the statement in error, represented above by xxxxx, is appended to the end of many messages. Normally this segment, which can be up to 16 bytes long, begins at the bad character or term. For some errors, however, the segment may begin after the bad character or term. The segment may include part of the remarks field.

If a diagnostic message follows a statement generated by a macro definition, the following items may be appended to the error message:

- The number of the model statement in which the error occurred, or the first five characters of the macro name.

- The SET symbol, parameter number, or value string associated with the error.

Note: References to macro parameters are by number (such as PARAM008) instead of by name. The first seven numbers are always assigned for the standard system parameters as follows:

```
  PARAM000 =  &SYSNDX
  PARAM001 =  &SYSECT
  PARAM002 =  &SYSLOC
  PARAM003 =  &SYSTIME
  PARAM004 =  &SYSDATE
  PARAM005 =  &SYSPARM
  PARAM006 =  Name Field Parameter
```

Then the keyword parameters are numbered in the order defined in the macro definition, followed by positional parameters. When there are no keyword parameters in the macro definition, PARAM007 refers to the first positional parameter.

If a diagnostic message follows a conditional assembler statement in the source program, the following items will be appended to the error message:

- The word "OPENC"
- The SET symbol or value string associated with the error

Several messages may be issued for a single statement or even for a single error within a statement. This happens because each statement is usually evaluated on more than one level (for example, term level, expression level, and operand level) or by more than one phase of the assembler. Each level or phase can diagnose errors; therefore, most or all of the errors in the statement are flagged. Occasionally, duplicate error messages may occur. This is a normal result of the error detection process.

Figure 3 on page 14 is an example of Assembler H handling of error messages.

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                              ASM H V 02 11.51 09/30/82

                                     1 ***********************************************************************
                                     2 *                        SAMPLE ERROR DIAGNOSTIC MESSAGES                    *
                                     3 *          IN SOURCE PROGRAM (OPEN CODE) AND GENERATED BY MACRO CALLS        *
                                     4 ***********************************************************************

000000                               6 A          CSECT
000000 0000 0000          00000      7            STM   14,U2,12(13(
   IEV044  *** ERROR ***  UNDEFINED SYMBOL
   IEV029  *** ERROR ***  INCORRECT REGISTER SPECIFICATION
   IEV179  *** ERROR ***  DELIMITER ERROR, EXPECT RIGHT PARENTHESIS
000004 05C0                          8            BALR  12,0
                          00006      9            USING *,12
000006 0000 0000          00000     10            ST    13,SAVE+4
   IEV044  *** ERROR ***  UNDEFINED SYMBOL
                                    11            OPEN  (CRDIN,(INPUT),CRDOUT,(OUTPUT)
   IEV088  *** ERROR ***  UNBALANCED PARENTHESES IN MACRO CALL OPERAND  --  OPENC/(CRDIN,(IN
00000A 0700                         12+           CNOP  0,4                                                    01-OPEN
00000C 4510 C00F          00014     13+           BAL   1,*+8                            LOAD REG1 W/LIST ADDR. 01-OPEN
000010 00000000                     14+           DC    A(0)                             OPT BYTE AND DCB ADDR. 01-OPEN
000014 0000 0000          00000     15+           ST    CRDIN,(INPUT),CRDOUT,(OUTPUT,0(1,0)                     X01-OPEN
                                     +                                                   STORE INTO LIST
   IEV029  *** ERROR ***  INCORRECT REGISTER SPECIFICATION
   IEV044  *** ERROR ***  UNDEFINED SYMBOL
   IEV177  *** ERROR ***  DELIMITER ERROR, EXPECT BLANK OR LEFT PARENTHESIS
000018 9280 1000     00000          16+           MVI   0(1),128                         MOVE IN OPTION BYTE    01-OPEN
00001C 0A13                         17+           SVC   19                               ISSUE OPEN SVC         01-OPEN

                                    19 **********************************************************************
                                    20 *     EDITING AND GENERATION ERRORS AND MNOTES FROM A LIBRARY MACRO  *
                                    21 **********************************************************************

                                    23            LOADR REG1=10,REG2=8,CHEROKEE,CHAMP
   IEV136  *** ERROR ***  ILLEGAL LOGICAL/RELATIONAL OPERATOR  --  MACRO - LOADR
   IEV083  *** ERROR ***  ARITHMETIC EXPRESSION CONTAINS ILLEGAL DELIMITER OR ENDS PREMATURELY  --  MACRO - LOADR
00001E 58A0 C02A          00030     24+           L     10,CHEROKEE                                              01-LOADR

                                    26            LOADR REG1=25,REG2=8,CHEROKEE,SWIFT
000022 0000 0000          00000     27+           L     25,CHEROKEE                                              01-LOADR
   IEV029  *** ERROR ***  INCORRECT REGISTER SPECIFICATION

                                    29            LOADR REG2=10,CHAMP,SWIFT
000026 5800 C02E          00034     30+           L     0,CHAMP                                                  01-LOADR


                                     6 **********************************************************************
                                     7 *    SAMPLE MACRO DEFINITION RERUN WITH EDITING ERRORS CORRECTED     *
                                     8 **********************************************************************

                                    10            MACRO
                                    11 &NAME      LOADR &REG1=,&REG2=,&OP1,&OP2
                                    12 &R(1)      SETA  &REG1,&REG2
                                    13            AIF   (T'&REG1 EQ 'O').ERR
                                    14            L     &R(1),&OP1
                                    15            L     &R(2),&OP2
                                    16            MEXIT
                                    17 .ERR       MNOTE 36,'YOU LEFT OUT THE FIRST REGISTER'
                                    18            MEND

                                    20 **********************************************************************
                                    21 *          SAMPLE MACRO CALLS WITH GENERATION ERRORS AND MNOTES      *
                                    22 **********************************************************************

                                    24            LOADR REG1=10,REG2=8,CHEROKEE,CHAMP
00000C 58A0 C004          00004     25+           L     10,CHEROKEE                                              01-00014
000010 5880 C008          00008     26+           L     8,CHAMP                                                  01-00015

                                    28            LOADR REG1=25,REG2=8,CHEROKEE,&SWIFT
   IEV003  *** ERROR ***  UNDECLARED VARIABLE SYMBOL. DEFAULT=0, NULL, OR TYPE=U  --  OPENC/SWIFT
000014 0000 0000          00000     29+           L     25,CHEROKEE                                              01-00014
   IEV029  *** ERROR ***  INCORRECT REGISTER SPECIFICATION
000018 0000 0000          00000     30+           L     8,                                                       01-00015
   IEV074  *** ERROR ***  ILLEGAL SYNTAX IN EXPRESSION

                                    32            LOADR REG2=8,CHAMP,SWIFT
   IEV254  *** MNOTE ***            33+      36,YOU LEFT OUT THE FIRST REGISTER                                  01-00017
                                    34            END
```

Figure 3. Sample Error Diagnostic Messages

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:

```
  IEV254 ***MNOTE***  severity code, message
```

Unless it has a severity code of * or the severity code is omitted, the statement number or the MNOTE is listed in the diagnostic cross-reference.

SUPPRESSION OF ERROR MESSAGES AND MNOTE STATEMENTS

Optionally, error messages and MNOTE statements below a specified severity level can be suppressed by specifying the assembler option 'FLAG(n)' (where "n" is the selected severity level when the assembler is invoked).

ABNORMAL ASSEMBLY TERMINATION

Whenever the assembly cannot be completed, Assembler H provides a message and, in some cases, a specially formatted dump for diagnostic information. This may indicate an assembler malfunction or it may 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 D, "Assembler H Messages" on page 100 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.

Assembler H Version 2: Logic contains a complete explanation of the format and contents of the abnormal termination dump.

MACRO TRACE FACILITY (MHELP)

The MHELP instruction controls a set of trace and dump facilities. Options are selected by 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 continuously until superseded by another MHELP statement. Appendix B, "Sample Macro Trace and Dump (MHELP)" is a sample MHELP trace and dump.

Macro Call Trace

MHELP B'1' or MHELP 1: This option provides a one-line trace for each macro call, giving the name of the called macro, its nested depth, and its &SYSNDX (total number of macro calls) value.

Note: This trace is provided upon entry into the macro. No trace is provided if error conditions prevent entry into the macro.

Macro Branch Trace

MHELP B'10', or MHELP 2: This option provides a one-line trace for each AGO and true AIF conditional-assembly statement within a macro. It gives the model-statement numbers of the "branched from" and "branched to" statements, and the name of the macro in which the branch occurs. This trace option is suppressed for library macros.

Macro Entry Dump

MHELP B'10000', or MHELP 16: This option dumps parameter values from the macro dictionary when the macro is called.

Macro Exit Dump

MHELP B'1000', or MHELP 8: This option dumps SET symbol values from the macro dictionary upon encountering a MEND or MEXIT statement.

Macro AIF Dump

MHELP B'100', or MHELP 4: This option dumps SET symbol values from the macro dictionary immediately before each AIF statement that is encountered.

Global Suppression

MHELP B'100000', or MHELP 32: This option suppresses global SET symbols in the two preceding options, MHELP 4 and MHELP 8.

MHELP Suppression

MHELP B'10000000', or MHELP 128: This option suppresses all currently active MHELP options.

Combining Options

Multiple options can be obtained by combining the option codes in one MHELP operand. For example, call and branch traces can be invoked by MHELP B'11', MHELP 2+1, or MHELP 3.

MHELP Control on &SYSNDX

The MHELP operand field is actually mapped into a fullword. Previously defined MHELP codes correspond to the fourth byte of this fullword.

&SYSNDX control is turned on by any bit in the third byte (operand values 256 to 65535 inclusive). Then, when &SYSNDX (total number of macro calls) exceeds the value of the fullword that contains the MHELP operand value, control is forced to stay at the open-code level, by in effect making every statement in a macro behave like an MEXIT. Open-code macro calls are honored, but with an immediate exit back to open code.

Examples:

```
MHELP 256       Limit &SYSNDX to 256.
MHELP 1         Trace macro calls.
MHELP 256+1     Trace calls and limit &SYSNDX to 257.
MHELP 65536     No effect.  No bits in bytes 3,4.
MHELP 65792     Limit &SYSNDX to 65792.
```

When the value of &SYSNDX reaches its limit, the message "ACTR EXCEEDED—&SYSNDX" is issued.

PART 2. OS/VS INFORMATION

CHAPTER 4. USING THE ASSEMBLER

This chapter describes assembler input and output; tells how the operating system handles your program; reviews the concepts of job, job step, and job control language; shows you how to invoke the assembler for simple jobs (using cataloged procedures); and lists the job control statements that make up the four assembler cataloged procedures. In addition, it describes the assembly-time options available to the assembler language programmer; data sets used by the assembler; and number of channel programs, return codes, and cataloged procedures of job control language supplied by IBM to simplify assembling, link-editing or loading, and execution of assembler language programs. The job control language is described in detail in the appropriate _JCL Reference_.

INPUT

As input, the assembler accepts a program written in the assembler language as defined in _Assembler H Version 2 Application Programming: Language Reference_. This program is referred to as a source module. Some statements in the source module (macro or COPY instructions) may cause additional input to be obtained from a macro library.

OUTPUT

The output from the assembler consists of an object module and a program listing. The object module can either be punched or included in a data set residing on a direct access device or a magnetic tape. From that data set, the object module can be read into the computer and processed by the linkage editor or the loader. See Appendix C, "Object Deck Output" 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."

HOW THE OPERATING SYSTEM HANDLES YOUR PROGRAM

Once you have coded and entered your program, it must be processed by the assembler and the linkage editor or the loader before it can be executed. Figure 4 on page 19 shows how the operating system handles your program.

The source program is read in for processing by the assembler.

The output of the assembler, the object module, is placed on auxiliary storage.

The object module is read into either the linkage editor or the loader for processing.

After processing your program, the loader gives control to it.

The linkage editor output, the load module, is placed on a load module library.

Your program, in load module format, is read into the computer for execution.

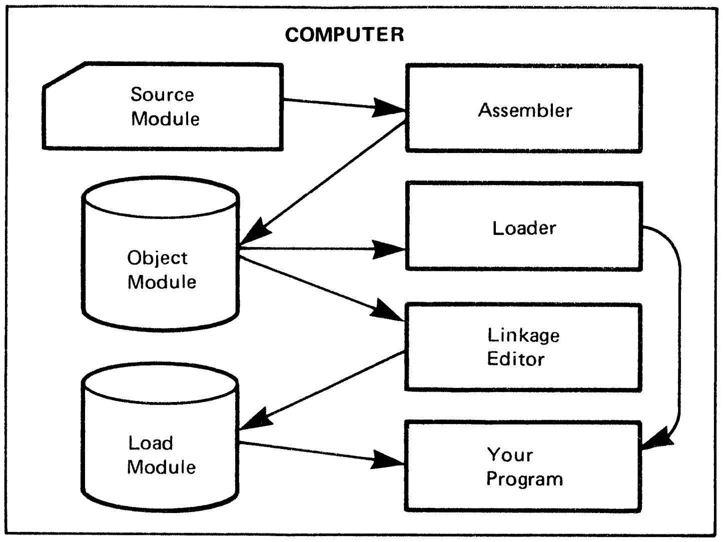


Figure 4. How the Operating System Handles Your Program

ASSEMBLER

The assembler translates your source module into an object module, the machine language equivalent of the source module. The object module, however, is not ready for execution; it must first be processed by the linkage editor or loader.

LINKAGE EDITOR

The linkage editor prepares your program for execution. The output of the linkage editor is called a load module and can be executed by the computer. The linkage editor can combine your program with other object and load modules to produce a single load module. The linkage editor stores your program in a load module library, a collection of data sets on a direct access device. These load modules can be read into the computer and given control. The load module library may be either permanent, so that you can execute your program in later jobs, or temporary, so that the program is deleted at the end of your job.

EXECUTION OF YOUR PROGRAM

Once you have included your program in a permanent load module library, you can execute it any number of times without assembly and link-editing. However, if you need to change your program, you must assemble and link-edit it again. Therefore, you should not store your program in a permanent load module library until it has been tested properly. To save time during test runs, you can use a program that combines the basic functions of the linkage editor with the execution of your program. That program is the loader.

LOADER

The loader performs most of the functions of the linkage editor; in addition, it loads your program into the computer and passes control to your program. The loader cannot, however, include your program in a load module library. For a full description of the linkage editor and loader, refer to the appropriate _Linkage Editor and Loader_.

JOB CONTROL LANGUAGE

JOBS AND JOB STEPS

Each time you request a service from the operating system, you are asking it to perform a _job_. A job may consist of several _steps_, each of which usually involves the execution of one processing program under the control of the operating system's control program. For example, if you submit a job to the computer calling for assembly and linkage editing of a program, that job will be a two-step job. The concepts of jobs and job steps are shown in Figure 5.

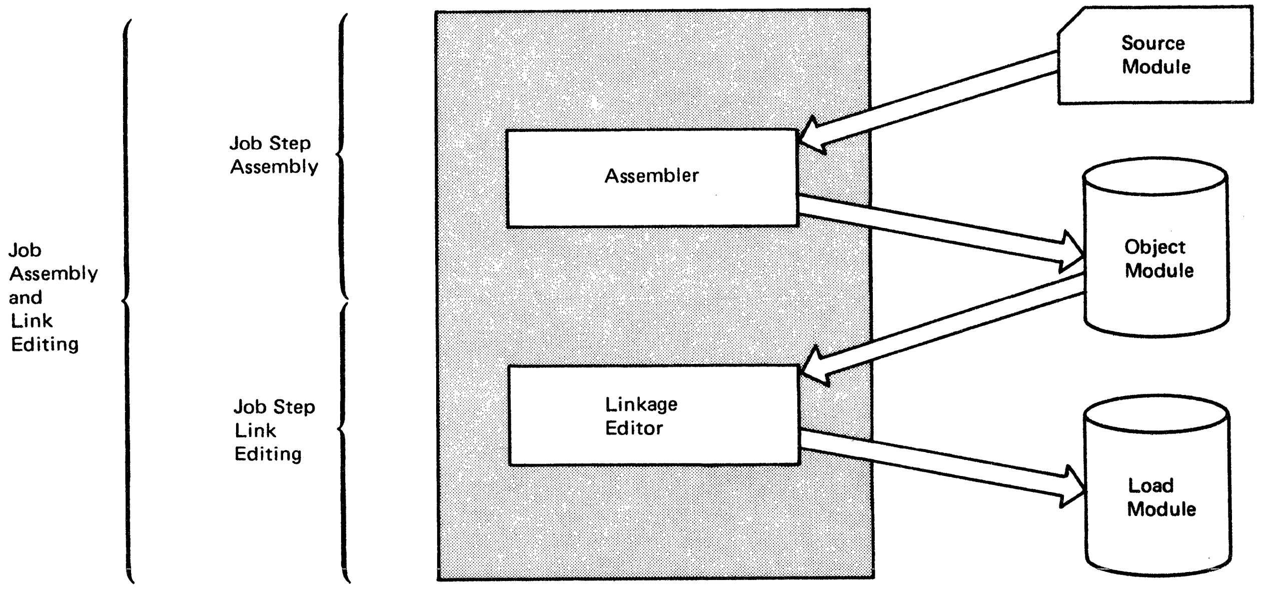


Figure 5. Jobs and Job Steps

JOB CONTROL LANGUAGE

The job control language is your way of communicating to the operating system control program what services you want used. Job control language statements are usually punched into cards and supplied in the job stream with your source module and other data needed by the job. For a detailed discussion of job control language statements, see the appropriate JCL Reference.

To save time and trouble, you can use predefined sets of JCL statements that reside in a library. Such a set of statements, called a cataloged procedure, can be included in your job by means of a single JCL statement naming the set. Figure 6 shows the concept of a cataloged procedure.

There are several cataloged procedures available for assembler jobs. They are described in the following sections.

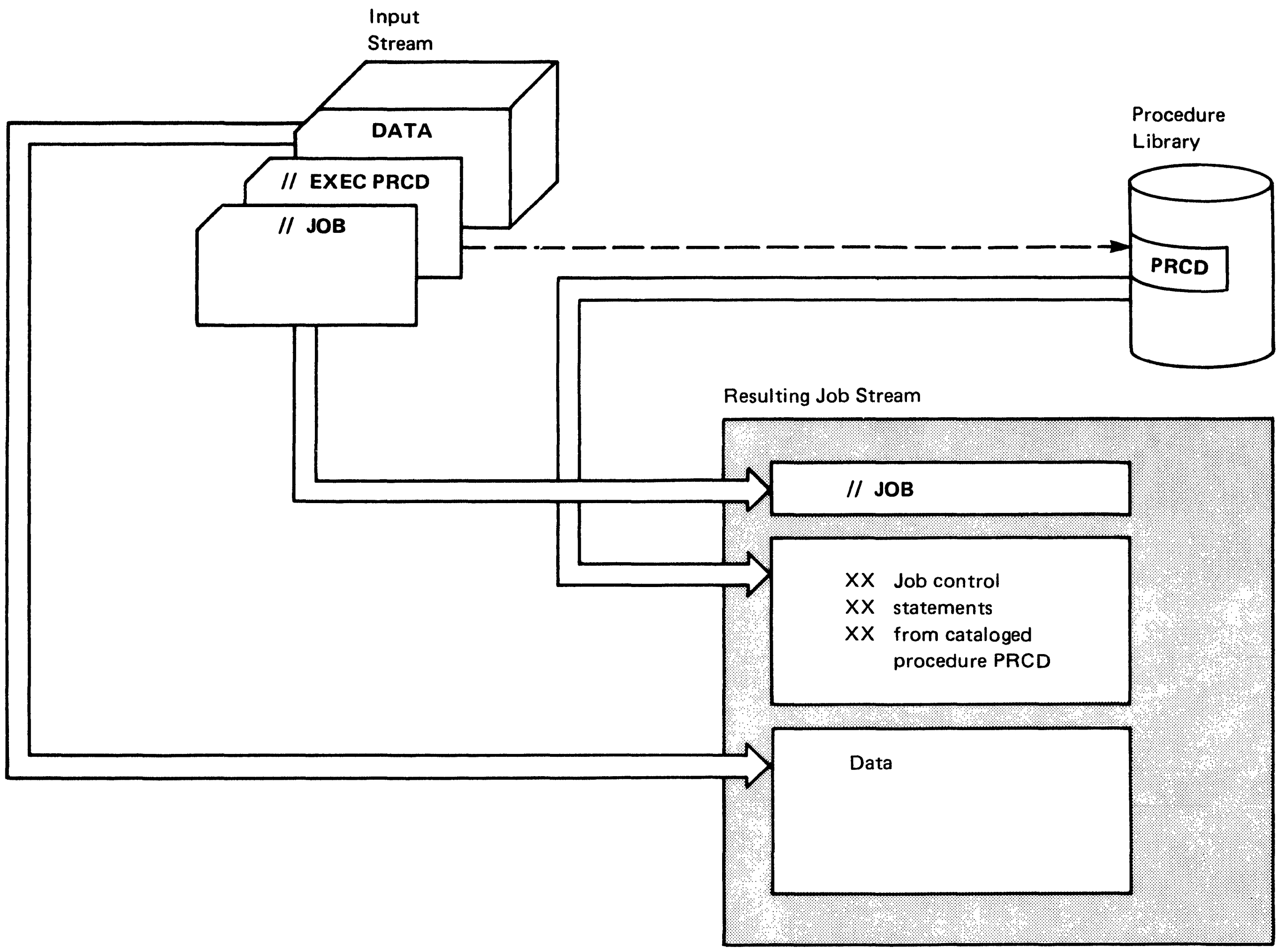


Figure 6. Cataloged Procedure Concept

JOB CONTROL STATEMENTS FOR ASSEMBLER JOBS

The following sections show you how to invoke the assembler for simple jobs, using cataloged procedures, and list the job control statements that make up the four assembler cataloged procedures.

SIMPLE ASSEMBLY AND EXECUTION

This section gives the minimum JCL statements needed for two simple assembly jobs:

- Assembly of your program to produce a listing and an object deck
- Assembly and execution of your program

Both jobs use cataloged procedures to call the assembler.

Assembly

To assemble your program, use the following JCL statements:

```
//jobname   JOB   accountno,progrname,MSGLEVEL=1    (1)
//          EXEC  ASMHC                             (2)
//SYSIN     DD    *                                 (3)

  (your source program)
```

Notes:

(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 ASMHC. As a result, a number of job control statements are included in the job from the procedure library. ASMHC is described under "Cataloged Procedure for Assembly (ASMHC)" on page 34; 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 and to produce a listing (described in "Chapter 2. Using the Assembler Listing") and an object module punched on cards (described in Appendix C, "Object Deck Output"). If you do not want any object module cards to be punched during the job, use the following statements:

```
//jobname   JOB   accountno,progrname,MSGLEVEL=1
//          EXEC  ASMHC,PARM=NODECK                 (1)
//SYSIN     DD    *

  (your source program)
```

Note:

(1) The second parameter (PARM) specifies the assembler option NODECK, which tells the assembler not to produce any punched object module. For a full discussion of assembler options, see "Assembler Options for OS/VS" on page 24.

Assembly and Execution

To run a job that both assembles and executes your program, code the following statements:

```
//jobname  JOB   accountno,progrname,MSGLEVEL=1
//         EXEC  ASMHCG                              (1)
//C.SYSIN  DD    *                                   (2)

  (your source program)

//G.SYSIN DD     *                                   (3)

  (data, if any, for your program)
```

Notes:

(1) Calls the procedure ASMHCG, containing job control statements for execution of the assembler (in procedure step C) and the loader (in step G). ASMHCG is described under "Cataloged Procedure for Assembly and Loader Execution (ASMHCG)" on page 38; an expanded job stream is shown there.

(2) Specifies that the input for procedure step C (assembly) follows immediately after this statement.

(3) Specifies that the input for step G (execution of your program under control of the loader) follows immediately after this statement.

The first step of the ASMHCG procedure executes the assembler. The assembler produces a listing, a punched object module on cards, and an object module on a direct access device. The second step causes the loader to be executed. The loader transforms the object module, which was written on a direct access device by the assembler, into a load module. In addition, the loader causes the load module (that is, your program) to be executed.

If you do not want the assembler to punch an object deck in this example, supply the following statements instead:

```
//jobname   JOB   accountno,progrname,MSGLEVEL=1
//          EXEC  ASMHCG,PARM.C=(OBJECT,NODECK)      (1)
//C.SYSIN DD   *
         .
  (your source program)
         .
//G.SYSIN  DD    *
  (data for your program)
         .
```

Note:

(1) The PARM parameter specifies the assembler options OBJECT (telling the assembler to produce an object module on the partitioned data set used as input by the loader) and NODECK for step C (assembly) of the procedure.

ASSEMBLER OPTIONS FOR OS/VS

Assembler H 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 deck or an object module. You select the options by including appropriate keywords in the PARM field of the EXEC JCL statement that invokes the assembler. There are two types of options:

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

Each of these options has a standard or default value which is used for the assembly if you do not specify an alternative value. The default values are explained under "Default Options" on page 29.

To override the option specification 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    ASMHCG,PARM.C='OBJECT,NODECK'
```

"Overriding Statements in Cataloged Procedures" contains more examples on how to specify options in a cataloged procedure.

PARM is a keyword parameter that is followed by a list of options. PARM options are coded according to the following rules:

- If you specify two or more options, the entire list must be enclosed within single quotation marks or parentheses.
- The options must be separated by commas.
- If you specify only one option and it does not include any special characters, the enclosing single quotation marks or parentheses can be omitted.
- The FLAG, LINECOUNT, SYSPARM, and XREF options must appear within single quotation marks (since they contain special characters).
- The options can be specified in any order.
- The option list must not be longer than 100 characters, including the separating commas.
- If you need to continue the PARM field onto another card, the entire PARM field must be enclosed in parentheses. However, any part of the PARM field enclosed in single quotation marks must not be continued on another card.
- If contradictory options are used (for example, LIST and NOLIST), the rightmost option (in this case, NOLIST) is used.

The following examples illustrate these rules:

,PARM=DECK	Only one option specified.
,PARM='LINECOUNT(40)'	LINECOUNT, FLAG, SYSPARM, and XREF must be surrounded by single quotation marks.
,PARM=(DECK,NOOBJECT) or ,PARM='DECK,NOOBJECT'	More than one option is specified. None of them require quotation marks.
,PARM='DECK,NOLIST,SYSPARM(PARAM)' or ,PARM=(DECK,NOLIST,'SYSPARM(PARAM)') or ,PARM=(DECK,'NOLIST,SYSPARM(PARAM)')	More than one option is specified. SYSPARM must appear within quotation marks.
,PARM=(DECK,NOLIST,'LINECOUNT(35)', NOALIGN,NORLD)	The whole field must be enclosed in parentheses because it is continued onto another card. The LINECOUNT option must be within single quotation marks, and the portions of the field that are enclosed within quotation marks cannot be continued onto another card.

The assembler options are:

ALIGN|NOALIGN
BATCH|NOBATCH
DECK|NODECK
ESD|NOESD
FLAG(nn)
LINECOUNT(nn)
LIST|NOLIST
OBJECT|NOOBJECT
RENT|NORENT
RLD|NORLD
SYSPARM (string)
TERM|NOTERM
TEST|NOTEST
XREF(FULL|SHORT)|NOXREF

Note: Even though the formats of some of the options previously supported by Assembler H have been changed, you can use the old formats for the following options: ALGN (now ALIGN), NOALGN (NOALIGN), LINECNT=nn (LINECOUNT(nn)), LOAD (OBJECT), NOLOAD (NOOBJECT), MULT (BATCH), NOMULT (NOBATCH), XREF (XREF(FULL)), and MSGLEVEL=nn (FLAG(nnn)).

ALIGN|NOALIGN

ALIGN
The assembler does not suppress the alignment error diagnostic message; all alignment errors are diagnosed.

NOALIGN
The assembler suppresses the diagnostic message "IEV033 ALIGNMENT ERROR" if fixed point, floating point, or logical data referred to by an instruction operand is not aligned on the proper boundary. The message will be produced, however, for references to instructions that are not aligned on the proper (halfword) boundary or for data boundary violations for privileged instructions such as

LPSW. In addition, DC, DS, DXD, or CXD constants, usually causing alignment, are not aligned.

Default: ALIGN

BATCH|NOBATCH

BATCH
The assembler will do multiple (batch) assemblies under the control of a single set of job control language cards. The source decks must be placed together with no intervening /* card; a single /* card must follow the final source deck.

NOBATCH
The BATCH option is suppressed.

Default: NOBATCH

DECK|NODECK

DECK
The object module is placed on the device specified in the SYSPUNCH DD statement.

NODECK
The DECK option is suppressed.

Default: DECK

ESD|NOESD

ESD
The assembler produces the external symbol dictionary as part of the listing.

NOESD
No ESD listing is printed.

Default: ESD

FLAG(nnn)

Error diagnostic messages below severity code nnn will not appear in the listing, and will not be used to set a condition code. Diagnostic messages can have a severity code of 0, 4, 8, 12, 16, or 20 (0 is the least severe). MNOTEs can have a severity code of 0 through 255.

For example, FLAG(8) will suppress messages for severity codes 0 through 7.

Default: 'FLAG(0)'

LINECOUNT(nn)

The number of lines to be printed between headings in the listing is nn. The permissible range is 1 to 99 lines.

Note: The heading occupies 5 of these lines.

Default: 'LINECOUNT(55)'

LIST|NOLIST

LIST
An assembler listing is produced.

NOLIST
No assembler listing is produced. This option overrides ESD, RLD, XREF, and LINECOUNT.

Default: LIST

OBJECT|NOOBJECT

OBJECT
The object module is placed on the device specified in the SYSLIN DD statement.

NOOBJECT
The OBJECT option is suppressed.

Default: NOOBJECT

Note: The OBJECT and DECK options are independent of each other. Both or neither can be specified. The output on SYSLIN and SYSPUNCH is identical, except that the control program closes SYSLIN with a disposition of LEAVE, and SYSPUNCH with a disposition of REREAD.

RENT|NORENT

RENT
The assembler checks for a possible coding violation of program reenterability. Code that makes your program nonreentrant is identified by an error message, but it cannot be an exhaustive check, because the assembler cannot check the logic of the code. Therefore, it is possible to have nonreentrant code not flagged.

NORENT
The RENT option is suppressed.

Default: NORENT

RLD|NORLD

RLD
The assembler produces the relocation dictionary as part of the listing.

NORLD
The relocation dictionary is not printed.

Default: RLD

SYSPARM(string)

'string' is the value of the system variable symbol &SYSPARM. The assembler uses &SYSPARM as a read-only SETC variable. If no value is specified for the SYSPARM option, &SYSPARM will be a null (empty) character string. The function of &SYSPARM is explained in Assembler H Version 2 Application Programming: Language Reference.

Because of JCL restrictions, the length of the SYSPARM value is limited (as explained in Note below). Two single quotation marks are needed to represent a single quotation mark, and two ampersands to represent a single ampersand. For example:

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

assigns the following value to &SYSPARM:

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

Any parentheses inside the string must be paired. If you call the assembler from a problem program (dynamic invocation), SYSPARM can be up to 256 characters long; otherwise, it is limited to 56 characters (see Note below).

Default: 'SYSPARM()'

Note: The restrictions imposed upon the PARM field limit the maximum length of the SYSPARM value to 56 characters, unless you use symbolic procedure parameters to substitute for the value, or the value contains commas that can be used as breaking points between cards. Consider the following example (the underscored characters indicate columns 1, 4, 13, and 68, respectively):

```
//  EXEC  ASMHC,PARM=(OBJECT,NODECK,
//  'SYSPARM(ABCD...................................)')
```

Because SYSPARM uses parentheses, it must be surrounded by single quotation marks. Thus, it cannot be continued onto a continuation card. The leftmost column that can be used is column 4 on a continuation card. A quotation mark and the keyword, as well as the closing quotation mark, must appear on that line. In addition, either a right parenthesis, indicating the end of the PARM field, or a comma, indicating that the PARM field is continued on the next card, must be coded before or in the last column of the statement field (column 71).

TERM|NOTERM

TERM
A summary of error diagnostics is written to the SYSTERM data set for use in sending error messages to a TSO terminal.

NOTERM
The TERM option is suppressed.

Default: NOTERM

TEST|NOTEST

TEST
The object module contains the special source symbol table required by the test translator (TESTRAN) routine.

NOTEST
The special source symbol table is not produced.

Default: NOTEST

XREF(FULL|SHORT)|NOXREF

XREF(FULL)
The assembler listing contains a cross-reference table of all symbols used in the assembly. This includes symbols that are defined but never referenced. The assembler listing also contains a cross-reference table of literals used in the assembly.

XREF(SHORT)
The assembler listing contains a cross-reference table of all symbols that are referred to in the assembly. Any symbols defined but not referred to are not included in the table. The assembler listing also contains a cross-reference table of literals used in the assembly.

NOXREF
No cross-reference tables are printed.

Default: 'XREF(FULL)'

DEFAULT OPTIONS

If you do not code an option in the PARM field, the assembler assumes a default option. The following default options are included when Assembler H is shipped by IBM:

```
PARM=(ALIGN,DECK,ESD,'FLAG(0)','LINECOUNT(55)',
     LIST,NOBATCH,NOOBJECT,NORENT,NOTERM,
     NOTEST,RLD,'SYSPARM()','XREF(FULL)'
```

However, these may not be the default options in effect in your installation; the defaults can be respecified when Assembler H is installed. For example, NODECK can be made the default in place of DECK. Also, a default option that you cannot override can be specified during installation.

The cataloged procedures described in this book assume the default entries. "Overriding Statements in Cataloged Procedures" on page 40 tells you how to override them. First, however, check whether any default options have been changed, or whether there are any you cannot override at your installation.

ASSEMBLER DATA SETS

Assembler H requires the following data sets, as shown in Figure 7 on page 30:

SYSUT1
: A utility data set used as intermediate external storage when processing the source program.

SYSIN
: An input data set containing the source statements to be processed.

In addition, the following five data sets may be required:

SYSLIB
: A data set containing macro definitions (for macro definitions not defined in the source program) and/or source code to be called for through COPY assembler instructions.

SYSPRINT
: A data set containing the assembly listing (unless the NOLIST option is specified).

SYSTERM
: A data set containing essentially a condensed form of SYSPRINT, principally error flagged statements and their error messages (only if the TERM option is specified).

SYSPUNCH
: A data set containing object module output, usually for punching (unless the NODECK option is specified).

SYSLIN
: A data set containing object module output usually for the linkage editor (only if the OBJECT option is specified).

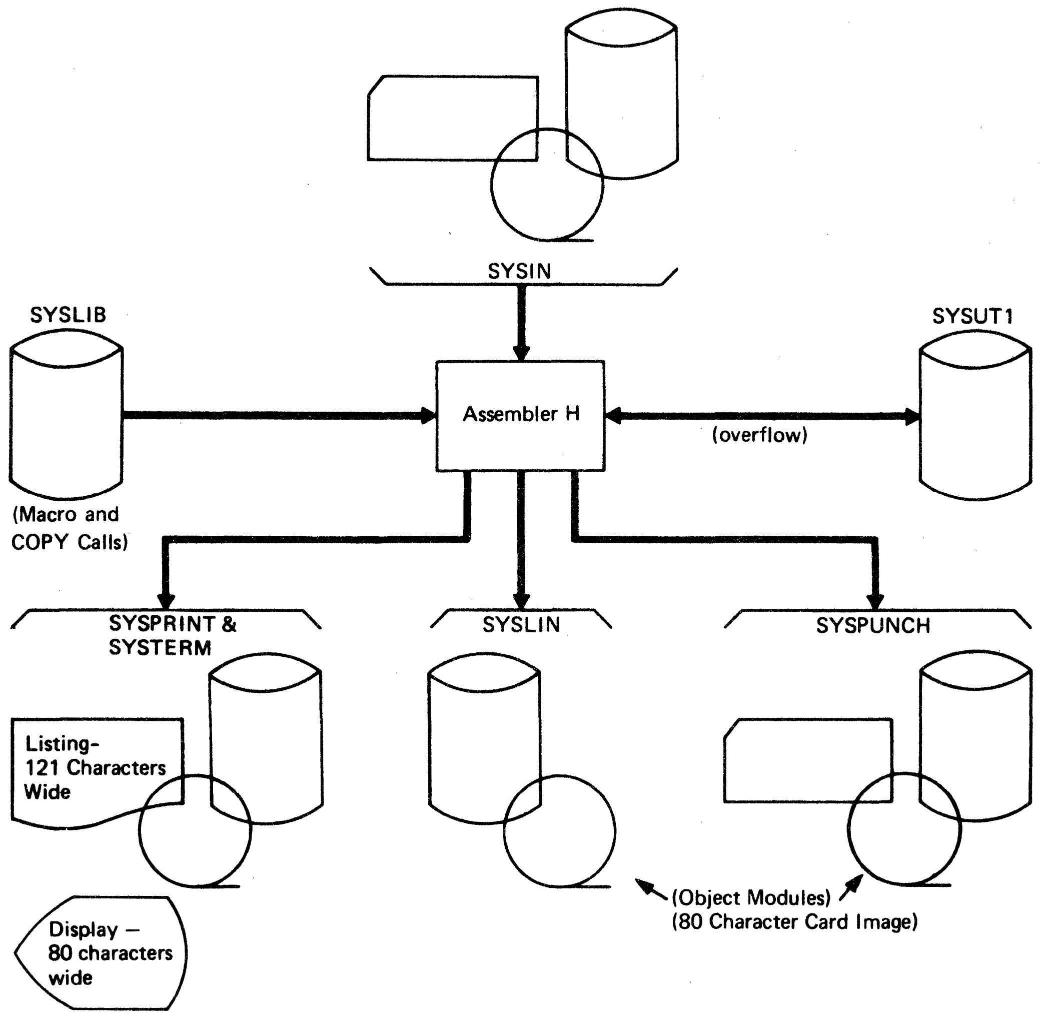


Figure 7. Assembler H Data Sets

The data sets listed above are described in the text following Figure 8 on page 31 and Figure 9 on page 32. The ddname that normally must be used in the DD statement describing the data set appears as the heading for each description. The characteristics of these data sets, those set by the assembler and those you can override, are shown in Figure 8 and Figure 9.

Data Set	SYSUT1	SYSPUNCH	SYSPRINT SYSTERM	SYSLIN	SYSIN	SYSLIB
Access Method	BSAM	BSAM	BSAM	BSAM	BSAM	BSAM
Logical Record Length (LRECL)	Fixed at BLKSIZE	Fixed at 80	Fixed at 121	Fixed at 80	Fixed at 80	Fixed at 80
Block Size (BLKSIZE)	(1)	(2)	(2)	(2)	(2)	(3)
Record Format (RECFM)	(4)	(4,6)	(5,6)	(4,6)	(4,6)	(4,6)
Number of Channel Programs (NCP)	(1)	(7)	(7)	(7)	(7)	Not Applicable

Figure 8. Assembler Data Set Characteristics

Notes to Figure 8:

(1) You can specify a block size (BLKSIZE) between 2008 and 5100 bytes in the DD statement or in the data set label. BLKSIZE should be a multiple of 8; if it is not, it will be rounded to the next lower multiple of 8. If you do not specify BLKSIZE, the assembler sets a default block size based on the device used for SYSUT1.

"Chapter 6. Calculating Storage Requirements" discusses the reasons for changing the default block size.

(2) If specified, BLKSIZE must equal LRECL or a multiple of LRECL. If BLKSIZE is not specified, it is set equal to LRECL. If BLKSIZE is not a multiple of LRECL, it is truncated.

Refer to the appropriate Linkage Editor and Loader for the block size requirements of SYSPUNCH and SYSLIN, if they are used as input to the linkage editor.

(3) BLKSIZE be specified in the DD statement or the data set label as a multiple of LRECL.

(4) Set by the assembler to F or FB if necessary.

(5) Set by the assembler to FM or FBM if necessary.

(6) You may specify B, S, or T.

(7) You can specify the number of channel programs (NCP) used by any assembler data set except SYSUT1 and SYSLIB. The NCP of SYSUT1 is fixed at 1. The assembler, however, can change your NCP specification under certain conditions. Figure 9 on page 32 shows how NCP is calculated.

Note: If the NCP is greater than 2, chained I/O request scheduling is set by the assembler.

	Unit Record Device	**No Unit Record Device**
NCP specified ≥2[1]	User specified	User specified
NCP specified = 1	Computed[2]	User specified(=1)
NCP not specified	Computed[2]	Computed[2]

Figure 9. Number of Channel Program (NCP) Selection

Notes to Figure 9:

[1] If the NCP is greater than 2, chained I/O scheduling is set by the assembler.

[2] For SYSPRINT and SYSTERM data sets, the NCP set by the assembler is the larger of 1210/BLKSIZE or 2. For SYSIN data set, the NCP set by the assembler is the larger of 800/BLKSIZE or 2. For SYSLIN or SYSPUNCH data sets, the NCP set by the assembler is the larger of 240/BLKSIZE or 2.

DDname SYSUT1

The assembler uses this utility data set as an intermediate external storage device when processing the source program. The input/output device assigned to this data set must be a direct-access device. The assembler does not support multivolume utility data sets.

The following are the devices supported for this data set: 3330/3333, 3340/3344, 3350, 3375, and 3380.

DDname SYSIN

This data set contains the input to the assembler—the source statements to be processed. The input/output device assigned to this data set may be either the device transmitting the input stream, or another sequential input device that you have designated. The DD statement describing this data set appears in the input stream. The IBM-supplied procedures do not contain this statement.

DDname SYSLIB

From this data set, the assembler obtains macro definitions and assembler language statements to be called by the COPY assembler instruction. It is a partitioned data set; each macro definition or sequence of assembler language statements is a separate member, with the member being the macro instruction mnemonic or COPY operand name.

The data set may be defined as SYS1.MACLIB or your private macro definition or COPY library. SYS1.MACLIB contains macro definitions for the system macro instructions provided by IBM. Your private library may be concatenated with SYS1.MACLIB. The two libraries must have the same logical record length (80 bytes), but the blocking factors may be different. The DD statement for the library with the largest block size must appear first in the job control language for the assembly (that is, before any library DD statements). The appropriate _JCL Reference_ explains the concatenation of data sets.

DDname SYSPRINT

This data set is used by the assembler to produce a listing. Output may be directed to a printer, a magnetic tape, or a direct-access storage device. The assembler uses the machine code carrier control characters for this data set.

DDname SYSTERM

This data set is used by the assembler to store a summary 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. The assembler uses the machine code carrier control character to skip to a new line for this data set.

DDname SYSPUNCH

The assembler uses this data set to produce the object module. The input/output unit assigned to this data set may be either a card punch or an intermediate storage device capable of sequential access.

DDname SYSLIN

This is a direct-access storage device, a magnetic tape, or a card punch data set used by the assembler. It contains the same output text as SYSPUNCH. It is used as input for the linkage editor.

NUMBER OF CHANNEL PROGRAMS (NCP)

The number of channel programs can be specified by the user or set by the assembler. The number will vary depending upon whether or not a unit record device is used. Figure 9 on page 32 shows how the NCP selection is made.

RETURN CODES

Assembler H issues return codes for use with the COND parameter of the JOB and EXEC job control language statements. The COND parameter enables you to skip or to execute a job step, depending on the results (indicated by the return code) of a previous job step. It is explained in the appropriate 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. See Appendix D, "Assembler H Messages" for a listing of the assembler errors and their severity codes.

CATALOGED PROCEDURES

Often the same set of job control statements is used over and over again (for example, to specify the compilation, linkage editing, and execution of many different programs). To save programming time and to reduce the possibility of error, sets of standard series of EXEC and DD statements can be prepared once and cataloged in a system library. Such a set of statements is termed a cataloged procedure and can be invoked by one of the following statements:

```
//stepname EXEC      procname
//stepname EXEC      PROC=procname
```

The specified procedure 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 (ASMHC); a procedure for assembling and linkage editing (ASMHCL); a procedure for assembling, link-editing, and executing (ASMHCLG); and a procedure for assembling and loader executing (ASMHCG).

CATALOGED PROCEDURE FOR ASSEMBLY (ASMHC)

This procedure consists of one job step: assembly. The name ASMHC must be used to call this procedure. The result of execution is an object module, in punched card form, and an assembler listing. (See also "Simple Assembly and Execution" on page 22 for more details and another example.)

In the following example, input enters via the input stream. An example of the statements entered in the input stream to use this procedure is:

```
//jobname           JOB
//stepname          EXEC PROC=ASMHC
//SYSIN           DD   *
                    .
                    .
                    .
          source program statements
                    .
                    .
                    .
/* (delimiter statement)
```

The statements of the ASMHC procedure are read from the procedure library and merged into the input stream.

Figure 10 shows the statements that make up the ASMHC procedure.

```
//C            EXEC   PGM=IEV90,REGION=200K                                 (1)

//SYSLIB       DD     DSN=SYS1.MACLIB,DISP=SHR                              (2)

//SYSUT1       DD     UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(10,5)),DSN=&SYSUT1     (3)

//SYSPUNCH     DD     SYSOUT=B,DCB=(BLKSIZE=800),SPACE=(CYL,(5,5,0))        (4)

//SYSPRINT     DD     SYSOUT=A,DCB=(BLKSIZE=3509),UNIT=(,SEP=(SYSUT1,SYSPUNCH)) (5)
```

Figure 10. Cataloged Procedure for Assembly (ASMHC)

Notes to Figure 10:

(1) PARM= or COND= parameters may be added to this statement by the EXEC statement that calls the procedure (see "Overriding Statements in Cataloged Procedures"). The system name IEV90 identifies Assembler H.

(2) This statement identifies the macro library data set. The data set name SYS1.MACLIB is an IBM designation.

(3) This statement specifies the assembler utility data set. The device class name used here, SYSDA, 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 3330 may be substituted for SYSDA.

(4) This statement describes the data set that will contain the object module produced by the assembler.

(5) This statement defines the standard system output class, SYSOUT=A, as the destination for the assembler listing.

CATALOGED PROCEDURE FOR ASSEMBLY AND LINKAGE EDITING (ASMHCL)

This procedure consists of two job steps: assembly and linkage editing. The name ASMHCL must be used to call this procedure. Execution of this procedure results in the production of an assembler listing, a linkage editor listing, and a load module.

The following example illustrates input to the assembler via the input job stream. SYSLIN contains the output from the assembly step and the input to the linkage edit step. It can be concatenated with additional input to the linkage editor as shown in the example. This additional input can be linkage editor 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=ASMHCL
//C.SYSIN        DD  *
                 .
                 .
                 .
         source program statements
                 .
                 .
                 .
/*
//L.SYSIN        DD  *
                 .
                 .
                 .
       object module or
       linkage editor
       control statements
/*
```

Note: //L.SYSIN is necessary only if the linkage editor is to combine modules or read linkage editor control information from the job stream.

Figure 11 on page 36 shows the statements that make up the ASMHCL procedure. Only those statements not previously discussed are explained.

```
//C          EXEC   PGM=IEV90,PARM=OBJECT,REGION=200K

//SYSLIB     DD     DSN=SYS1.MACLIB,DISP=SHR

//SYSUT1     DD     UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(10,5)),DSN=&SYSUT1

//SYSPUNCH   DD     SYSOUT=B,DCB=(BLKSIZE=800),SPACE=(CYL,(5,5,0))

//SYSPRINT   DD     SYSOUT=A,DCB=(BLKSIZE=3509),UNIT=(,SEP=(SYSUT1,SYSPUNCH))

//SYSLIN     DD     DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5,0)),        (1)

//                  DCB=(BLKSIZE=400),DSN=&&LOADSET

//L          EXEC   PGM=IEWL,PARM='MAP,LET,LIST,NCAL',REGION=96K,COND=(8,LT,C)
                                                                        (2)

//SYSLIN     DD     DSN=&&LOADSET,DISP=(OLD,DELETE)                     (3)

//           DD     DDNAME=SYSIN                                        (4)

//SYSLMOD    D      DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(2,1,2)),DSN=&GOSET(GO)
                                                                        (5)

//SYSUT1     DD     UNIT=SYSDA,SPACE=(CYL,(3,2)),DSN=&SYSUT1            (6)

//SYSPRINT   DD     SYSOUT=A,DCB=(RECFM=FB,BLKSIZE=3509)                (7)
```

Figure 11. Cataloged Procedure for Assembling and Link-Editing (ASMHCL)

Notes to Figure 11:

(1) In this procedure, the SYSLIN DD statement describes a temporary data set, the object module, which is passed to the linkage editor.

(2) This statement initiates linkage editor execution. The linkage editor options in the PARM field cause the linkage editor to produce a cross-reference table, a module map, and a list of all control statements processed by the linkage editor. The NCAL option suppresses the automatic library call function of the linkage editor.

(3) This statement identifies the linkage editor 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 linkage editor from the input stream (object decks and/or linkage editor control statements) with the input from the assembler.

(5) This statement specifies the linkage editor output data set (the load module). As specified, the data set will be deleted at the end of the job. If it is desired to retain the load module, the DSN parameter must be respecified and a DISP parameter added. See "Overriding Statements in Cataloged Procedures." If the output of the linkage editor is to be retained, the DSN parameter must specify a library name and a member name at which the load module is to be placed. The DISP parameter must specify either KEEP or CATLG.

(6) This statement specifies the utility data set for the linkage editor.

(7) This statement identifies the standard output class as the destination for the linkage editor listing.

CATALOGED PROCEDURE FOR ASSEMBLY, LINK-EDITING, AND EXECUTION (ASMHCLG)

This procedure consists of three job steps: assembly, link-editing, and execution.

The name ASMHCLG must be used to call this procedure. An assembler listing, an object deck, and a linkage editor listing are produced.

The statements entered in the input stream to use this procedure are:

```
//jobname         JOB
//stepname        EXEC PROC=ASMHCLG
//C.SYSIN         DD  *
                  .
                  .
                  .
       source program statements
                  .
                  .
                  .
/*
//L.SYSIN         DD  *
                  .
                  .
                  .
           object module or
           linkage editor
           control statements
                  .
                  .
                  .
/*
//G.ddname        DD   (parameters)
//G.ddname        DD   (parameters)
//G.ddname        DD   *
                  .
                  .
                  .
         problem program input
                  .
                  .
                  .
/*
```

Notes:

1. //L.SYSIN is necessary only if linkage editor is to combine modules or read linkage editor control information from the job stream.

2. //G.ddname statements are included only if necessary.

Figure 12 shows the statements that make up the ASMHCLG procedure. Only those statements not previously discussed are explained in the figure.

```
//C          EXEC    PGM=IEV90,PARM=OBJECT,REGION=200K

//SYSLIB     DD      DSN=SYS1.MACLIB,DISP=SHR

//SYSUT1     DD      UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(10,5)),DSN=&SYSUT1

//SYSPUNCH   DD      SYSOUT=B,DCB=(BLKSIZE=800),SPACE=(CYL,(5,5,0))

//SYSPRINT   DD      SYSOUT=A,DCB=(BLKSIZE=3509),UNIT=(,SEP=(SYSUT1,SYSPUNCH))

//SYSLIN     DD      DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5,0)),

//                   DCB=(BLKSIZE=400),DSN=&&LOADSET

//L          EXEC    PGM=IEWL,PARM='MAP,LET,LIST,NCAL',REGION=96K,COND=(8,LT,C)
                                                                      (1)

//SYSLIN     DD      DSN=&&LOADSET,DISP=(OLD,DELETE)

//           DD      DDNAME=SYSIN

//SYSLMOD    DD      DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(2,1,2)),DSN=&GOSET(GO)
                                                                      (2)

//SYSUT1     DD      UNIT=SYSDA,SPACE=(CYL,(3,2)),DSN=&SYSUT1

//SYSPRINT   DD      SYSOUT=A,DCB=(RECFM=FB,BLKSIZE=3509)

//G          EXEC    PGM=*.L.SYSLMOD,COND=((8,LT,C),(4,LT,L))          (3)
```

Figure 12. Cataloged Procedure for Assembly, Link-Editing, and Execution (ASMHCLG)

Notes to Figure 12:

(1) The LET linkage editor option specified in this statement causes the linkage editor to mark the load module as executable even though errors were encountered during processing.

(2) The output of the linkage editor 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 initiates execution of the assembled and link-edited program. The notation *.L.SYSLMOD identifies the program to be executed as being in the data set described in job step L by the DD statement named SYSLMOD.

CATALOGED PROCEDURE FOR ASSEMBLY AND LOADER EXECUTION (ASMHCG)

This procedure consists of two job steps: assembly and loader execution. Loader execution is a combination of linkage editing and loading the program for execution. Load modules for program libraries are not produced. (See also "Simple Assembly and Execution" on page 22 for more details and another example.)

The statements entered in the input stream to use this procedure are:

```
//jobname        JOB
//stepname       EXEC PROC=ASMHCG
//C.SYSIN        DD  *
                 .
                 .
                 .
          source program
                 .
                 .
                 .
/*
//G.ddname       DD   (parameters)
//G.ddname       DD   (parameters)
//G.ddname       DD   *
                 .
                 .
                 .
        problem program input
                 .
                 .
                 .
/*
```

Note: //G.ddname statements are included only if necessary.

Figure 13 shows the statements that make up the ASMHCG procedure. Only those statements not previously discussed are explained in the figure.

The name ASMHCG must be used to call this procedure. Assembler and loader listings are produced.

```
//C              EXEC    PGM=IEV90,PARM=OBJECT,REGION=200K

//SYSLIB         DD      DSN=SYS1.MACLIB,DISP=SHR

//SYSUT1         DD      UNIT=(SYSDA,SEP=SYSLIB),SPACE=(CYL,(10,5)),DSN=&SYSUT1

//SYSPUNCH       DD      SYSOUT=B,DCB=(BLKSIZE=800),SPACE=(CYL,(5,5,0))

//SYSPRINT       DD      SYSOUT=A,DCB=(BLKSIZE=3509),UNIT=(,SEP=(SYSUT1,SYSPUNCH))

//SYSLIN         DD      DISP=(,PASS),UNIT=SYSDA,SPACE=(CYL,(5,5,0)),

//                       DCB=(BLKSIZE=400),DSN=&&LOADSET

//G              EXEC    PGM=LOADER,PARM='MAP,LET,PRINT,NOCALL'            (1)

//SYSLIN         DD      DSN=&&LOADSET,DISP=(OLD,DELETE)                   (2)

//               DD      DDNAME=SYSIN

//SYSLOUT        DD      SYSOUT=A                                          (3)
```

Figure 13. Cataloged Procedure for Assembly and Loader Execution (ASMHCG)

Notes to Figure 13:

(1) This statement initiates loader execution. 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 linkage editor, and the LET option is the same as for the linkage editor.

(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

Any parameter in a cataloged procedure can be overridden except the PGM= parameter in the EXEC statement. Such 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.

Overriding for the purposes of respecification, addition, or nullification is accomplished by including in the input stream statements containing the desired changes and identifying 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 notation PARM.stepname=, or COND.stepname=, followed by the desired parameters. "Stepname" identifies the EXEC statement within the procedure to which the modification applies.

If the procedure consists of more than one job step, a PARM.procstepname= or COND.procstepname= parameter may be entered for each step. The entries must be in order (PARM.procstepname1=, PARM.procstepname2=, etc.).

DD Statements

All parameters in the operand field of DD statements may be overridden by including in the input stream (following the EXEC card 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.

Note: 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

Example 1: In the assembly procedure ASMHC (Figure 10 on page 34), the production of a punched object deck could be suppressed and the UNIT= and SPACE= parameters of data set SYSUT1 respecified, by including the following statements in the input stream:

```
//stepname      EXEC   PROC=ASMHC,                          X
//                     PARM=NODECK
//SYSUT1        DD     UNIT=3330,                           X
//                     SPACE=(200,(300,40))                 X
//SYSIN         DD     *
                .
                .
                .
                source statements
                .
                .
                .
/*
```

Example 2: In procedure ASMHCLG (Figure 12 on page 38), suppressing production of an assembler listing and adding the COND= parameter to the EXEC statement, which specifies execution of the linkage editor, may be desired. In this case, the EXEC statement in the input stream would appear as follows:

```
  //stepname     EXEC   PROC=ASMHCLG,                 X
  //                    PARM.C=(NOLIST,OBJECT),       X
  //                    COND.L=(8,LT,stepname.C)
```

For this execution of procedure ASMHCLG, no assembler listing would be produced, and execution of the linkage editor job step //L would be suppressed if the return code issued by the assembler (step C) were greater than 8.

Note: When you override the PARM field in a procedure, the entire PARM field is overridden. Thus, in this example, overriding the LIST parameter effectively deletes PARM=OBJECT. PARM=OBJECT must be repeated in the override statement; otherwise, the assembler default value NOOBJECT will be used.

Example 3: The following list shows how to use the procedure ASMHCL (Figure 11 on page 36) 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 linkage editor with a blocking factor of 5.

4. Link-edit the module only if there are no errors in the assembler (COND=0).

5. Link-edit onto a previously allocated and cataloged data set USER.LIBRARY with a member name of PROG.

```
//jobname      JOB
//stepname     EXEC   PROC=ASMHCL,                    X
//                    COND.L=(0,NE,stepname.C)
//C.SYSPRINT   DD     DSNAME=PROG1,UNIT=TAPE,         X
//                    VOLUME=SER=TAPE10,DCB=(BLKSIZE=605)
//C.SYSLIN     DD     DCB=(BLKSIZE=800)
//C.SYSIN      DD     UNIT=282,LABEL=(,NL),           X
//                    DCB=(RECFM=FBS,BLKSIZE=800)
//L.SYSIN      DD     DCB=stepname.C.SYSLIN
//L.SYLMOD     DD     DSNAME=USER.LIBRARY(PROG),DISP=OLD
/*
```

Note: 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 appropriate _JCL Reference_.

Example 4: The following example shows assembly of two programs, link-editing of the two assemblies into one load module, and execution of the load module. The input stream appears as follows:

```
//stepname1      EXEC     PROC=ASMHC,PARM=OBJECT
//SYSLIN         DD       DSNAME=&LOADSET,UNIT=SYSSQ,        X
//                        SPACE=(80,(100,50)),               X
//                        DISP=(MOD,PASS),DCB=(BLKSIZE=800)
//SYSIN          DD       *
                          .
                          .
                          .
                          source program 1 statements
                          .
                          .
                          .
/*
//stepname2      EXEC     PROC=ASMHCLG
//C.SYSLIN       DD       DCB=(BLKSIZE=800),DISP=(MOD,PASS)
//C.SYSIN        DD       *
                          .
                          .
                          .
                          source program 2 statements
                          .
                          .
                          .
/*
//L.SYSIN        DD       *
                 ENTRY PROG
/*
//G.ddname       DD       dd cards for G step
```

The appropriate JCL Reference provides additional descriptions of overriding techniques.

CHAPTER 5. PROGRAMMING CONSIDERATIONS

This chapter discusses some topics in assembler language programming.

SAVING AND RESTORING GENERAL REGISTER CONTENTS

A problem program should save the values contained in the general registers upon commencing execution and, upon completion, restore to the general registers these same values. Thus, as control is passed from the operating system to a problem 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.

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 problem program is given control, register 13 contains the address of an area in which the general contents should be saved.

If the program calls any subprograms, or uses any operating system services other than GETMAIN, FREEMAIN, ATTACH, and XCTL, it must first save the contents of register 13 and then load the address of an 18-fullword save area into register 13. This save area is in the problem program and is used by any subprograms or operating system services called by the problem program.

At completion, the problem 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 execution of the RETURN macro instruction.

The coding sequence that follows illustrates 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 appropriate Supervisor Services and Macro Instructions.

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	

PROGRAM TERMINATION

You indicate completion of an assembler language source program by using the RETURN system macro instruction to pass control from the terminating program to the program that initiated it. The initiating program may be the operating system or, if a subprogram issued the RETURN, the program that called the subprogram.

In addition to indicating program completion and restoring register contents, the RETURN macro instruction may also pass a return code—a condition indicator that may be used by the program receiving control.

If the return is to the operating system, the return code is compared against the condition stated in the COND= parameter of the JOB or EXEC statement.

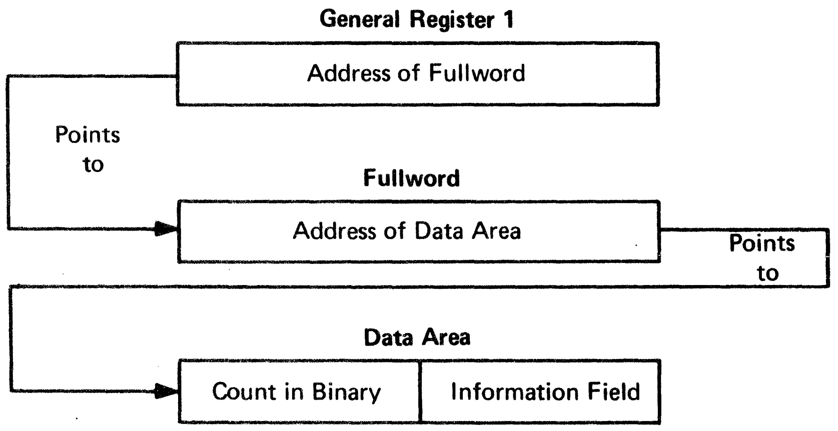
If return is to another problem program, the return code is available in general register 15, and may be used as desired. Your program should restore register 13 before issuing the RETURN macro instruction.

The RETURN system macro instruction is discussed in detail in the appropriate Supervisor Services and Macro Instructions.

PARM FIELD ACCESS

Access to information in the PARM field of an EXEC statement is gained through general register 1. When control is given to the problem 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. The following diagram illustrates this process:



MACRO DEFINITION LIBRARY ADDITIONS

Source statement coding, to be retrieved by the COPY assembler instruction, and macro definitions may be added to the macro library. The IEBUPDTE utility program is used for this purpose. Details of this program and its control statements are contained in the appropriate _Utilities_ publication. The following example shows how a new macro definition, NEWMAC, is added to the system library, SYS1.MACLIB.

```
//CATMAC    JOB         12345,BROWN.JR,...
//STEP1     EXEC        PGM=IEBUPDTE,PARM=MOD
//SYSUT1    DD          DSNAME=SYS1.MACLIB,DISP=OLD
//SYSUT2    DD          DSNAME=SYS1.MACLIB,DISP=OLD
//SYSPRINT  DD          SYSOUT=A
//SYSIN     DD          DATA
./          ADD         LIST=ALL,NAME=NEWMAC,LEVEL=01,SOURCE=0
            MACRO
            NEWMAC &OP1,&OP2
            LCLA   &PAR1,&PAR2
            .
            .
            .
            MEND
./          ENDUP
/*
```

The SYSUT1 and SYSUT2 DD statements indicate that SYS1.MACLIB, an existing program library, is to be updated. Output from the IEBUPDTE program is printed on the Class A output device (specified by SYSPRINT). 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 macro definition.

LOAD MODULE MODIFICATION—ENTRY POINT RESTATEMENT

If the editing functions of the linkage editor are to be used to modify a load module, the entry point to the load module must be restated when the load module is reprocessed by the linkage editor. Otherwise, the first byte of the first control section processed by the linkage editor will become 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. External symbol identification is done automatically by the assembler if the entry point is the name of a control section or START statement; otherwise, an assembler ENTRY statement must be used to identify the entry point as an external symbol.

When a new object module is added to or replaces part of the load module, the entry point is restated in one of three ways:

- By placing the entry point symbol in the operand field of an EXTRN statement and an END statement in the new object module
- By using an END statement in the new object module to designate a new entry point in the new object module
- By using a linkage editor ENTRY statement to designate either the original entry point or a new entry point for the load module

Further discussion of load module entry points is contained in the appropriate _Linkage Editor and Loader_.

OBJECT MODULE LINKAGE

Object modules, whether generated by the assembler or by another language processor, may be combined by the linkage editor to produce a composite load module, provided each object module conforms to the data formats and linkage conventions required. 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. This topic discusses the use of the CALL system macro instruction to link an assembler language main program to subprograms produced by another processor. The appropriate *Supervisor Services and Macro Instructions* manual contains additional details concerning linkage conventions and the CALL system macro instruction.

Figure 14 on page 47 is an example of statements used to establish the assembler language program linkage to FORTRAN and COBOL subprograms.

If any input/output operations are performed by called subprograms, appropriate DD statements for the data sets used by the subprograms must be supplied. See the appropriate language programmer's guide for an explanation of the DD statements and special data set record formats used for the processor. See Appendix C, "Object Deck Output" for the format of the object deck.

```
ENTRPT SAVE   (14,12)
       LR     12,15
       USING  ENTRPT,12
       ST     13,SVAREA+4         (1)
       LA     15,SVAREA
       ST     15,8(13)
       LR     13,15
       .
       .
       CALL   name,(V1,V2,V3),VL  (2)
       .
       .
       L      13,SVAREA+4
       RETURN (14,12)
SVAREA DC     18F'0'              (3)
V1     DC     (data)              (4)
V2     DC     (data)              (5)
V3     DC     (data)              (6)
       END
```

(1) This is an example of OS/VS linkage convention. See your system Supervisor Services and Macro Instructions for details.

(2) The symbol used for "name" in this statement is:

(a) The name of a subroutine or function, when the linkage is to a FORTRAN-written subprogram.

(b) The name defined by the following COBOL statements in the procedure division:
ENTER LINKAGE. ENTRY'name'.

(c) The name of a CSECT or START statement, or a name used in the operand field of an ENTRY statement in an assembler-language subprogram.

The order in which the parameter list is written must reflect the order in which the called subprogram expects the argument. If the called routine is a FORTRAN-written function, the returned argument is not in the parameter list: a real or double precision function returns the value in floating point register zero; an integer function returns the value in general purpose register zero.

Note: When linking to FORTRAN-written subprograms, consideration must be given to the storage requirements of IBCOM (FORTRAN execution-time I/O and interrupt handling routines) which accompanies the compiled FORTRAN subprogram. In some instances, the call for IBCOM is not automatically generated during the FORTRAN compilation. VS FORTRAN Application Programming: Library Reference provides information about IBCOM requirements and assembler statements used to call IBCOM.

FORTRAN-written subprograms and FORTRAN library subprograms allow variable-length parameter lists in linkages which call them; therefore, all linkages to FORTRAN subprograms are required to have the high-order bit in the last parameter in the linkage set to 1. COBOL-written subprograms have fixed-length calling linkages; therefore, for COBOL the high-order bit in the last parameter need not be set to 1.

(3) This statement reserves the save area needed by the called subprogram. When control is passed to the subprogram, register 13 contains the address of this area.

(4,5,6) When linking to a FORTRAN or COBOL subprogram, the data formats declared in these statements are determined by the data formats required by the FORTRAN or COBOL subprograms.

Figure 14. Sample Assembler Linkage Statements for FORTRAN or COBOL Subprograms

LINKING WITH IBM-SUPPLIED PROCESSING PROGRAMS

You usually use the EXEC job control statement to load and give control to a processing program of the operating system. However, you can also load and give control to a sort program, a utility program, or even a compiler "dynamically," that is, by using a system macro instruction (LINK, XCTL, CALL, or ATTACH) in your own program.

Note: If you use the ATTACH macro instruction, the MVS/XA object program will not run on S/370. See MVS/Extended Architecture Conversion Notebook for more details.

When calling a program dynamically, make sure you follow the OS/VS linking conventions described in the appropriate Supervisor Services and Macro Instructions manual. You must also pass certain parameters to the processing program. These parameters give the same information to the program as you would supply in job control statements if you called the program with an EXEC statement. The following section describes how to call the assembler dynamically. Dynamic invocation of each of the other IBM-supplied processing programs is covered in one of the manuals describing that program.

INVOKING THE ASSEMBLER DYNAMICALLY

Assembler H can be invoked by a problem program at execution time through use of the CALL, LINKAGE, XCTL, or ATTACH macro instruction. If the XCTL macro instruction is used to invoke the assembler, no user options may be stated. The assembler will use the standard default, as set during system generation, for each option.

If the assembler is invoked by CALL, LINKAGE, or ATTACH, you may supply:

- The assembler options
- The ddnames of the data sets to be used during processing

Name	Operation	Operand
[symbol]	CALL	IEV90,(optionlist [,ddnamelist]),VL
	{LINK| ATTACH}	EP=IEV90, PARAM=(optionlist [,ddnamelist]),VL=1

EP
specifies the symbolic name of the assembler. The entry point at which execution is to begin is determined by the control program (from the library directory entry).

PARAM
specifies, as a sublist, address parameters to be passed from the problem program to the assembler. The first word in the address parameter list contains the address of the option list. The second word contains the address of the ddname list.

optionlist
specifies the address of a variable-length list containing the options. This address must be written even if no option list is provided.

The option list must begin on a halfword boundary, that is, not also a fullword 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 blanks or zeros appear in the list.

DDnamelist
specifies the address of a variable-length list containing alternative ddnames for the data sets used during compiler processing. If standard ddnames are used, this operand may be omitted.

The ddname 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. Each name of less than 8 bytes must be left-justified and padded with blanks. If an alternative ddname is omitted, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the 8-byte entries in the ddname list is as follows:

Entry	**Alternative**
1	SYSLIN
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSTERM

Note: An overriding ddname specified when Assembler H was added to the operating system occupies the same place in the above list as the IBM-supplied ddname it overrides. The overriding ddname can itself be overridden during invocation. For example, if SYSWORK1 replaced SYSUT1, it occupies position 8 in the above list. SYSWORK1 can be overridden by another name during invocation.

VL
specifies that the sign bit is to be set to 1 in the last word of the address parameter list.

The appropriate JCL Reference provides additional description of overriding techniques.

CHAPTER 6. CALCULATING STORAGE REQUIREMENTS

MAIN STORAGE

When Assembler H is run in a 200K-byte region, about half the region is devoted to fixed storage for load modules, data management, and operating system workspace. The other half is allotted to variable storage for buffers, tables, and intermediate results. If the region size is varied, the size of the variable storage will be affected. There are ways to decrease the size of fixed storage, whether the region size is increased or kept at 200K bytes.

FIXED STORAGE

Fixed storage accounts for approximately 95K bytes, of which about 86K bytes are needed for load modules. Figure 15 on page 51 shows the assembler's use of a 200K-byte region. Neither time nor main storage is drawn to scale. The shaded portion represents main storage that is free at any point in time.

Figure 15 represents a series of assemblies in BATCH mode. The first few events follow. For further details, see "Program Organization" in Assembler H Version 2: Logic.

1. Module IEV90 is loaded first.
2. Module IEV90 loads modules IEV00 and IEV10, then transfers control to module IEV00.
3. Module IEV00 loads module IEV60, opens the necessary data sets (bringing in Data Management modules), gets all remaining free space in the region by a GETMAIN, releases 4K bytes for OS transient use, and returns to module IEV90.
4. Module IEV90 deletes module IEV00, loads module IEV50, and transfers control to module IEV10.
5. Module IEV90 deletes module IEV10, loads module IEV20, and transfers control to module IEV20.
6. Module IEV90 deletes module IEV20, loads module IEV10, etc.

An installation can reduce the region size or increase the amount of variable storage by putting one or more modules into the link pack area. Note that approximately 6K bytes can be saved if the required BSAM data management modules are in link pack.

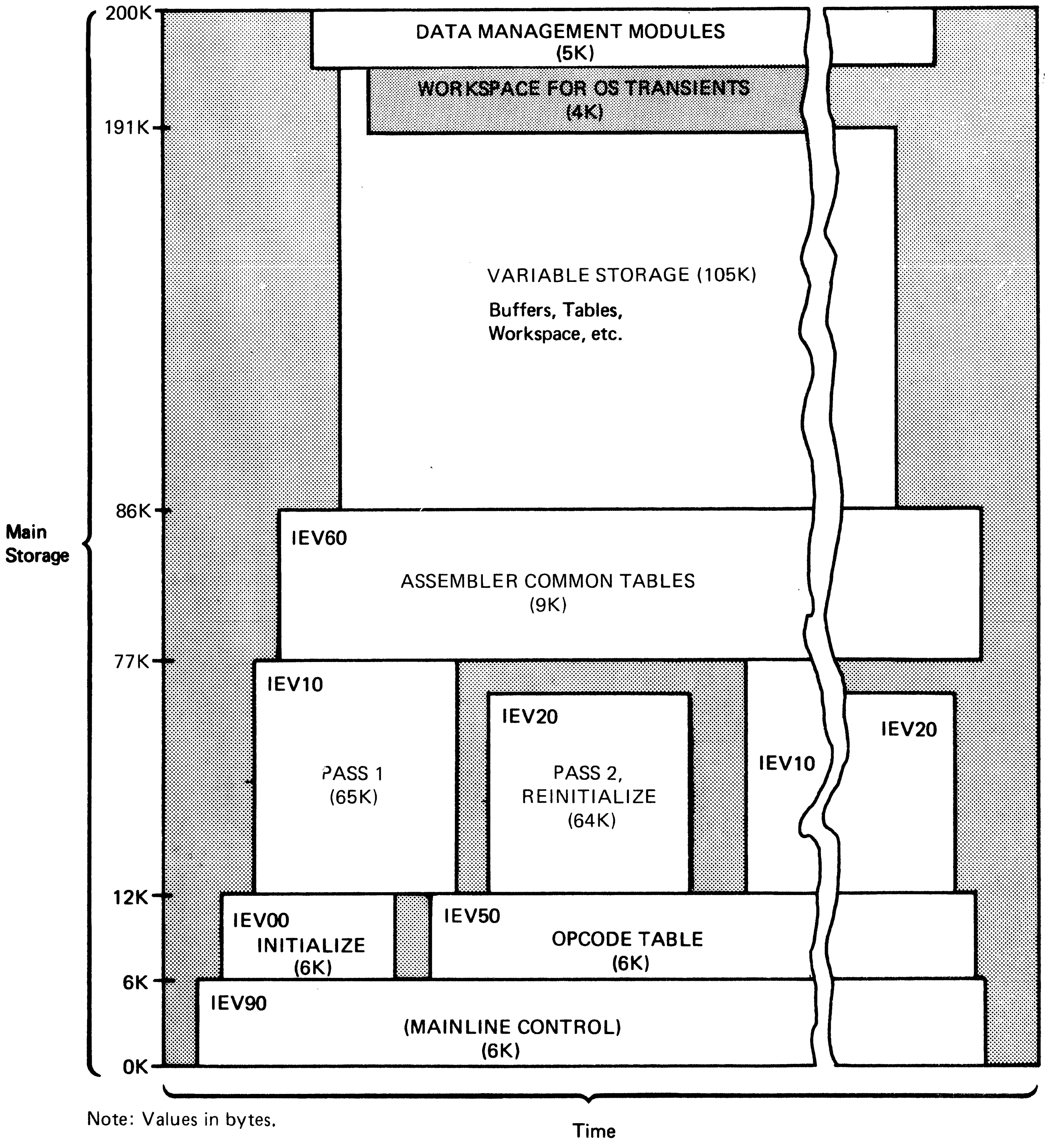


Figure 15. Basic Layout for Assembler H

Figure 16 shows the amount of space required in link pack by the indicated modules, and the reduced minimum region required for the assembler.

Modules	Space in Link Pack	Assembler H Region (in Bytes)
None	0	200K
IEV90	6K	194K
IEV10 IEV20 IEV80	133K	135K
IEV10 IEV20 IEV80 IEV90	139K	129K

Figure 16. Required Space in Link Pack

Module IEV80 was not shown in Figure 15 on page 51. It is called by IEV90 only if an I/O error from which the system cannot recover occurs, or if Assembler H encounters an impossible situation. Module IEV80 produces a formatted dump of the region.

If Assembler H is in link pack, a maximum of only a few seconds is saved for each assembly. However, if a high volume of assemblies justifies keeping two regions active, the saving in region size shown in Figure 16 becomes more meaningful.

VARIABLE STORAGE

Buffers

The amount of main storage that module IEV00 sets aside for buffers can be considerable. Consider the following example:

```
  OPTIONS=BATCH,DECK,OBJECT
  BLKSIZE=3200  (for SYSIN)
          3360  (for SYSLIB)
          3146  (for SYSPRINT)
           400  (for SYSPUNCH and SYSLIN)
  2 buffers for each data set
```

Then,

```
  BUFFERS = 2(3200+3360+3146+400+400) bytes
          = 21,012 bytes
```

If all factors are as above except PARM=NOBATCH, then

```
  BUFFERS = MAX[(SYSIN+SYSLIB),(SYSPRINT+SYSPUNCH+SYSLIN)]
          = MAX[2(3200+3360),2(3146+400+400)]
          = MAX[(13,120),(7,892)]
          = 13,120 bytes
```

Either way, the assembler is tying up a lot of variable storage for buffers.

Suppose 200K bytes is the size of the largest region available in a particular installation and there is no possibility of putting Assembler H modules into link pack. If a particularly large source deck will not assemble under Assembler H because of a lack of variable storage, then you can attempt the following procedures, in the indicated sequence, singly or in combination:

1. If both the options TERM and LIST have been specified, see whether one of them can be eliminated.

2. Decrease BLKSIZE, particularly on SYSIN and SYSPRINT. The distributed cataloged procedures (for details, see "Cataloged Procedures" on page 33) include the following DD statement:

   ```
   //SYSPRINT   DD  SYSOUT=A,DCB=(BLKSIZE=3509)
   ```

   Override these as follows:

   ```
   //SYSPRINT   DD  SYSOUT=A,DCB=(BLKSIZE=1210)
   //SYSIN      DD  *,DCB=(BLKSIZE=800)
   ```

   Note that BLKSIZE must be a multiple of 121 for SYSPRINT and SYSTERM, and a multiple of 80 for SYSIN, SYSLIB, SYSPUNCH, and SYSLIN.

3. Copy SYSLIB to a private library, reblocking it to a smaller size. The new BLKSIZE must be a multiple of 80. Override the SYSLIB DD statement to indicate the new blocking factor and the new DSNAME.

4. Consider the default setting of SYSUT1 described below. Specify, by overriding the default, a smaller BLKSIZE on the SYSUT1 DD card. See "Work File Blocks," below, for details.

5. If none of these procedures solves the problem, you are faced with the prospect of breaking the single, large program down into two or more smaller ones.

Assembler H keeps the ordinary symbol table and global dictionary in main storage throughout Pass 1 (IEV10). This leads to the type of problem covered by the above five steps. Before breaking the program into smaller ones, you might attempt to decrease the number of symbols that are in your program or are generated by your program.

For example, if you use the DCBD macro to define all possible symbolic fields of a DCB and actually use only one such field, you have unknowingly put about 100 unused symbols into the symbol table. These 100 symbols occupy about 3400 bytes.

Work File Blocks

After setting aside sufficient variable storage for data set buffers, IEV00 divides the remaining variable storage into work file blocks.

Several factors are considered in determining the block size. They include the following:

1. Many of these blocks will be spilled onto SYSUT1. For efficient utilization of SYSUT1 space, the block sizes should be chosen from full-track, half-track, third-track, etc., sizes corresponding to the device assigned to SYSUT1.

2. The block size should be reasonable.

3. For ease of internal processing, the block size should be a multiple of 8.

The default size selected (that is, the largest block size satisfying 1, 2, and 3 above) for a 3330/3333 direct-access device is 4248 bytes.

The various routines in Assembler H are given one block of work space at a time, as needed. Once obtained, the blocks are not reusable until the requesting routine indicates that they can be returned or spilled onto SYSUT1. Depending on the assembly and the device used for SYSUT1, this may result in inefficient use of main storage.

For example, Pass 2 needs a block for RLDs. If SYSUT1 is a 3330 and there is only one RLD involved, then Pass 2 ties up 4248 bytes of main storage for 8 bytes of useful information. Because there is room for fewer than 20 of these blocks in the normal 200K-byte region, it is conceivable that the assembler could run out of main storage in some situations.

As pointed out in the previous section, one method of attempting to remedy this situation would be to override the block size (BLKSIZE) for SYSUT1 on the DD statement. Thus, in the case of the 3330 (refer to "Cataloged Procedures" on page 33), you could use the following:

```
  //SYSUT1 DD UNIT=SYSDA,DCB=(BLKSIZE=2056),SPACE=(CYL,(10,5))
```

Strictly speaking, you do not need to restrict yourself to the natural divisors (full, half, third, quarter) of device tracks. However, you should be aware of the consequences of a poor choice. For example, 4248 bytes is nearly a third of a track for the 3330; 4144 bytes is also nearly a third of a track, but 4352 bytes is too big—only two 4352-byte blocks would fit on each track. In addition, making the block size too small may cause unusually heavy I/O activity on SYSUT1 and hinder performance. Assembler H will set the SYSUT1 block size to the default value if you attempt to set it to less than 2008 bytes.

You can specify a BLKSIZE larger than the size of a track for the device if you also specify the parameter RECFM=T (for track overflow); naturally, the device used for SYSUT1 must have the track overflow feature. If the BLKSIZE specified is larger than a track but track overflow is not specified in RECFM, the assembler takes the default block size for the device.

Symbol Tables

A program containing approximately 1000 symbols, each symbol occupying about 34 bytes of main storage work space, can be assembled in the Assembler H 200K-byte region.

Figure 17 on page 55 can be used as a guide in assessing the amount of main storage needed to assemble a program with a given number of symbols.

Overall Dynamic Storage

Assembler H uses BPAM to access library data and BSAM for general data management. The assembler can run on any OS/VS system that has a virtual storage area of 200K bytes assigned to it.

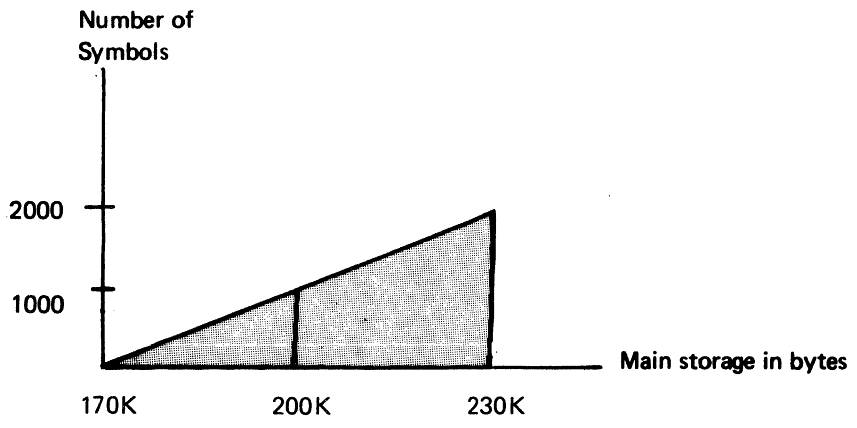


Figure 17. Aid in Assessing Main Storage Required by a Symbol Table with 1000 or 2000 Symbols

AUXILIARY STORAGE ESTIMATES

WORK FILE SPACE FOR SYSUT1

During both Pass 1 and Pass 2, the single work file SYSUT1 is used for intermediate results. Distributed cataloged procedures (see "Cataloged Procedures" on page 33) for SYSUT1 show a primary allocation of 10 cylinders and up to 15 additional secondary allocations in increments of 5 cylinders each. This should be sufficient for most assemblies.

The amount of SYSUT1 space used is almost independent of region size. As pointed out earlier, a poor choice of BLKSIZE for SYSUT1 could drastically increase the direct access space needed. Whenever IEV10 fills a block that can be spilled to SYSUT1, the block is written out to SYSUT1 in anticipation of a need to reuse the main storage space. If this need never arises and the main storage space is never overlaid, the data is simply not read back from SYSUT1. However, such data is taking up space on SYSUT1.

AUXILIARY SPACE ON LINKLIB AND PROCLIB

The following list shows the number of tracks needed for the Assembler H load modules on SYS1.LINKLIB (or a private library) when the system uses the OS/VS Linkage Editor or Loader. The PROCLIB uses approximately 1 track regardless of device type.

Number of Directory Blocks: 2

Number of Tracks Required for LINKLIB:

```
3330 DASD - 19
3340 DASD - 29
3350 DASD - 13
3375 DASD -  8
3380 DASD -  6
```

PART 3. CMS INFORMATION

CHAPTER 7. ASSEMBLER LANGUAGE PROGRAMMING UNDER CMS

This chapter is for programmers who code in the assembler language under CMS (Conversational Monitor System). It is intended to help you assemble and execute your program, to choose and specify the options you need, and to interpret the listing and the diagnostic messages issued by the assembler. To use this section effectively, you should be familiar with the assembler language described in Assembler H Version 2 Application Programming: Language Reference.

This chapter is composed of the following major sections:

- "Introduction" describes the relationship of the assembler to CMS, and the input for and output of the assembler.
- "CMS Management of Your Assembly" describes how CMS manages the processing of permanent and temporary files created during assembly.
- "Creating an Assembler Language Program: CMS Editor" describes how you create an assembler language program using the CMS editor. This section also describes how to define an OS/VS data set as a CMS file.
- "Using Macros" refers you to another manual for a description of CMS Assembler macros, and describes how to add macro definitions to a macro library and specify the order in which those macro libraries are searched.
- "Assembling Your Program: HASM Command" describes the format of the CMS HASM command.
- "Assembler Options for CMS" describes how you use the assembler options when you assemble your program.
- "Assembler Data Sets and Storage Requirements" describes the assembler data sets and storage requirements of the assembler.
- "Loading and Executing Your Assembled Program" describes the commands for execution and for executing more than one module in an assembly. This section also describes CMS register usage during program execution and how parameters are passed to the program. Finally, this section tells you how to create a module of your program, so that it will execute when you invoke its file name on the command line.
- "Programming Aids" supplies information about the SYSTERM listing, and about the diagnostic messages generated by CMS.

RELATIONSHIP OF ASSEMBLER TO CMS

The assembler language program can be executed under control of CMS. This assembler program is the same as that supplied with the OS/VS systems. When you are using CMS, VM/SP CP Command Reference for General Users, VM/SP CMS Command and Macro Reference, or VM/Extended Architecture Migration Aid Command, Macro, and Diagnose Code Reference should be used for more detailed information about CMS.

INPUT

As input, the assembler accepts a program written in assembler language (as defined in the Glossary). This program is referred to as a source module.

OUTPUT

The output from the assembler consists of an object module and a program listing. The object module is stored on your virtual disk in a TEXT file. You can bring it into your virtual storage and execute 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."

CMS MANAGEMENT OF YOUR ASSEMBLY

When you assemble a program under CMS, permanent and temporary files are created and CMS performs certain processing steps. This section describes how CMS manages this processing.

FILES CREATED DURING ASSEMBLY

During the assembly of your program, files are created by CMS. Some files are permanent, others temporary. The permanent files are:

- An ASSEMBLE file, which is the source code used as input by the assembler
- The LISTING file, which contains the listing produced by the assembler, describing the results of the assembly
- The TEXT file, which contains the object code created during the assembly

A temporary file, SYSUT1, is created during assembly. It is used as a work file during assembly of your program. Figure 18 on page 60 shows input to the assembler and its output.

The utility files are placed on the read/write disk with the most available write space.

The TEXT and LISTING files are placed on one of three possible disks, if they are available:

- The disk on which the source file resides
- The parent disk of the above disk (if it exists)
- The primary disk

If all three attempts fail to place the information on a read/write disk, the assembly will terminate with an error message.

FILE PROCESSING BY THE ASSEMBLER

When assembling under CMS, two new files are created, each with the file name of the source ASSEMBLE file, but with file types of TEXT and LISTING. During assembly, any files residing on the virtual disk being processed, with the file name of the file you are processing and file types of TEXT or LISTING, will be erased. Unless you specify otherwise, the new TEXT and LISTING files created during assembly take their place on your processing disk. These files are erased even if you specify via NOOBJECT and NOLIST that there will be no new files to replace them.

CMS also defines a utility file for your assembly, thus eliminating the need for you to define it. At the end of assembly, the utility file is erased.

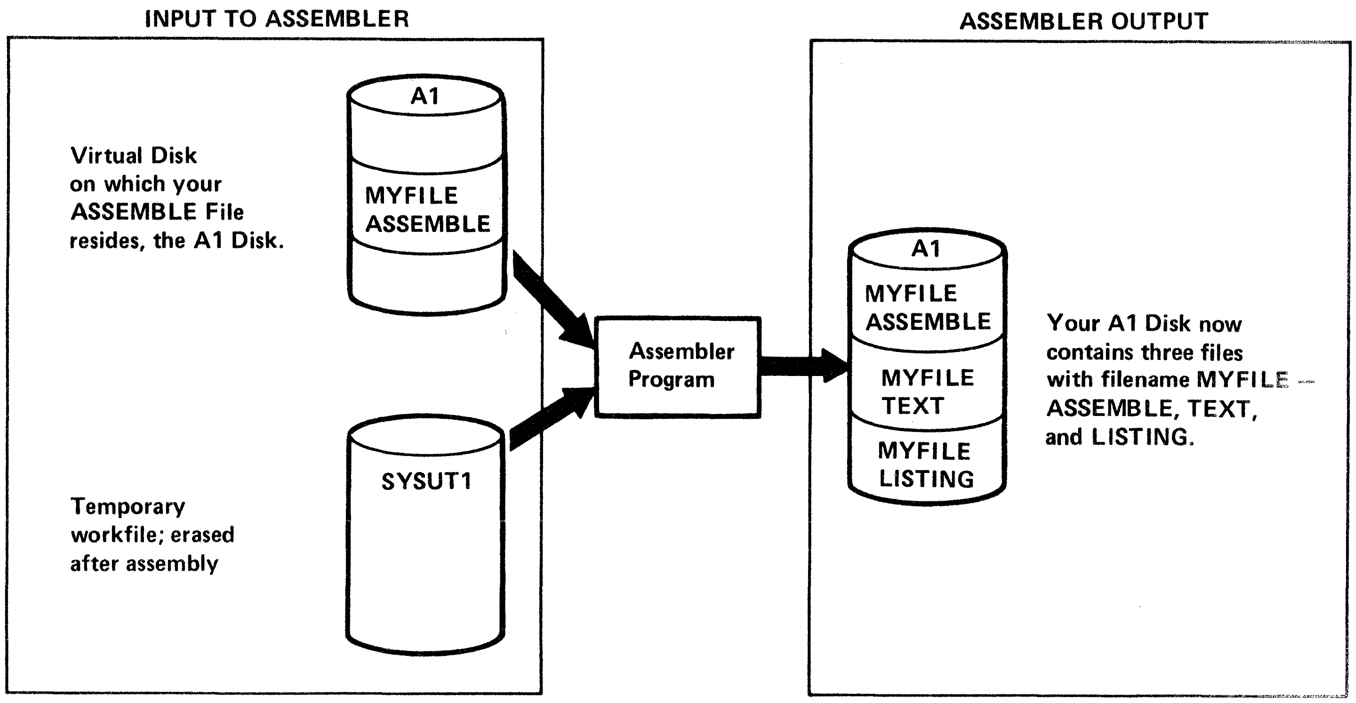

INPUT TO ASSEMBLER
A1
Virtual Disk on which your ASSEMBLE File resides, the A1 Disk.
MYFILE ASSEMBLE
SYSUT1
Temporary workfile; erased after assembly
Assembler Program
ASSEMBLER OUTPUT
A1
MYFILE ASSEMBLE
MYFILE TEXT
MYFILE LISTING
Your A1 Disk now contains three files with filename MYFILE — ASSEMBLE, TEXT, and LISTING.

Figure 18. Files Created during Assembly

CREATING AN ASSEMBLER LANGUAGE PROGRAM: CMS EDITOR

To create an assembler language program using CMS, you can use the CMS EDIT command. The EDIT command invokes the CMS editor, which provides an interactive environment for program creation, including subcommands that allow you to perform such functions as inserting and deleting lines and automatic tab setting. When you create an assembler language program under CMS, the EDIT command is entered in the following form:

```
  EDIT   filename ASSEMBLE
```

where filename is the name of your file. You must ensure that you enter a filetype of ASSEMBLE, thus specifying to the editor (and CMS) that you are creating an assembler language program. You can find a complete description of the editor and its facilities in VM/SP CMS Command and Macro Reference or VM/Extended Architecture Migration Aid Command, Macro, and Diagnose Code Reference.

When you have created your assembler language program, you use the CMS HASM command to invoke the assembler program to assemble your program file.

OVERRIDING HASM FILE DEFAULTS

When you issue the HASM command, default FILEDEF commands are issued for assembler data sets. You may want to override these with explicit FILEDEF commands. The ddnames used are:

SYSIN	Input to the assembler
SYSLIB	Macro/COPY library

SYSUT1	Utility work file
SYSPUNCH	Object module output
SYSLIN	Object module output
SYSPRINT	Listing output
SYSTERM	Diagnostic output

The default FILEDEF commands issued by HASM for these ddnames are:

```
FILEDEF SYSLIN DISK fn ASSEMBLE * (RECFM FB LRECL 80 BLOCK 3200
FILEDEF SYSLIB DISK CMSLIB MACLIB * (RECFM FB LRECL 80 BLOCK 3200
FILEDEF SYSUT1 DISK fn SYSUT1 m4 (BLOCK 4000
FILEDEF SYSPUNCH PUNCH
FILEDEF SYSLIN DISK fn TEXT m1
FILEDEF SYSPRINT DISK fn LISTING m1 (RECFM FB BLOCK 121
FILEDEF SYSTERM TERMINAL
```

In the FILEDEFs for SYSUT1, SYSLIN, and SYSPRINT, the file modes 'm4' and 'm1' are established dynamically by the HASM command processor as follows:

In the FILEDEF for SYSUT1, the file mode 'm4' is set to use the read/write disk with the most available space. For example, if three read/write disks were accessed as the A, B, and D disks, and if the D disk had the most available space, then 'm4' would be set to 'D4' for use during the assembly.

In the FILEDEFs for SYSLIN and SYSPRINT, if the assembler source file (SYSIN input) is <u>not</u> on disk or is on a read-only disk, the file mode 'm1' is set to 'A1'. If the source file is on a read/write disk, the mode letter 'm' is set to the mode of that read/write disk. For example, if the source file were on a read/write B disk, the file mode 'm1' would be set to 'B1'.

A FILEDEF command, issued to any of the above ddnames prior to invoking the assembler, overrides the default FILEDEF issued by the HASM command processor. Assume that there is an assembler source file in card deck form that you want to assemble. If you have this card deck available to your CMS card reader, you could issue an overriding FILEDEF command prior to assembling; that is, FILEDEF SYSIN READER. Now you can invoke the assembler as follows:

```
  HASM SAMPLE (options....
```

The name SAMPLE is used by the HASM as the file name for any TEXT or LISTING files produced by the assembler. An existing TEXT and/or LISTING file on your read/write A-disk would be replaced by new versions created by the HASM command processor.

Similarly, if you have a tape containing an assembler input file that you want to assemble, you must issue the following command:

```
  FILEDEF SYSIN TAPn (RECFM F LRECL 80 BLOCK 80
```

or, if the file were blocked 80x800, you could specify BLOCK 800 in the preceding FILEDEF. In either case, the FILEDEF would be followed by the command HASM SAMPLE (options....

You can read OS/VS data sets on CMS files by defining those data sets with the FILEDEF command. For example,

```
  FILEDEF SYSIN DISK OSDS ASSEMBLE fm
                DSN OS DATASET (options...
  HASM (options...
```

It is also possible to assemble a member of an OS/VS partitioned data set by using the MEMBER parameter of the FILEDEF command.

The same techniques used in these examples can be applied to other ddnames. Care should be taken that any attributes specified for a file conform to the attributes expected by the assembler for the device.

USING MACROS

ASSEMBLER MACROS SUPPORTED BY CMS

There are several 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 VM/SP CP Command Reference for General Users or the appropriate VM/Extended Architecture Migration Aid manual.

MACRO DEFINITION LIBRARY ADDITIONS

Source statement coding, to be retrieved by the COPY assembler instruction, and macro definitions may be added to a macro library. The CMS MACLIB command is used to create and modify CMS macro libraries. Details of this command are contained in VM/SP CMS Command and Macro Reference or the appropriate VM/Extended Architecture Migration Aid manual.

SPECIFYING MACRO LIBRARIES

The GLOBAL command is used to identify which CMS libraries are to be searched for macro definitions and COPY code. Private libraries and CMSLIB may be concatenated with each other in any order by the GLOBAL command. The format of this command is described in VM/SP CMS Command and Macro Reference or the appropriate VM/Extended Architecture Migration Aid manual.

ASSEMBLING YOUR PROGRAM: HASM COMMAND

Once you have created or defined a source program, you assemble the program using the CMS HASM command. This command invokes the assembler program. This section describes how you use HASM.

HASM COMMAND FORMAT

You use the HASM command to create an object file from a source file. The source program can be created by the CMS editor, or it can be created externally and defined for use under CMS by the FILEDEF command. HASM takes the following form:

```
HASM filename (options[)]
```

where 'filename' is the name of the file you are assembling and 'options' is a series of keywords used to specify functions associated with the assembler. The options are described in "Assembler Options for CMS" on page 63.

File Name Entry

When your file has been created by the CMS editor, you use the file name associated with the file when you issue the HASM command. If your file has been defined for use under CMS by the FILEDEF command, you use a dummy or unique file name to be used by the assembler to define the LISTING and TEXT files the assembler produces. You need not enter the standard CMS file-type field, since the default file type is ASSEMBLE.

ASSEMBLER OPTIONS FOR CMS

HASM offers a number of optional facilities. For example, you can suppress printing of your assembly listing or parts of the listing, and you can specify whether you want an object deck or an object module. You select the options by including appropriate keywords in the HASM command that invokes the assembler. There are three types of options:

- Simple pairs of keywords: a positive form (such as OBJECT) that requests a facility, and an alternative negative form (such as NOOBJECT) that rejects that facility.
- Keywords that permit you to assign a value to a function (such as LINECOUN(50)).
- HASM command processor options (such as PRINT) which are not passed to Assembler H but are used to control certain aspects of the assembly process. Such options are referred to in later sections as "CMS options" to distinguish them from Assembler H options.

Each of these options has a standard or default value that is used for the assembly if you do not specify an alternative value. The default values are discussed in "Command Defaults" below.

The HASM command processor combines all the assembler options into a string of characters with a comma separating each option. This string is passed to the assembler when it is invoked. If n options are specified (n>1), then n-1 commas are inserted. The total number of characters in the assembler options plus the number of inserted commas must not be greater than 100. The CMS options are not included in this count. You may specify the options in any order. If contradictory options are used (for example, LIST and NOLIST), the rightmost option (in this case, NOLIST) is used.

The command options are described under "Command Format."

COMMAND DEFAULTS

If you do not code a given option in the HASM command, a default option will be assumed. The following default options are included when HASM is shipped by IBM:

DECK, NOOBJECT, LIST, XREF(FULL), NORENT, NOTEST, NOBATCH, ALIGN, ESD, RLD, LINECOUN(55), FLAG(0), SYSPARM(), DISK, NUMBER, NOSTMT, NOTERM

However, these may _not_ be the default options in effect at your installation. The defaults could have been respecified when HASM was installed. For example, RENT could be made the default in place of NORENT. Also, a default option can be specified during installation so that you cannot override it. Similar considerations apply to the assembler ddnames for which the HASM command processor issues FILEDEFs. In the description of the HASM command, the options and ddnames specified as being "default values" are those included when HASM is shipped by IBM.

You should determine which default values are in effect at your installation and whether there are any you cannot override.

COMMAND FORMAT

The HASM command is used to invoke Assembler H to assemble a specified file. HASM processing and output are controlled by the options selected. The format of the HASM command follows:

```
  HASM [fn] [(options...[)]]
```

where:

fn
is the file name of the source file to be assembled. The file specified must consist of fixed-length, 80-character records. If a user-issued FILEDEF for SYSIN is active, and if the FILEDEF specified DISK, the file name may be omitted. If the user FILEDEF specified TAPn or READER, a "dummy" file name must be supplied and is used to name the TEXT and LISTING files. If no user FILEDEF for SYSIN is active, the source file must exist on an ACCESSed disk and must have a file type of ASSEMBLE.

options
(see below)

Listing Control Options:

```
[ESD|NOESD]
[FLAG (0)|FLAG (nnn)]
[LIST|NOLIST]
[LINECOUN (55)|LINECOUN (nn)]
[RLD|NORLD]
[XREF (FULL)|XREF (SHORT)|NOXREF]
[DISK|PRINT|NOPRINT]
```

Object Module Control Options:

```
[DECK|NODECK]
[OBJECT|NOOBJECT]
[TEST|NOTEST]
```

SYSTERM Options:

```
[NUMBER|NONUMBER]
[STMT|NOSTMT]
[TERMINAL (0)|TERMINAL (n)|NOTERM]
```

Other Options:

```
[ALIGN|NOALIGN]
[BATCH|NOBATCH]
[RENT|NORENT]
[SYSPARM ()|SYSPARM (string)|SYSPARM(?)]
```

The list below, in alphabetic order, describes the assembler options you can use to control the assembler listing.

ALIGN|NOALIGN

ALIGN
The assembler does not suppress the alignment error diagnostic message; all alignment errors are diagnosed.

NOALIGN
The assembler suppresses the diagnostic message "IEV033 ALIGNMENT ERROR" if fixed-point, floating point, or logical data referred to by an instruction operand is not aligned on the proper boundary. The message will be produced, however, for references to instructions that are not aligned on the proper (halfword) boundary or for data boundary violations for privileged instructions such as LPSW. In addition, DC, DS, DXD, or CXD constants, usually causing alignment, are not aligned.

Default: ALIGN

BATCH|NOBATCH

BATCH
The assembler will do multiple (batch) assemblies under the control of a single HASM command. The source decks must be placed together in one file. The TEXT file produced will contain multiple object decks. The LISTING file produced will contain multiple listings.

NOBATCH
The BATCH option is suppressed.

Default: NOBATCH

DECK|NODECK

DECK
The object module is placed on the SYSPUNCH device.

NODECK
The DECK option is suppressed.

Default: DECK

ESD|NOESD

ESD
The assembler produces the external symbol dictionary as part of the listing.

NOESD
No ESD listing is printed.

Default: ESD

FLAG(nnn)

Error diagnostic messages below severity code nnn will not appear in the listing nor on the SYSTERM device. Diagnostic messages can have severity codes of 0, 4, 8, 12, 16, or 20 (0 is the least severe). MNOTEs can have a severity code of 0 through 255.

For example, FLAG(8) will suppress messages for severity codes 0 through 7.

Default: 'FLAG(0)'

LINECOUN(nn)

The number of lines to be printed between headings in the listing is nn. The permissible range is 1 to 99 lines.

Note: The heading occupies 5 of these lines.

Default: 'LINECOUN(55)'

LIST|NOLIST

LIST
An assembler listing is produced. Note that the NOLIST option overrides the ESD, RLD, XREF, DISK, and PRINT options. In addition, no diagnostic information will be written on the SYSTERM device.

NOLIST
No assembler listing is produced. This option overrides ESD, RLD, XREF, and LINECOUN.

Default: LIST

NUMBER|NONUMBER

NUMBER|NUM
This CMS option writes the line number field (columns 73 to 80 of the input records) on the SYSTERM device for statements for which diagnostic information is produced.

NONUMBER|NONUM
This CMS option suppresses the NUMBER option.

Default: NUMBER

OBJECT|NOOBJECT

OBJECT
The object module is placed on the SYSLIN device.

NOOBJECT
The OBJECT option is suppressed.

Default: NOOBJECT

Note: The OBJECT and DECK options are independent of each other. Both or neither can be specified. The output on SYSLIN and SYSPUNCH is identical, except that the control program closes SYSLIN with a disposition of LEAVE, and SYSPUNCH with a disposition of REREAD.

PRINT|NOPRINT|DISK

PRINT|PR
This CMS option writes the LISTING file on the printer. The LISTING file is not written on disk.

NOPRINT|NOPR
This CMS option suppresses the writing of the LISTING file. Any assembler diagnostic messages to be written to the SYSTERM device are not affected.

DISK|DI
This CMS option writes the LISTING on a virtual disk.

Default: DISK

RENT|NORENT

RENT
The assembler checks for a possible coding violation of program reenterability. Code that makes your program nonreentrant is identified by an error message, but it cannot be an exhaustive check as the assembler cannot check the logic of the code. Therefore, it is possible to have nonreentrant code not flagged.

NORENT
The RENT option is suppressed.

Default: NORENT

RLD|NORLD

RLD
The assembler produces the relocation dictionary as part of the listing.

NORLD
The relocation dictionary is not printed.

Default: RLD

STMT|NOSTMT

STMT
This CMS option writes the statement number assigned by the assembler on the SYSTERM device for those statements for which diagnostic information is produced.

NOSTMT
This CMS option suppresses the STMT option.

Default: NOSTMT

SYSPARM(string)

'string' is the value of the system variable symbol &SYSPARM. The assembler uses &SYSPARM as a read-only SETC variable. If no value is specified for the SYSPARM option, &SYSPARM will be a null (empty) character string.

In the CMS environment, 'string' cannot be longer than 8 characters. If you wish to enter a string of more than 8 characters, use the SYSPARM(?) format. Using this form, you will be prompted at your terminal with the message:

```
ENTER SYSPARM:
```

You may then enter as many characters as you want up to the option limit of 100 characters. It is also necessary to use the SYSPARM(?) form to enter parentheses and/or embedded blanks in 'string'.

Default: 'SYSPARM()'

TERMINAL(n)|NOTERM

TERMINAL(n)|TERM(n)
This CMS option provides the ability to stop diagnostic information of a given severity from being written on the SYSTERM device. The value of n is a decimal number between 0 and 7. The value of n can be thought of as a 3-bit binary number, and it is this 3-bit "mask" that serves as the diagnostic message filter. Consider the 3 bits to be labeled b0, b1, b2 from left to right. Then, the following apply:

b0 = 1	suppress 'ERROR' diagnostics
b1 = 1	suppress 'WARNING' diagnostics
b2 = 1	suppress 'MNOTE' diagnostics

For example, TERM(4) will suppress ERROR diagnostics, and TERM(5) will suppress ERROR and MNOTE diagnostics.

NOTERM
This CMS option suppresses the writing of all diagnostic information on the SYSTERM device. NOTERM has the same effect as the option TERM(7).

Default: NOTERM

TEST|NOTEST

TEST
The object module contains the special source symbol table (SYM cards).

NOTEST
The special source symbol table is not produced.

Default: NOTEST

XREF(FULL|SHORT)|NOXREF

XREF(FULL)
The assembler listing contains a cross-reference table of all symbols used in the assembly. This includes symbols that are defined but never referenced. The assembler listing also contains a cross-reference table of literals used in the assembly.

XREF(SHORT)
The assembler listing contains a cross-reference table of all symbols that are referred to in the assembly. Any symbols defined but not referred to are not included in the table. The assembler listing also contains a cross-reference table of literals used in the assembly.

NOXREF
No cross-reference tables are printed.

Default: 'XREF(FULL)'

ASSEMBLER DATA SETS AND STORAGE REQUIREMENTS

This section describes the data set used by the assembler. It also describes the main storage and auxiliary storage requirements of the assembler. This description is intended for programmers who want to alter the assembler's region size or data set parameters.

ASSEMBLER DATA SETS FOR CMS USERS

This section describes the data sets used by the assembler to assemble your program under CMS; these data sets are referred to as files.

DDname SYSUT1

The assembler uses this utility data set as an intermediate external storage device when processing the source program. This data set must be organized sequentially, and the device assigned to it must be a direct-access device.

DDname SYSIN

This data set contains the input to the assembler—the source statements to be processed. The input device assigned to this data set may be DISK, READER, or TAPn, or another sequential input device that you have designated. The FILEDEF command describing this data set appears in the input stream.

DDname SYSLIB

From this data set, whose file type must be MACLIB, the assembler obtains macro definitions and assembler language statements that can be called by the COPY or a macro assembler instruction. It is a partitioned data set: Each macro definition or sequence of assembler language statements is a separate member, with the member name being the macro instruction mnemonic or COPY code name. The data set may be CMSLIB or a private macro library. OSMACRO contains macro definitions for the IBM-supplied OS macro instructions supported by CMS. DMSSP contains macro definitions for the IBM-supplied CMS macro instructions for VM/SP. Private libraries and CMSLIB can be concatenated with each other in any order by the GLOBAL command.

DDname SYSPRINT

This data set is used by the assembler to produce a listing. Output may be directed to a printer, a magnetic tape, or a direct-access storage device. The default device is DISK. The assembler uses the American National Standard Institute (ANSI) carrier-control characters for this data set. The smallest block size recommended is 1089 bytes (with a blocking factor of 9).

DDname SYSPUNCH

The assembler uses this data set to produce a punched copy of the object module. The output unit assigned to this data set may be either a card punch or an intermediate storage device capable of sequential access. The object module is placed on the SYSPUNCH device if the assembler option DECK is specified.

DDname SYSLIN

This is a direct-access storage device or a magnetic tape data set used by the assembler. It contains the same output text (object module) as SYSPUNCH. It is used as input for the CMS LOADER. The object module is placed on the SYSLIN device if the assembler option OBJECT is specified.

DDname SYSTERM

This data set is used by the assembler to produce diagnostic information. The output may be directed to a remote terminal, a printer, a magnetic tape, or a direct-access storage device. The assembler uses the ANSI carrier-control characters for this data set. The smallest block size recommended is 1089 bytes (with a blocking factor of 9).

ASSEMBLER VIRTUAL STORAGE REQUIREMENTS

The minimum size virtual machine required by the assembler is 344K bytes. However, better performance is generally achieved if the assembler is run in 396K bytes of virtual storage. This size is recommended for medium and large assemblies.

If more virtual storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to work space, larger and more complex macro definitions can be handled.

LOADING AND EXECUTING YOUR ASSEMBLED PROGRAM

Once you have assembled your program file, you can load and execute the resulting TEXT file (containing object code) using the CMS LOAD and START commands. The LOAD command causes your TEXT file to be loaded into storage in your virtual machine and the START command begins execution of the program. If you are assembling more than one file, use the CMS INCLUDE command to bring the additional files into storage. These commands and the options associated with them are described in VM/SP CP Command Reference for General Users or the appropriate VM/Extended Architecture Migration Aid manual.

CMS REGISTER USAGE DURING EXECUTION OF YOUR PROGRAM

CMS reserves four registers for its own use during the execution of an assembler language program. When control is received from the user program, the entry point address for the program is placed in register 15. Register 1 contains the address of a parameter list, which contains any parameters passed to the program. Register 13 contains the address of the save area. Register 14 contains the section address to return control to the control program.

PASSING PARAMETERS TO YOUR ASSEMBLER LANGUAGE PROGRAM

CMS provides you with the ability to pass parameters to an assembler language program by means of the START command. The statement below shows how to pass parameters to your program using the CMS START command:

```
  START MYJOB PARM1 PARM2
```

The parameters must be no longer than 8 characters each, and must be separated by blanks.

CMS creates a list of the parameters for use during execution. The parameter list for the command above would look like:

```
  PLIST DS 0D
        DC CL8'MYJOB'
        DC CL8'PARM1'
        DC CL8'PARM2'
        DC 8X'FF'
```

where the list is delimited by hexadecimal FFs.

CREATING A MODULE OF YOUR PROGRAM

When you are sure that your program executes properly, you may want to create a module of it, so that you can execute it by simply invoking its file name on the command line.

To create a module, you use the LOAD, GENMOD, and, in some cases, the LOADMOD commands. For more information, see the section in VM/SP CP Command Reference for General Users or the appropriate VM/Extended Architecture Migration Aid manual on creating a module.

PROGRAMMING AIDS

This section contains reference information about the assembler. It describes the SYSTERM listing and the diagnostic messages generated by CMS.

CMS SYSTERM LISTING

The SYSTERM data set is used by the assembler to store a summary form of SYSPRINT containing flagged statements and their associated messages.

You use the assembler option TERMINAL(n) to specify that you want a SYSTERM listing to be produced.

Each diagnosed statement in the assembly listing printed in the SYSTERM listing is immediately followed by its associated error message. If there are multiple error messages associated with a source statement, the source statements will be listed once for each error message. To help identify the position of the statement in your program, two additional assembler options are available:

- NUMBER, which prints the line number(s) of the diagnosed statement
- STMT, which prints the statement number assigned to the diagnosed statement by the assembler

The format of the flagged statement as it appears in the listing is:

Name	Operation	Operand
Line No. (option NUMBER)	Statement No. (option STMT)	Source Records (columns 1-72 of the source statement lines)

DIAGNOSTIC MESSAGES WRITTEN BY CMS

If an error occurs during execution of the HASM command, a message may be typed at the terminal and, at completion of the command, register 15 contains a nonzero return code.

There are two types of messages that may be issued:

- Messages that are issued by the assembler (see Appendix D, "Assembler H Messages" on page 100)
- Messages that are issued directly by the HASM command processor (refer to the following section)

The messages issued directly by the HASM command processor are in two parts: a message code and the message text. The message code is in the form 'IEVCMSnnnt', where IEVCMS indicates that the message was generated by the HASM command program, nnn is the number of the message, and t is the type of message. The message text describes the error condition.

The actual message typed may not be complete. By using the CP SET (EMSG) command, the user can specify that the entire error message be typed, or only the error code, or only the text, or neither code nor text. VM/SP CP Command Reference for General Users or the appropriate VM/Extended Architecture Migration Aid manual contains a description of the CP SET command.

Unless NOTERM is specified, diagnostic and error messages originating in the assembler are typed at the terminal in the form IEVnnn text. Errors detected by the HASM command program, which terminate the command before Assembler H is called, result in error messages (type E).

For additional information about the text, format, or codes in the messages for HASM, see VM/SP System Messages and Codes or VM/Extended Architecture Migration Aid System Messages and Codes.

HASM COMMAND ERROR MESSAGES

IEVCMS002E FILE 'fn ft fm' NOT FOUND.

Explanation: The filename you included in the HASM command does not correspond to the names of any of the files on your disks.

Supplemental Information: The variable filename, filetype, and filemode in the text of the message indicate the file that could not be found.

System Action: RC=28. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the HASM with an appropriate filename.

IEVCMS003E INVALID OPTION 'option'.

Explanation: You have included an invalid option with your HASM command.

Supplemental Information: The variable option in the text of the message indicates the invalid option.

System Action: RC=24. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the HASM command, and reissue the command with the correct option.

IEVCMS004E IMPROPERLY FORMED OPTION 'option'.

Explanation: You have included an improperly formed option with your HASM command.

Supplemental Information: The variable option in the text of the message indicates the improperly formed option.

System Action: RC=24. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the HASM command, and reissue the command with the correct option.

IEVCMS006E 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. Execution 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.

IEVCMS007E FILE 'fn ft fm' DOES NOT CONTAIN FIXED LENGTH 80 CHARACTER RECORDS.

Explanation: The source file you specified in the HASM command does not contain fixed-length records of 80 characters.

Supplemental Information: The variable filename, filetype, and filemode in the text of the message indicate the file that is in error.

System Action: RC=32. The command cannot be executed.

Programmer Response: You must reformat your file into the correct record length. CMS EDIT or COPYFILE can be used to reformat the file.

IEVCMS010E FILENAME OMITTED AND DDNAME 'SYSIN' IS UNDEFINED.

Explanation: You have not included a filename in the HASM command, and no FILEDEF could be found for the ddname specified.

System Action: RC=24. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the HASM command and specify a filename, or issue a FILEDEF for the ddname specified.

IEVCMS011E FILENAME OMITTED AND FILEDEF 'SYSIN' IS NOT FOR DISK.

Explanation: You have not included a filename in the HASM command, and the FILEDEF for the ddname specified is not for disk.

System Action: RC=24. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the HASM command and specify a filename, or reissue the FILEDEF for the ddname specified with a device type of 'DISK'.

IEVCMS038E FILEID CONFLICT FOR DDNAME 'SYSIN'.

Explanation: You issued a FILEDEF command that conflicts with an existing FILEDEF for the ddname specified.

Supplemental Information: The variable ddname in the text of the message indicates the ddname in error.

System Action: RC=40. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the FILEDEF command with an appropriate ddname.

IEVCMS052E OPTIONS SPECIFIED EXCEED 100 CHARACTERS.

Explanation: The string of options that you specified with your HASM command exceeded 100 characters in length.

System Action: RC=24. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your HASM command with fewer options specified.

IEVCMS070E INVALID PARAMETER 'parm'.

Explanation: You specified an invalid parameter for an option in the HASM command.

Supplemental Information: The variable parameter in the text of the message indicates the invalid parameter.

System Action: RC=40. Execution 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 appropriate parameters, and reissue the command with the correct parameter.

IEVCMS074E ERROR {SETTING|RESETTING} AUXILIARY DIRECTORY.

Explanation: One of two conditions causes this message to be generated:

1. The disk containing the assembler modules (that is, the disk specified at auxiliary directory generation by means of the GENDIRT mode field) has not been accessed.
2. An attempt to reset the file status table has failed, thereby removing the auxiliary directory from the search chain. Either the auxiliary directory was not included in the file status table chain, or a processing error has caused the disk containing the assembler modules to appear to be not accessed.

System Action: RC=40. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Verify that the disk containing the assembler modules has been accessed using the proper mode specification (that is, the mode specified by means of the GENDIRT mode field when the auxiliary directory was generated). If the error occurred resetting the auxiliary directory, contact installation maintenance personnel.

IEVCMS075E DEVICE 'device' INVALID FOR INPUT.

Explanation: The device specified in your FILEDEF command cannot be used for the input operation that is requested in your program. For example, you have tried to read data from the printer.

Supplemental Information: The variable device name in the text of the message indicates the incorrect device that was specified.

System Action: RC=40. Execution of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your FILEDEF command, specifying an appropriate device for the desired input operation.

CHAPTER 8. PROGRAMMING CONSIDERATIONS

This chapter discusses various topics in assembler language programming.

SAVING AND RESTORING GENERAL REGISTER CONTENTS

A problem program should save the values contained in the general registers upon commencing execution and, upon completion, restore to the general registers these same values. Thus, as control is passed from the operating system to a problem 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.

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 problem program is given control, register 13 contains the address of an area in which the general contents should be saved.

If the program calls any subprograms, or uses any operating system services other than GETMAIN, FREEMAIN, ATTACH, and XCTL, it must first save the contents of register 13 and then load the address of an 18-fullword save area into register 13. This save area is in the problem program and is used by any subprograms or operating system services called by the problem program.

At completion, the problem 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 execution of the RETURN macro instruction.

The coding sequence that follows illustrates 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 appropriate Supervisor Services and Macro Instructions.

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	

PROGRAM TERMINATION

You indicate completion of an assembler language source program by using the RETURN system macro instruction to pass control from the terminating program to the program that initiated it. The initiating program may be the operating system or, if a subprogram issued the RETURN, the program that called the subprogram.

In addition to indicating program completion and restoring register contents, the RETURN macro instruction may also pass a return code—a condition indicator that may be used by the program receiving control.

If the return is to CMS, the return code is displayed to the user.

If return is to another problem program, the return code is available in general register 15, and may be used as desired. Your program should restore register 13 before issuing the RETURN macro instruction.

The RETURN system macro instruction is discussed in detail in the appropriate Supervisor Services and Macro Instructions.

APPENDIX A. SAMPLE PROGRAM

The sample program included with Assembler H when it is received from IBM is described in this appendix. This program demonstrates some basic assembler language, macro, and conditional assembly features, most of which are unique to Assembler H. The letters in parentheses in the descriptions below refer to corresponding letters in the listing that precedes the descriptions.

```
BIGNAME                                 EXTERNAL SYMBOL DICTIONARY                                   PAGE    1

SYMBOL   TYPE  ID   ADDR  LENGTH  LD ID FLAGS                                          ASM H V 02 13.19 02/19/82

  A      SD 0001 000000 0000DC         00
(A) PD2  CM 0002 000000 0007D2         00
```

```
BIGNAME  SAMPLE PROGRAM.  1ST TITLE STATEMENT HAS NO NAME, 2ND ONE DOES                              PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  ASM H V 02 13.19 02/19/82

000000                                2 A        CSECT                                                        00100000
                            00000     3          USING *,8                                                    00150000

                                      5 ************************************************************************ 00250000
                                      6 *                 PUSH  AND POP  STATEMENTS                            * 00300000
                               (B)    7 * PUSH DOWN THE PRINT STATEMENT, REPLACE IT, RETRIEVE ORIGINAL       * 00350000
                                      8 ************************************************************************ 00400000

                                     10          PUSH  PRINT     SAVE DEFAULT SETTING '  PRINT ON,NODATA,GEN'  00500000
                                     11          PRINT NOGEN,DATA                                               00550000
                                     12          WTO   MF=(E,(1))                      EXPANSION NOT SHOWN      00600000
000002 01230ABC0102030A        (C)   14  DC X'123,ABC',(REALLYLONGSYMBOL-TRANSYLVANIA)B'1,10,11,1010,1011,1100' 00650000
00000A 0B0C0102030A0B0C
000012 0102030A0B0C0102
00001A 030A0B0C
                                     15          POP   PRINT                     RESTORE DEFAULT PRINT SETTING  00700000
                                     16          WTO   MF=(E,(1))                      EXPANSION SHOWN          00750000
00001E 0A23                          17+         SVC   35                                  ISSUE SVC         01-WTO
000020 01230ABC0102030A              18  DC X'123,ABC',(REALLYLONGSYMBOL-TRANSYLVANIA)B'1,10,11,1010,1011,1100' 00800000

                                     20 ************************************************************************ 00900000
                               (D)   21 *                     LOCTR  INSTRUCTION                               * 00950000
                                     22 * LOCTR ALLOWS 'REMOTE' ASSEMBLY OF CONSTANT                           * 01000000
                                     23 ************************************************************************ 01050000

00003C 5850 8098            00098    25          L     5,CONSTANT                                               01150000
000098                               26 DEECEES  LOCTR                                                          01200000
000098 00000005                      27 CONSTANT DC    F'5'      CONSTANT CODED HERE, ASSEMBLED BEHIND LOCTR A  01250000
000040                               28 A        LOCTR                     RETURN TO 1ST LOCTR IN CSECT A       01300000

                               (E)   30 ************************************************************************ 01400000
                                     31 *  3 OPERAND EQUATE WITH FORWARD REFERENCE IN 1ST OPERAND              * 01450000
                                     32 ************************************************************************ 01500000

000040 1812                          34 A5       LR    1,2               L'A5 = 2, T'A5 = I                    01600000
                                     35          PRINT DATA                                                     01650000
000042 000000000000
000048 413243F6A8885A30              36 A7 DC L'3.14159265358979323846264338327950288841972' L'A7 = 16,T'A7 = L  01700000
000050 338D313198A2E037
                                     37 &TYPE    SETC  T'A7                                                     01750000
                                     38 A8       EQU   B5,L'A5,C'&TYPE'                                         01800000
                            000A0      +A8       EQU   B5,L'A5,C'L'                                             01800000
```

(A) The external symbol dictionary shows a named common statement. The named common section is defined in statement 158.

(B) Statement 10: Save the current status of the PRINT statement (ON,NODATA,GEN).

Statement 11: Leave ON in effect, modify the other two options to DATA,NOGEN.

Statement 12: Macro call; note that the expansion (statement 10) is not printed.

Statement 14: All 28 bytes of data are displayed to the two-operand DC.

Statement 15: Restore prior status of PRINT.

Statements 17 and 18: The generated output of the macro WTO is shown and only the first 8 bytes of the data are displayed.

(C) Statements 14 and 18: 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 108 and 109.

(D) Statements 26, 28, 136, and 155 illustrate use of the LOCTR assembler instruction. This feature allows one to break control sections down into subcontrol sections. It may be used in CSECT, DSECT, and COM. LOCTR has many of the features of a control section; for example, all of the first LOCTR in a section is assigned space, then the second, and so on. The name of the control section automatically names the first LOCTR section. Thus LOCTR A is begun, or resumed, at statements 2, 28, and 155. Note that the location counter value shown each time is the resumed value of the LOCTR. On the other hand, various LOCTR sections within a control section have common addressing as far as USING statements are concerned, subject to the computed displacement falling within 0 through 4095. In the sample, CONSTANT is in LOCTR DEECEES but the instruction referring to it (statement 25) has no addressing problems.

(E) Three-operand EQU. Here, we are assigning: (a) the value of B5 (not yet defined) to A8, (b) the length attribute of A5 to A8, and (c) the type attribute of A7 to A8. If no operand is present in an EQU statement, the type attribute is U and the length attribute is that of the first term in the operand expression. Symbols present in the label and/or operand field must be previously defined. Note that it is not possible to express the type attribute of A7 directly in the EQU statement. The EQU statement at 38 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    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                       ASM H V 02 13.19 02/19/82
                                      40 ************************************************************************  01900000
                                      41 * IMPLICIT DECLARATION OF LOCALS &A, &C -- USE OF SETC DUP FACTOR TO *  01950000
                                      42 *   PRODUCE SETC STRING LONGER THAN 8, MNOTE IN OPEN CODE              *  02000000
                                      43 ************************************************************************  02050000

                                      45 &LA8     SETA  L'A8                                                       02150000
                                  F   46 &TA8     SETC  T'A8                                                       02200000
                                      47          MNOTE *,'LENGTH OF A8 = &LA8, TYPE OF A8 = &TA8'                 02250000
                                  G     +*,LENGTH OF A8 = 2, TYPE OF A8 = L                                        02250000

                                      49 &A       SETA  2                                                          02350000
                                  H   50 &C       SETC  (&A+3)'STRING,'                                            02400000
                                      51          MNOTE *,'&&C HAS VALUE = &C'                                     02450000
                                        +*,&C HAS VALUE = STRING,STRING,STRING,STRING,STRING,                      02450000

                                      53 ************************************************************************  02550000
                                  I   54 * EXAMPLES OF 4 BYTE SELF-DEFINED TERMS, UNARY + AND -                 *  02600000
                                      55 ************************************************************************  02650000

000058 7FFFFFFFC1C2C3C4               57          DC    A(2147483647,C'ABCD',X'FFFFFFFF')                          02750000
000060 FFFFFFFF
000064 181D                           58          LR    -1+2,16+-3                                                 02800000

                          FFFFFFE8    60 X        EQU   4*-6                                                       02900000
```

(F) Set symbols &LA8 and &TA8 have not been declared in an LCL or GEL statement prior to their use here. Therefore, they are defaulted 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 may appear in open code. As such, they have all properties of MNOTEs inside macros, including substitution.

(H) A SETC expression may have a duplicate 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 57 through 60 illustrate 4-byte self-defining values and unary + and -. The value of X will appear later in a literal address constant (see statement 162).

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    ASM H V 02 13.19 02/19/82
                                    62 ******************************************************************** 03000000
                                    63 *  MIXED KEYWORDS AND POSITIONAL PARAMETERS, EXTENDED AGO AND AIF    * 03050000
                                    64 *     STATEMENTS, DECLARATION AND USE OF SUBSCRIPTED SET SYMBOLS,    * 03100000
                                    65 *       USE OF CREATED SET SYMBOLS, EXTENDED SET STATEMENTS          * 03150000
                                    66 ******************************************************************** 03200000
                               (J)
                                    68          MACRO                                                         03300000
                                    69          DEMO  &P1,&KEY1=A,&P2,&KEY2=1,&P3,&KEY3=3,&P4                  03350000
                               (K)  70 &LOC(1)  SETC  '2','3'      &LOC IS DIMENSIONED LCLC BY DEFAULT         03400000
                                    71          GBLC  &XA(5),&XB(20),&XC(1)                                   03450000
                               (L)  72          &P1   &SYSLIST(4),&SYSLIST(5),&SYSLIST(6),MF=E                03500000
                                    73 &N       SETA  1                                                       03550000
                                    74          AGO   (&KEY2).MNOTE1,.MNOTE2,.MNOTE3                          03600000
                               (M)  75 &N       SETA  2                                                       03650000
                                    76          MNOTE *,'&&KEY2 NOT 1,2, OR 3---USE &&KEY3 IN PLACE OF IT'    03700000
                               (N)  77          AIF   (&KEY3 EQ 1).MNOTE1,                                   X03750000
                                                      (&KEY3 EQ 2).MNOTE2,(&KEY3 EQ 3).MNOTE3                 03800000
                                    78          MNOTE *,'BOTH &&KEY2 AND &&KEY3 FAIL TO QUALIFY'              03850000
                                    79          AGO   .COMMON                                                 03900000
                                    80 .MNOTE1  MNOTE *,'&&KEY&LOC(&N) = 1'                                   03950000
                                    81          AGO   .COMMON                                                 04000000
                                    82 .MNOTE2  MNOTE *,'&&KEY&LOC(&N) = 2'                                   04050000
                                    83          AGO   .COMMON                                                 04100000
                                    84 .MNOTE3  MNOTE *,'&&KEY&LOC(&N) = 3'                                   04150000
                                    85 .COMMON  L     5,8(,10)       NOTE THAT OPCODES, OPERANDS & COMMENTS   04200000
                               (O)  86 &XB(2) SR 9,10                  ON MODEL STATEMENTS                    04250000
                                    87 &(X&KEY1)(2) LM 12,13,=A(A5,X)    ARE KEPT IN PLACE UNLESS DISPLACED   04300000
                                    88 &P2 ST 7,&P3                        AS A RESULT OF SUBSTITUTION        04350000
                                    89          MEND                                                          04400000

                                    91 *****          DEMO  MACRO  INSTRUCTION (CALL)                         04500000
                               (P)
                                    93          GBLC  &XA(1),&XB(2),&XC(3)                                    04600000
                                    94 &XA(1)   SETC  'A','MISSISSIPPI'                                       04650000
                                    95 &XB(1)   SETC  'B','SUSQUEHANNA'                                       04700000
                               (Q)  96 &XC(1)   SETC  'C','TRANSYLVANIA'                                      04750000
                                    97          DEMO  KEY3=2,WRITE,REALLYLONGSYMBOL,                         M04800000
                                                      A8+8*(B5-CONSTANT-7)(3),KEY1=C,(6),SF,                 N04850000
                                                      (8),KEY2=7                                              04900000
000066 1816                         98+         LR    1,6                          LOAD DECB ADDRESS      03-IHBRD
000068 9220 1005      00005         99+         MVI   5(1),X'20'               SET TYPE FIELD             03-IHBRD
00006C 5081 0008            00008  100+         ST    8,8(1,0)                     STORE DCB ADDRESS      03-IHBRD
000070 58F1 0008            00008  101+         L     15,8(1,0)           LOAD DCB ADDRESS                03-IHBRD
000074 58F0 F030            00030  102+         L     15,48(0,15)                  LOAD RDWR ROUTINE ADDR 03-IHBRD
000078 05EF                        103+         BALR  14,15                        LINK TO RDWR ROUTINE   03-IHBRD
                                   104+*,&KEY2 NOT 1,2, OR 3---USE &KEY3 IN PLACE OF IT                   01-00076
                                   105+*,&KEY3 = 2                                                        01-00082
00007A 5850 A008            00008  106+         L     5,8(,10)       NOTE THAT OPCODES, OPERANDS & COMMENTS   01-00085
00007E 1B9A                     (R)107+SUSQUEHANNA SR 9,10             ON MODEL STATEMENTS                    01-00086
000080 98CD 8090            00090  108+TRANSYLVANIA LM 12,13,=A(A5,X)    ARE KEPT IN PLACE UNLESS DISPLACED   01-00087
000084 5073 80A8            000A8  109+REALLYLONGSYMBOL ST 7,A8+8*(B5-CONSTANT-7)(3)                         X01-00088
                                      +                                AS A RESULT OF SUBSTITUTION
```

(J) The programmer macro DEMO is defined after the start of the assembly. Macros can be defined at any point and, having been defined and/or expanded, can be redefined. Note that the parameters on the prototype are a mixture of keywords and positional operands. &SYSLIST may be used. The positional parameters are identified and numbered 1, 2, 3 from left to right; keywords are skipped over.

(K) Statement 70 illustrates 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 72 is a model statement with a symbolic parameter in its operation field. This statement will be edited as if it is a macro call; at this time, each operand will be denoted as positional or keyword. At macro call time, it will not be possible to reverse this decision. Even though treated as a macro, it is still expanded as a machine or assembler operation.

(M) Statement 74 illustrates the computed AGO statement. Control will pass to .MNOTE1 if &KEY2 is 1, to .MNOTE2 if &KEY2 is 2, to .MNOTE3 if &KEY2 is 3 or will fall through to the model statement at 75 otherwise.

(N) Statement 77 illustrates 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, else falls through to the next model statement.

(O) Statement 87 contains a subscripted created SET symbol in the name field. Exclusive of the subscript notation, these SET symbols have the form &(e), where e is an expression made up of character strings and/or variable symbols. When such a symbol is encountered at expansion time, the assembler evaluates e and attempts to use &(value) in place of &(e). Looking ahead, we see that DEMO is used as a macro instruction in statement 97 and &KEY1=C. Thus, the 'e' in this case is X&KEY1, which has the value XC. Finally, the macro-generator will use &XC(2) as the name field of this model statement. In statement 108, note that &XC(2) equals TRANSYLVANIA (statement 96). Finally, in the sequence field of statement 108, we see that this statement is a level 01 expansion of a programmer macro and the corresponding model statement is statement number 87.

Created SET symbols may be used wherever regular SET symbols are used in declarations, name fields, or operands of SET statements, in model statements, etc. Likewise, they are subject to all the restrictions of regular SET symbols. In the programmer macro DEMO, it would not have been valid to have the statement GBLC &(X&KEY1)(1) because, in statement 71, &XA, &XB, and &XC are declared as global variable symbols and &(X&KEY1)(2) becomes &XC(2) unless, of course, &KEY1 were assigned something other than the value A, B, or C in the macro instruction DEMO, statement 97. In that case, we would need a global declaration statement if we wanted &(X&KEY1) to be a global SET symbol. Because global declarations are processed at generation time and then only if the statement is encountered, we would insert the following statements between, say, statements 71 and 72:

```
      AIF ('&KEY1' EQ 'A' OR '&KEY1' EQ 'B' OR '&KEY1' EQ 'C').SKIP
      GBLC &(X&KEY1)(1)
 .SKIP ANOP
```

As the macro is defined, &(X&KEY1) will be a global SETC if &KEY1 is A, B, or C; otherwise it will be a LCLC or, possibly, a LCLA. In the macro, if &(X&KEY1) becomes a local, it will have a null or zero value.

(P) In statements 93 and 94, note that &XA is declared as a subscripted global SETC variable with a maximum subscript of 1 and, in the next statement (an extended SET statement), we store something into &XA(2). There is no contradiction here. The statement GBLC &XA(1) marks &XA as a subscripted global SETC symbol. Any decimal self-defined number (1 through 2147483647) can be used. Furthermore, only a nominal amount of space is set aside in the global dictionary. This space is open-ended and will be increased on demand and only on demand.

(Q) Statement 97 is the macro instruction DEMO. Note that &P1 has the value WRITE. Therefore, the model statement at statement 72 becomes an inner macro, WRITE, producing the code at statements 98-103. 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 108 and 109 contain some ordinary symbols longer than 8 characters. The limit for ordinary symbols, operation codes (for programmer and library macros and operation codes defined through OPSYN), variable symbols, and sequence symbols is 63 characters (including the & and . in the latter two instances, respectively). Most long symbols will probably be nearer to 8 than 63 characters in length. Extremely long symbols are simply too difficult to write, especially if the symbol is used frequently. The requirement that the operation field be present in the first statement of a continued statement is still in effect. Furthermore, names of START, CSECT, EXTRN, WXTRN, ENTRY, etc., symbols are still restricted to 8 characters.

```
BIGNAME  INSERT PROGRAMMER MACRO IN SOURCE STREAM NOW                                                  PAGE    5

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                   ASM H V 02 13.19 02/19/82
                                    111 ********************************************************************* 05000000
                                    112 *  COPY 'NOTE' MACRO IN FROM MACLIB, RENAME IT 'MARK', CALL IT UNDER * 05050000
                                    113 *     ITS ALIAS -- IN EXPANSION OF MARK, NOTICE REFERENCE BACK TO    * 05100000
                                    114 *       DEFINITION STATEMENTS IN 'COLUMNS' 76-80 OF EXPANSION        * 05150000
                                    115 ********************************************************************* 05200000

                              S     117          COPY  NOTE                                                   05300000
                                    118          MACRO                                                        00020000
                                    119 &NAME    NOTE  &DCB,&DUMMY=                                           00040017
                                    120          AIF   ('&DCB' EQ '').ERR                                     00060000
                                    121 &NAME    IHBINNRA &DCB                                                00080000
                                    122          L     15,84(0,1)                       LOAD NOTE  RTN ADDRESS 00100000
                                    123          BALR  14,15                            LINK TO NOTE  ROUTINE  00120000
                                    124          MEXIT                                                        00140000
                                    125 .ERR     IHBERMAC 6                                                   00160000
                                    126          MEND                                                         00180000

                              T     129 MARK     OPSYN NOTE     COMMENTS OF GENERATED STATEMENTS OCCUPY SAME   05450000
                                    130          MARK  (6)         'COLUMNS' AS THOSE IN MODEL STATEMENTS     05500000
000088 1816                         131+         LR    1,6                              LOAD PARAMETER REG 1   02-IHBIN
00008A 58F0 1054             00054  132+         L     15,84(0,1)                       LOAD NOTE  RTN ADDRESS 01-00122
00008E 05EF                         133+         BALR  14,15                            LINK TO NOTE  ROUTINE  01-00123

                                    135 ********************************************************************* 05600000
00009C                              136 DEECEES  LOCTR        SWITCH TO ALTERNATE LOCATION COUNTER            05650000
00009C 00000000
0000A0 0B0000A000000050             137 B5       CCW   X'0B',B5,0,80                                          05700000

                                    139 ********************************************************************* 05800000
                                    140 *  DISPLAY OF &SYSTIME, &SYSDATE, &SYSPARM AND &SYSLOC               * 05850000
                                    141 ********************************************************************* 05900000

                              U     143          PRINT NODATA                                                 06000000
                                    144   DC  C'TIME = &SYSTIME, DATE = &SYSDATE, PARM = &SYSPARM'            06050000
0000A8 E3C9D4C5407E40F1               + DC  C'TIME = 13.19, DATE = 02/19/82, PARM = SAMPLE PROGRAM'       06050000

                                    146          MACRO                                                        06150000
                                    147          LOCATE                                                       06200000
                                    148 &SYSECT  CSECT      DISPLAY OF CURRENT CONTROL SECTION                06250000
                                    149 &SYSLOC  LOCTR          AND LOCATION COUNTER                          06300000
                                    150          MEND                                                         06350000
                              V
                                    152          LOCATE                                                       06450000
0000DC                              153+A        CSECT      DISPLAY OF CURRENT CONTROL SECTION                01-00148
0000DC                              154+DEECEES  LOCTR          AND LOCATION COUNTER                          01-00149
000090                              155 A        LOCTR                                                        06500000

                              W     157 ********************************************************************* 06600000
000000                              158 PD2      COM                     NAMED COMMON THROWN IN FOR GOOD MEASURE 06650000
000000                              159          DS    500F                                                   06700000
0007D0 1867                         160          LR    6,7                                                    06750000
                              X     161          END                                                          06800000
000090 00000040FFFFFFE8             162                =A(A5,X)
```

(S) Library macros may be inserted into the source stream as programmer macros by use of a COPY statement. The result (statements 118 to 126) is essentially a programmer macro definition. When a library macro is brought in and expanded by use of a macro instruction, the assembler (1) looks the macro up by its member-name and (2) verifies that this same name is used in the operation field of the prototype statement. Therefore, for example, DCB has to be cataloged as DCB. However, as COPY code, the member name bears no relationship to any of the statements in the member. Thus, several variations of a given macro could be stored as a library under separate names, then copied in at various places in a single assembly as needed. (Assembler H allows you to define and redefine a macro any number of times).

(T) In statement 129, MARK is made a synonym for NOTE. To identify NOTE as a macro, it has to be used as either a system macro call (that is, from a macro library) or a programmer macro definition prior to its use in the operand field of an OPSYN statement. The COPY code at 118 through 126 is a programmer macro definition. The macro instruction at statement 130 is MARK. We can use MARK and NOTE interchangeably. If desired, we could remove NOTE as a macro definition in the following way:

```
MARK                  OPSYN           NOTE
NOTE                  OPSYN
```

We could then refer to the macro only as MARK.

(U) Statement 144 demonstrates &SYSTIME, &SYSDATE and &SYSPARM. The values for the first two are the same as we use in the heading line. The value for &SYSPARM is the value passed in the PARM field of the EXEC statement of the default value assigned to &SYSPARM when Assembler H is installed.

(V) System variable symbols &SYSLOC and &SYSECT are displayed. The sequence field indicates that the model statements are statements 148 and 149.

(W) Illustration of named COMMON. You can establish addressability for a named COMMON section with:

```
USING           section-name, register
```

You can address data in a blank COMMON section by labeling a statement after the COMMON statement and using relative addressing.

(X) If there are literals outstanding when the END statement is encountered, they are assigned to the LOCTR currently in effect for the first control section in the assembly. This may or may not put the literals at the end of the first control section. In this sample assembly, the first control section, A, has two LOCTRs, A and DEECEES. Because A is active (at statement 155), the literals are assembled there. You always have the ability to control placement of literal pools by means of the LTORG statement. Note that X'FFFFFFE8' is used for the contents of A(A5,X), statement 162. The symbol X was assigned the value (4*-6) by an EQU in statement 60.

```
BIGNAME                                    RELOCATION DICTIONARY                                   PAGE    6

 POS.ID    REL.ID    FLAGS    ADDRESS                                                 ASM H V 02 13.19 02/19/82

  0001      0001      0C      000090
  0001      0001      08      0000A1
```

```
BIGNAME                                 CROSS REFERENCE                                            PAGE    7

SYMBOL     LEN     VALUE   DEFN    REFERENCES                                         ASM H V 02 13.19 02/19/82

A         00001 00000000 0002    0028 0153 0155
A5        00002   000040 0034    0038 0162
A7        00016   000048 0036
A8        00002 000000A0 0038    0109
B5        00008   0000A0 0137    0038 0109 0137
CONSTANT  00004   000098 0027    0025 0109
DEECEES   00001 00000098 0026    0136 0154
PD2       00001 00000000 0158
REALLYLONGSYMBOL
          00004   000084 0109    0014 0018
SUSQUEHANNA
          00002   00007E 0107
TRANSYLVANIA
          00004   000080 0108    0014 0018
X         00001 FFFFFFE8 0060    0162
=A(A5,X)  00004   000090 0162    0108
```

```
BIGNAME                       DIAGNOSTIC CROSS REFERENCE AND ASSEMBLER SUMMARY                     PAGE    8

                                                                                  ASM H V 02 13.19 02/19/82

     NO STATEMENTS FLAGGED IN THIS ASSEMBLY

   OVERRIDING PARAMETERS-  SYSPARM(SAMPLE PROGRAM),NODECK,BATCH
   OPTIONS FOR THIS ASSEMBLY
    NODECK, OBJECT, LIST, XREF(FULL), NORENT, NOTEST, BATCH, ALIGN, ESD, RLD, NOTERM, LINECOUNT(55),
    FLAG(0), SYSPARM(SAMPLE PROGRAM)
   NO OVERRIDING DD NAMES

     136 CARDS FROM SYSIN      524 CARDS FROM SYSLIB
     198 LINES OUTPUT           11 CARDS OUTPUT
```

APPENDIX B. SAMPLE MACRO TRACE AND DUMP (MHELP)

The macro trace and dump (MHELP) facility is a useful means of debugging macro definitions. MHELP can be used anywhere in the source program or in macro definitions. MHELP is processed during macro generation. It is completely dynamic; you can branch around the MHELP statements by using AIF or AGO statements. Therefore, its use can be controlled by symbolic parameters and SET symbols.

The following sample program illustrates the five primary functions of MHELP. Because most of the information produced is unrelated to statement numbers, the dumps and traces in the listing are marked with numbers in parentheses. Most dumps refer to statement numbers. If you request MHELP information about a library macro definition, the first five characters of the macro name will appear in place of the statement number. To get the statement numbers, you should use COPY to copy the library definition into the source program prior to the macro call.

MACRO CALL TRACE (MHELP1)

Item (1A) illustrates an outer macro call, (1B) an inner one. In each case, the amount of information given is brief. This trace is given after successful entry into the macro; no dump is given if error conditions prevent an entry.

MACRO ENTRY DUMP (MHELP 16)

This provides values of system variable symbols and symbolic parameters at the time the macro is called. The following numbering system is used:

Number	**Item**
000	&SYSNDX
001	&SYSECT
002	&SYSLOC
003	&SYSTIME
004	&SYSDATE
005	&SYSPARM
006	Name Field on Macro Instruction

If there are NKW keyword parameters, they follow in order of appearance on the prototype statement.

007	1st keyword value
008	2nd keyword value
.	.
.	.
.	.
006+NKW	NKWth keyword value

If there are NPP positional parameters, they follow in order of appearance in the macro instruction.

007+NKW	1st positional parameter values
008+NKW	2nd positional parameter values
.	
.	
.	
006+NKW+NPP	NPPth positional parameter values

For example, item (16A) has one keyword parameter (&OFFSET) and one positional parameter. The value of the keyword parameter appears opposite 110006, the positional parameter, opposite 110007. In both the prototype (statement 3) and the macro instruction (statement 54), the positional parameter appears in the first operand field, the keyword in the second. A length appears between the NUM and VALUE fields. A length of NUL indicates the corresponding item is empty.

Item (16B) illustrates an inner call containing zero keywords and two positional parameters.

MACRO AIF DUMP (MHELP 4)

Items (4A), (4B), (4C),... are examples of these dumps. Each such 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.

MACRO BRANCH TRACE (MHELP 2)

This provides a one-line trace for each AGO and true AIF branch within a programmer macro. In any such branch, the "branched from" statement number, the "branched to" statement number, and the macro name are included. Note, in example (2A), the "branched to" statement number indicated is not that of the ANOP statement bearing the target sequence symbol but that of the statement following it. The branch trace facility is suspended when library macros are expanded and MHELP 2 is in effect. To obtain a macro branch trace for such a macro, one would have to insert a COPY "macro-name" statement in the source deck at some point prior to the MHELP 2 statement of interest.

MACRO EXIT DUMP (MHELP 8)

This 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.

Note: Local and/or global variable symbols are not displayed at any point unless they appear in the current macro explicitly as SET symbols.

```
                                                                                          PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                     ASM H V 02 13.19 02/19/82

000000                                1          CSECT                                           06850000
                                      2 *        COPY   LNSRCH                                   06900000
                                      3          MACRO                                           06950000
                                      4 &NAME   LNSRCH &ARG,&OFFSET=STNUMB-STCHAIN               07000000
                                      5          LCLC   &LABEL                                   07050000
                                      6 &LABEL   SETC   'A&SYSNDX'         GENERATE SYMBOL       07100000
                                      7          AIF    (T'&NAME EQ 'O').SKIP                    07150000
                                      8 &LABEL   SETC   '&NAME'            IF MACRO CALL HAS LABEL, USE IT  07200000
                                      9 .SKIP    ANOP                      INSTEAD OF GENERATED SYMBOL      07250000
                                     10 &LABEL   LA     0,&OFFSET          LOAD REG. 0           07300000
                                     11          SCHI   &ARG,0(1)          SEARCH                07350000
                                     12          BC     1,&LABEL           IF MAX REACHED, CONTINUE         07400000
                                     13          MEND                                            07450000
```

```
                                                                                          PAGE    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                     ASM H V 02 13.19 02/19/82

                                     15 *        COPY   SCHI                                     07550000
                                     16          MACRO                                           07600000
                                     17 &NM      SCHI   &COMP,&LIST                              07650000
                                     18          LCLA   &CNT                                     07700000
                                     19          LCLC   &CMPADR                                  07750000
                                     20 &CNT     SETA   1                                        07800000
                                     21 &NM      STM    1,15,4(13)                               07850000
                                     22 .TEST    ANOP                                            07900000
                                     23 &CMPADR  SETC   '&CMPADR'.'&COMP'(&CNT,1)                07950000
                                     24          AIF    ('&COMP'(&CNT,1) EQ '(').LPAR            08000000
                                     25 &CNT     SETA   &CNT+1                                   08050000
                                     26          AIF    (&CNT LT K'&COMP).TEST                   08100000
                                     27 .NOLNTH  ANOP                                            08150000
                                     28          LA     3,&COMP            COMPARAND             08200000
                                     29          AGO    .CONTIN                                  08250000
                                     30 .LPAR    AIF    ('&COMP'(&CNT+1,1) EQ ',').FINISH        08300000
                                     31 &CNT     SETA   &CNT+1                                   08350000
                                     32          AIF    (&CNT LT K'&COMP).LPAR                   08400000
                                     33          AGO    .NOLNTH                                  08450000
                                     34 .FINISH  ANOP                                            08500000
                                     35 &CMPADR  SETC   '&CMPADR'.'&COMP'(&CNT+2,K'&COMP-&CNT)   08550000
                                     36          LA     3,&CMPADR          COMPARAND SANS LENGTH 08600000
                                     37 .CONTIN  ANOP                                            08650000
                                     38          LA     1,&LIST            LIST HEADER           08700000
                                     39          MVC    &COMP,0(0)         DUMMY MOVE TO GET COMP LENGTH    08750000
                                     40          ORG    *-6                CHANGE MVC TO MVI     08800000
                                     41          DC     X'92'              MVI OPCODE            08850000
                                     42          ORG    *+1                PRESERVE LENGTH AS IMMED OPND    08900000
                                     43          DC     X'D000'            RESULT IS MVI 0(13),L 08950000
                                     44          L      15,=V(SCHI)                              09000000
                                     45          BALR   14,15                                    09050000
                                     46          LM   1,15,4(13)                                 09100000
                                     47          MEXIT                                           09150000
                                     48          MEND                                            09200000
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          ASM H V 02 13.19 02/19/82

000000                               50 TEST     CSECT                                                               09300000
000000 05C0                          51          BALR  12,0                                                          09350000
                            00002    52          USING *,12                                                          09400000


                                     54          MHELP B'11111'                                                      09500000
                                     55          LNSRCH LISTLINE,OFFSET=LISTLINE-LISTNEXT                            09550000
                             1A         ++//MHELP. CALL TO MACRO LNSRCH   . DEPTH=001, SYSNDX=0001, STMT 00055

                             16A        //MHELP ENTRY TO  LNSRCH   . MODEL STMT 00000, DEPTH=001, SYSNDX=0001, KWCNT=001
                                        ////PARAMETERS (SYSNDX,SYSECT,SYSLOC,SYSTIME,SYSDATE,SYSPARM,NAME,KWS,PPS) ///
                                        //NUM  LNTH  VALUE (64 CHARS/LINE)
                                        //0000  004  0001
                                        //0001  004  TEST
                                        //0002  004  TEST
                                        //0003  005  13.19
                                        //0004  008  02/19/82
                                        //0005  014  SAMPLE PROGRAM
                                        //0006  NUL
                                        //0007  017  LISTLINE-LISTNEXT
                                        //0008  008  LISTLINE

                             4A         //MHELP AIF IN    LNSRCH   . MODEL STMT 00007, DEPTH=001, SYSNDX=0001, KWCNT=001
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLC       LABEL                                        LNTH= 005
                                        //     VAL=A0001

                             2A         ++//MHELP. BRANCH FROM STMT 00007 TO STMT 00010 IN MACRO LNSRCH
000002 4100 0002            00002    56+A0001    LA    0,LISTLINE-LISTNEXT LOAD REG. 0                               01-00010
                             1B         ++//MHELP. CALL TO MACRO SCHI     . DEPTH=002, SYSNDX=0002, STMT 00011

                             16B        //MHELP ENTRY TO  SCHI     . MODEL STMT 00000, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////PARAMETERS (SYSNDX,SYSECT,SYSLOC,SYSTIME,SYSDATE,SYSPARM,NAME,KWS,PPS) ///
                                        //NUM  LNTH  VALUE (64 CHARS/LINE)
                                        //0000  004  0002
                                        //0001  004  TEST
                                        //0002  004  TEST
                                        //0003  005  13.19
                                        //0004  008  02/19/82
                                        //0005  014  SAMPLE PROGRAM
                                        //0006  NUL
                                        //0007  008  LISTLINE
                                        //0008  004  0(1)
000006 901F D004            00004    57+         STM   1,15,4(13)                                                    02-00021
                             4B         //MHELP AIF IN    SCHI     . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
```

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  ASM H V 02 13.19 02/19/82
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000001
                                        //0001 LCLC       CMPADR                                       LNTH= 001
                                        //     VAL=L

                               4C       //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000002
                                        //0001 LCLC       CMPADR                                       LNTH= 001
                                        //     VAL=L

                               2B       ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                               4D       //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000002
                                        //0001 LCLC       CMPADR                                       LNTH= 002
                                        //     VAL=LI

                               4E       //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000003
                                        //0001 LCLC       CMPADR                                       LNTH= 002
                                        //     VAL=LI

                               2C       ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                                        //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000003
                                        //0001 LCLC       CMPADR                                       LNTH= 003
                                        //     VAL=LIS

                                        //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000004
                                        //0001 LCLC       CMPADR                                       LNTH= 003
                                        //     VAL=LIS

                                        ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                                        //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000004
                                        //0001 LCLC       CMPADR                                       LNTH= 004
                                        //     VAL=LIST
```

```
LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  ASM H V 02 13.19 02/19/82

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000005
                                         //0001 LCLC       CMPADR                                   LNTH= 004
                                         //      VAL=LIST

                                         ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000005
                                         //0001 LCLC       CMPADR                                   LNTH= 005
                                         //      VAL=LISTL

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000006
                                         //0001 LCLC       CMPADR                                   LNTH= 005
                                         //      VAL=LISTL

                                         ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000006
                                         //0001 LCLC       CMPADR                                   LNTH= 006
                                         //      VAL=LISTLI

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000007
                                         //0001 LCLC       CMPADR                                   LNTH= 006
                                         //      VAL=LISTLI

                                         ++//MHELP. BRANCH FROM STMT 00026 TO STMT 00023 IN MACRO SCHI

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00024, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                         //0000 LCLA       CNT                                      VAL=  0000000007
                                         //0001 LCLC       CMPADR                                   LNTH= 007
                                         //      VAL=LISTLIN

                                         //MHELP AIF IN    SCHI    . MODEL STMT 00026, DEPTH=002, SYSNDX=0002, KWCNT=000
                                         ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
```

```
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          ASM H V 02 13.19 02/19/82
                                        //0000 LCLA       CNT                                          VAL=  0000000008
                                        //0001 LCLC       CMPADR                                       LNTH= 007
                                        //      VAL=LISTLIN

00000A 4130 C024            00026     58+          LA    3,LISTLINE          COMPARAND                         02-00028

                                        ++//MHELP. BRANCH FROM STMT 00029 TO STMT 00038 IN MACRO SCHI

00000E 4111 0000            00000     59+          LA    1,0(1)              LIST HEADER                       02-00038
000012 D202 C024 0000 00026 00000     60+          MVC   LISTLINE,0(0)       DUMMY MOVE TO GET COMP LENGTH     02-00039
000018                      00012     61+          ORG   *-6                 CHANGE MVC TO MVI                 02-00040
000012 92                             62+          DC    X'92'               MVI OPCODE                        02-00041
000013                      00014     63+          ORG   *+1                 PRESERVE LENGTH AS IMMED OPND     02-00042
000014 D000                           64+          DC    X'D000'             RESULT IS MVI 0(13),L             02-00043
000016 58F0 C02E            00030     65+          L     15,=V(SCHI)                                           02-00044
00001A 05EF                           66+          BALR  14,15                                                 02-00045
00001C 981F D004            00004     67+          LM    1,15,4(13)                                            02-00046
                           8A           //MHELP EXIT FROM SCHI     . MODEL STMT 00047, DEPTH=002, SYSNDX=0002, KWCNT=000
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLA       CNT                                          VAL=  0000000008
                                        //0001 LCLC       CMPADR                                       LNTH= 007
                                        //      VAL=LISTLIN

000020 4710 C000            00002     68+          BC    1,A0001             IF MAX REACHED, CONTINUE          01-00012
                           8B           //MHELP EXIT FROM LNSRCH   . MODEL STMT 00013, DEPTH=001, SYSNDX=0001, KWCNT=001
                                        ////SET SYMBOLS (SKIPPED NUMBERS MAY BE SEQUENCE SYMBOLS).//
                                        //0000 LCLC       LABEL                                        LNTH= 005
                                        //      VAL=A0001

000024                                69 LISTNEXT DS    H                                                     09600000
000026                                70 LISTLINE DS    FL3'0'                                                09650000
000030                                71          LTORG                                                      09700000
000030 00000000                       72                =V(SCHI)
000000                                73          END   TEST                                                  09750000
```

APPENDIX C. OBJECT DECK OUTPUT

ESD CARD FORMAT

The format of the ESD card follows:

Columns	**Contents**
1	X'02'
2-4	ESD
5-10	Blank
11-12	Variable field count—number of bytes of information in variable field (columns 17-64)
13-14	Blank
15-16	ESDID of first SD, XD, CM, PC, ER, or WX in variable field
17-64	Variable field. One to three 16-byte items of the following format: 8 bytes—Name 1 byte —ESD type code; the hexadecimal value is: 00 SD 01 LD 02 ER 04 PC 05 CM 06 XD(PR) 0A WX 3 bytes—Address 1 byte —Alignment if XD —Blank if LD, ER, or WX —AMODE/RMODE flags if SD, PC, or CM Bit 5: 0 = RMODE is 24 1 = RMODE is ANY Bits 6-7: 00 = AMODE is 24 01 = AMODE is 24 10 = AMODE is 31 11 = AMODE is ANY 3 bytes—Length, LDID, or blank
65-72	Blank
73-80	Deck ID and/or sequence number. The deck ID is the name from the first TITLE statement that has a nonblank name field. This name can be 1 to 8 characters long. If the name is fewer than 8 characters long or if there is no name, the remaining columns contain a card sequence number. (Columns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a sequence number.)

TEXT (TXT) CARD FORMAT

The format of the TXT cards follows:

Columns	Contents
1	X'02'
2-4	TXT
5	Blank
6-8	Relative address of first instruction on card
9-10	Blank
11-12	Byte count—number of bytes in information field (columns 17-72)
13-14	Blank
15-16	ESDID
17-72	56-byte information field
73-80	Deck ID and/or sequence number. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1 to 8 characters long. If the name is fewer than 8 characters long or if there is no name, the remaining columns contain a card sequence number. (Columns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a sequence number.)

RLD CARD FORMAT

The format of the RLD card follows:

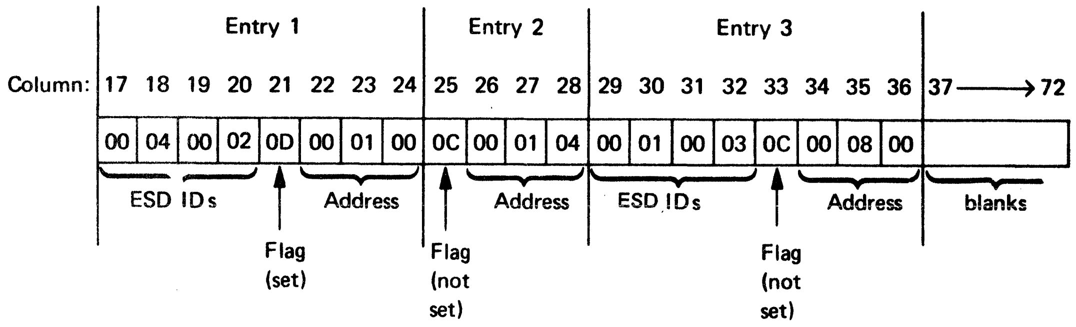
Columns	Contents
1	X'02'
2-4	RLD
5-10	Blank
11-12	Data field count—number of bytes of information in data field (columns 17-72)
13-16	Blank
17-72	Data field:
17-18	Relocation ESDID
19-20	Position ESDID
21	Flag byte
22-24	Absolute address to be relocated
25-72	Remaining RLD entries
73-80	Deck ID and/or sequence number. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1 to 8 characters long or if there is no name, the remaining columns contain a card sequence number. (Columns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a 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 will not be repeated; if the rightmost bit of the flag byte is not set, the next RLD entry has a different relocation ESDID and/or position ESDID, and both ESDIDs will be recorded.

For example, if the RLD entries 1, 2, and 3 of the program listing contain the following information:

Entry	Position ESDID	Relocation ESDID	Flag	Address
1	02	04	0C	000100
2	02	04	0C	000104
3	03	01	0C	000800

then columns 17-72 of the RLD card would be as follows:


Entry 1
Entry 2
Entry 3
Column: 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 → 72
00 04 00 02 0D 00 01 00 0C 00 01 04 00 01 00 03 0C 00 08 00
ESD IDs
Address
Address
ESD IDs
Address
blanks
Flag (set)
Flag (not set)
Flag (not set)

END CARD FORMAT

The format of the END card follows:

Columns	Contents
1	X'02'
2-4	END
5	Blank
6-8	Entry address from operand of END card in source deck (blank if no operand)
9-14	Blank
15-16	ESDID of entry point (blank if no operand)
17-32	Blank
33	Number of IDR items that follow (EBCDIC1 or EBCDIC2)
34-52	Translator identification, version and release level (such as 0201), and date of the assembly (yyddd)
53-71	When present, they are the same format as columns 34-52
72-80	Deck ID and/or sequence number. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1 to 8 characters long. If the name is fewer than 8 characters long or if there is no name, the remaining columns contain a card sequence number. (Columns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a sequence number.)

TESTRAN (SYM) CARD FORMAT

If you request it, the assembler punches out symbolic information for TESTRAN concerning the assembled program. This output appears ahead of all loader text. The format of the card images for TESTRAN output follows:

Columns	Contents
1	X'02'
2-4	SYM
5-10	Blank
11-12	Variable field—number of bytes of text in variable field (columns 17-72)
13-16	Blank
17-72	Variable field (see below)
73-80	Deck ID and/or sequence number. The deck ID is the name from the first TITLE statement that has a nonblank name field. The name can be 1 to 8 characters long. If the name is fewer than 8 characters long or if there is no name, the remaining columns contain a card sequence number. (Columns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a sequence number.)

The variable field (columns 17-72) contains up to 56 bytes of TESTRAN text. The items comprising the text are packed together; consequently, only the last card may contain less than 56 bytes of text in the variable field. The formats of a text card and an individual text item are shown in Figure 19 on page 99 . The contents of the fields within an individual entry are as follows:

1. Organization (1 byte)

   Bit 0: 0 = nondata type
   1 = data type

   Bits 1-3 (if nondata type):
   000 = space
   001 = control section
   010 = dummy control section
   011 = common
   100 = instruction
   101 = CCW, CCW0, CCW1

   Bit 1 (if data type):
   0 = no multiplicity
   1 = multiplicity (indicates presence of M field)

   Bit 2 (if data type):
   0 = independent (not a packed or zoned decimal constant)
   1 = cluster (packed or zoned decimal constant)

   Bit 3 (if data type):
   0 = no scaling
   1 = scaling (indicates presence of S field)

   Bit 4:
   0 = name present
   1 = name not present

   Bits 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

   Note: If the entry is nondata type and space, an extra byte is present that contains the number of bytes that have been skipped.

4. Data Type (1 byte)—contents in hexadecimal

00	=	character
04	=	hexadecimal
08	=	binary
10	=	fixed point, full
14	=	fixed point, half
18	=	floating point, short
1C	=	floating point, long
20	=	A-type or Q-type data
24	=	Y-type data
28	=	S-type data
2C	=	V-type data
30	=	packed decimal
34	=	zoned decimal
38	=	floating point, extended

5. Length (2 bytes for character, hexadecimal, decimal, or binary items; 1 byte for other types)—length of data item minus 1

6. Multiplicity—M field (3 bytes)—equals 1 if not present

7. Scale—signed integer—S field (2 bytes)—present only for F-, H-, E-, D-, P-, and Z-type data, and only if scale is nonzero.

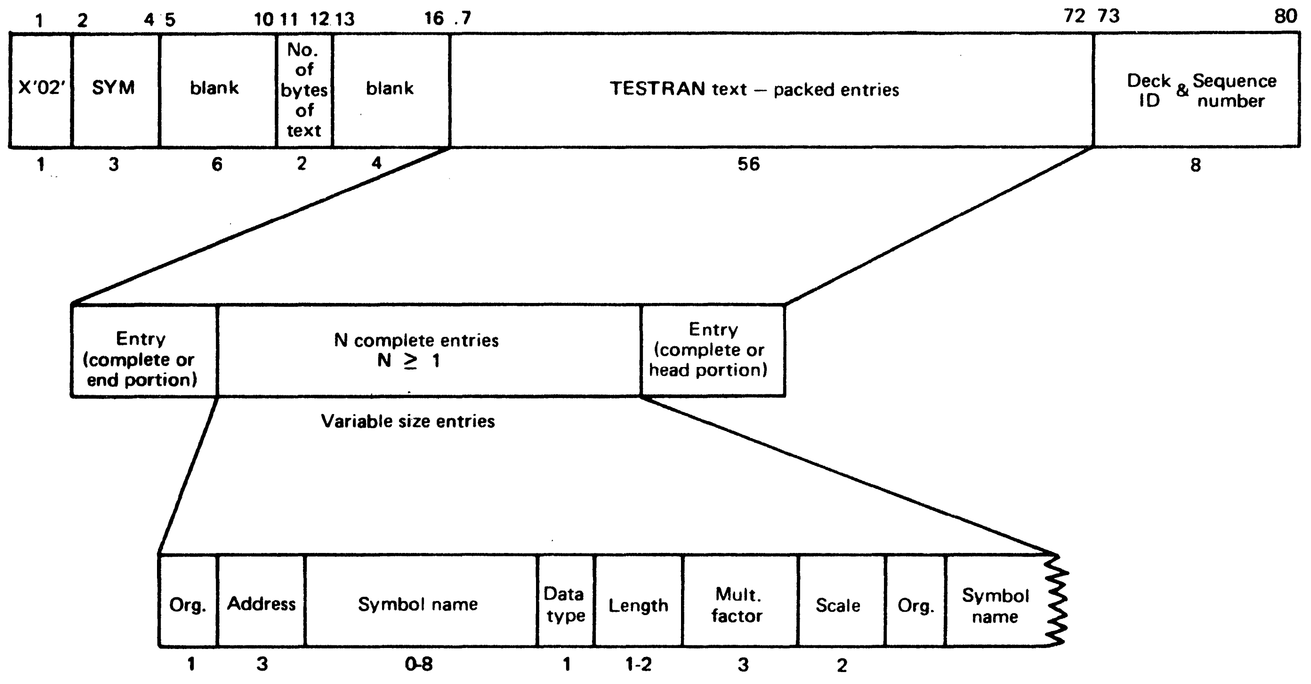


Figure 19. TESTRAN SYM Card Format

APPENDIX D. ASSEMBLER H MESSAGES

Assembler H has two types of messages: Assembly error diagnostic messages and assembly abnormal termination messages. The following section describes both types and gives their format and placement. "Assembly Error Diagnostic Messages" and"Abnormal Assembly Termination Messages" on page 126 describe and list each type of message.

MESSAGE DESCRIPTIONS

Each message entry in this book has five sections:

- Message Number and Text
- Explanation of Message
- System Action
- Programmer Response
- Severity Code

Message Number and Text

Only the message number and the major fixed portion of the message text included in the message description. Any abbreviations in actual message text are spelled out in full in the book. Unused message numbers account for the gaps in the message number sequence. No messages are defined for numbers, such as IEV006, not included in this section.

Explanation of Message

There may be more than one explanation for some messages, because they are generated by different sections of the assembler. Several of the assembler termination messages have identical explanations.

System Action

This section tells how the assembler handles statements with errors. A machine instruction is assembled as all zeros. An assembler instruction is usually ignored; it is printed but has no effect on the assembly. Many assembler instructions, however, are partially processed or processed with a default value.

For some instructions, the operands preceding the operand in error or every operand except the operand in error is processed. For example, if one of several operands on a DROP statement is a symbol that has not been equated to a register number, only that operand is ignored. All the correctly specified registers are correctly processed.

For some assembler statements, especially macro prototype and conditional assembly statements, the operand or term in error is given a default value. Thus the statement will assemble completely, but will probably cause incorrect results if the program is executed.

Programmer Response

Many errors have specific or probable causes. In such a case, the Programmer Response section gives specific steps for fixing the error. Most messages, however, have too many possible causes (from keypunch error to wrong use of the statement) to list. The programmer response 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 severity code indicates the seriousness of the error. The severity codes and their meanings are shown in the table at the end of this appendix.

This code is the return code issued by the assembler when it returns control to the operating system. The IBM-supplied cataloged procedures include a COND parameter on the linkage edit and execution steps. The COND parameter prevents execution of these steps if the return code from the assembler is 8 or greater. Thus errors with ***ERROR*** in the message prevent the assembled program from linkage editing or executing. Errors with **WARNING** in the message do not.

ASSEMBLY ERROR DIAGNOSTIC MESSAGES

Assembler H prints most error messages in the listing immediately following the statements in error. It also prints the total number of flagged statements and their line numbers in the Diagnostic Cross Reference section at the end of the listing.

The messages do not follow the statement in error when:

- Errors are detected during editing of macro definitions read from a library. A message for such an error appears after the first call in the source program to that macro definition. You can, however, bring the macro definition into the source program with a COPY statement. The editing error messages will then be attached to the statements in error.

- Errors are detected by the lookahead function of the assembler. (Lookahead scans, for attribute references, statements after the one being assembled.) Messages for these errors appear after the statements in which they occur. The messages may also appear at the point at which lookahead was called.

- Errors are detected on conditional assembly statements during macro generation or MHELP testing. Such a message follows the most recently generated statement or MHELP output statement.

A typical error diagnostic messsage is:

```
IEV057  ***ERROR***  UNDEFINED OPERATION
        CODE—xxxxx
```

The term ***ERROR*** is part of the message if the severity code is 8 or greater. The term **WARNING** is part of the message if the severity code is 0 or 4.

A copy of a segment of the statement in error, represented above by xxxxx, is appended to the end of many messages. Normally this segment, which can be up to 16 bytes long, begins at the bad character or term. For some errors, however, the segment may begin after the bad character or term. The segment may include part of the remarks field.

If a diagnostic message follows a statement generated by a macro definition, the following items may be appended to the error message:

- The number of the model statement in which the error occurred, or the first five characters of the macro name.

- The SET symbol, parameter number, or value string associated with the error.

Note: References to macro parameters are by number (such as PARAM008) instead of by name. The first seven numbers are always assigned for the standard system parameters as follows:

```
PARAM000 = &SYSNDX
PARAM001 = &SYSECT
PARAM002 = &SYSLOC
PARAM003 = &SYSTIME
PARAM004 = &SYSDATE
PARAM005 = &SYSPARM
PARAM006 = Name Field Parameter
```

Then the keyword parameters are numbered in the order defined in the macro definition, followed by positional parameters. When there are no keyword parameters in the macro definition, PARAM007 refers to the first positional parameter.

If a diagnostic message follows a conditional assembly statement in the source program, the following items will be appended to the error message:

- The word "OPENC"

- The SET symbol or value string associated with the error

Several messages may be issued for a single statement or even for a single error within a statement. This happens because each statement is usually evaluated on more than one level (for example, term level, expression level, and operand level) or by more than one phase of the assembler. Each level or phase can diagnose errors; therefore, most or all of the errors in the statement are flagged. Occasionally, duplicate error messages may occur. This is a normal result of the error detection process.

MESSAGE NOT KNOWN

The following message may appear in a listing:

```
IEVnnn ***ERROR*** MESSAGE NOT
       KNOWN—xxxxxxxxxx
```

The statement preceding this message contains an error but the assembler routine that detected the error issued the number (IEVnnn) of a nonexistent error message to the assembler's message generation routine. The segment of the statement in error may be appended to the message. If you can correct the error, this statement will assemble correctly. However, there is a bug in the error detection process of the assembler. Save the output and the source deck from this assembly and report the problem to your IBM customer engineer.

MESSAGES

IEV001 OPERATION-CODE NOT ALLOWED TO BE GENERATED

Explanation: An attempt was made to produce a restricted operation code by variable symbol substitution. Restricted operation codes are:

ACTR	AGO	AGOB	AREAD
AIF	AIFB	ANOP	SETA
COPY	REPRO	ICTL	SETB
MACRO	MEND	MEXIT	SETC
GBLA	GBLB	GBLC	
LCLA	LCLB	LCLC	

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

IEV002 GENERATED STATEMENT TOO LONG. STATEMENT TRUNCATED

Explanation: The statement generated by a macro definition is more than 864 characters long.

System Action: The statement is truncated; the leading 864 characters are retained.

Programmer Response: Shorten the statement.

Severity: 12

IEV003 UNDECLARED VARIABLE SYMBOL. DEFAULT=0, NULL, OR TYPE=U

Explanation: A variable symbol in the operand field of the statement has not been declared (defined) in the name field of a SET statement, in the operand field of an LCL or GBL statement, or in a macro prototype statement.

System Action: The variable symbol is given a default value as follows:

SETA = 0
SETB = 0
SETC = null (empty) string

The type attribute (T') of the variable is given a default value of U (undefined).

Programmer Response: Declare the variable _before_ you use it as an operand.

Severity: 8

IEV004 DUPLICATE SET SYMBOL DECLARATION. FIRST IS RETAINED

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

IEV005 NO CORE FOR INNER MACRO CALL. CONTINUE WITH OPEN CODE

Explanation: An inner macro call could not be executed because no main storage was available.

System Action: The assembly is continued 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

IEV007 PREVIOUSLY DEFINED SEQUENCE SYMBOL

Explanation: The sequence symbol in the name field has been used in the name field of a previous statement.

System Action: The first definition of the sequence symbol is used; this definition is ignored.

Programmer Response: Remove or change one of the sequence symbols.

Severity: 12

IEV008 PREVIOUSLY DEFINED SYMBOLIC PARAMETER

Explanation: The same variable symbol has been used to define two different symbolic parameters.

System Action: When the parameter name (the variable symbol) is used inside the macro definition, it will refer to the _first_ definition of the parameter in the prototype. However, if the second parameter defined by the variable symbol is a positional parameter, the count of positional operands will still be increased by one. The second parameter can then be referred to only through use of &SYSLIST.

Programmer Response: Change one of the parameter names to another variable symbol.

Severity: 12

IEV009 SYSTEM VARIABLE SYMBOL ILLEGALLY RE-DEFINED

Explanation: A system variable symbol has been used in the name field of a macro prototype statement. The system variable symbols are:

&SYSECT	&SYSDATE
&SYSLIST	&SYSLOC

&SYSNDX &SYSPARM
&SYSTIME

System Action: The name parameter is ignored. The name on a corresponding macro instruction will not be generated.

Programmer Response: Change the parameter to one that is not a system variable symbol.

Severity: 12

IEV011 INCONSISTENT GLOBAL DECLARATIONS. FIRST IS RETAINED

Explanation: A global SET variable symbol has been defined in more than one macro definition or in a macro definition and in the source program, and the two definitions are inconsistent in type or dimension.

System Action: The first definition encountered is retained.

Programmer Response: Assign a new SET symbol or make the definitions compatible.

Severity: 8

IEV012 UNDEFINED SEQUENCE SYMBOL. 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.

Severity: 12

IEV013 ACTR COUNTER EXCEEDED

Explanation: The conditional assembly loop counter (set by an ACTR statement) has been decremented to zero. The ACTR counter is decremented by one each time an AIF or AGO branch is executed successfully. The counter is halved for most errors encountered by the macro editor phase of the assembler.

System Action: A macro expansion is terminated. If the ACTR statement is in the source program, the assembly is terminated.

Programmer Response: Check for an AIF/AGO loop or another type of error. (You can use the MHELP facility, described in Chapter 3 and Appendix B, to trace macro definition logic.) If there is no error, increase the initial count on the ACTR instruction.

Severity: 12

IEV017 UNDEFINED KEYWORD PARAMETER. DEFAULT TO POSITIONAL INCLUDING KEYWORD

Explanation: A keyword parameter in a macro call is not defined in the corresponding macro prototype statement.

Note: This message may be 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.

Severity: 4

IEV018 DUPLICATE KEYWORD IN MACRO CALL. LAST VALUE IS USED

Explanation: A keyword operand occurs more than once in a macro call.

System Action: The latest value assigned to the keyword is used.

Programmer Response: Eliminate one of the keyword operands.

Severity: 12

IEV020 ILLEGAL GBL OR LCL STATEMENT

Explanation: A global (GBL) or local (LCL) declaration statement does not have an operand.

System Action: The statement is ignored.

Programmer Response: Remove the statement or add an operand.

Severity: 8

IEV021 ILLEGAL SET STATEMENT

Explanation: The operand of a SETB statement is not 0, 1, or a SETB expression enclosed in parentheses.

System Action: The statement is ignored.

Programmer Response: Correct the operand or delete the statement.

Severity: 8

IEV023 SYMBOLIC PARAMETER TOO LONG

Explanation: A symbolic parameter in this statement is too long. It must not exceed 63 characters, including the initial ampersand.

System Action: The symbolic parameter and any operand following it in this statement are ignored.

Programmer Response: Make sure all symbolic parameters consist of an ampersand followed by 1 to 62 alphameric characters, the first of which is alphabetic.

Severity: 8

IEV024 INVALID VARIABLE SYMBOL

Explanation: One of these errors has occurred:

- A symbolic parameter or a SET symbol is not an ampersand followed by 1 to 62 alphameric 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

IEV025 INVALID MACRO PROTOTYPE OPERAND

Explanation: The format of the operand field of a macro prototype statement is invalid. For example, two parameters are not separated by a comma, or a parameter contains an invalid character.

System Action: The operand field of the prototype is ignored.

Programmer Response: Supply a valid operand field.

Severity: 12

IEV026 MACRO CALL OPERAND TOO LONG. 255 LEADING CHARACTERS DELETED

Explanation: An operand of a macro instruction is more than 255 characters long.

System Action: The leading 255 characters are deleted.

Programmer Response: Limit the operand to 255 characters, or limit it into two or more operands.

Severity: 12

IEV027 EXCESSIVE NUMBER OF OPERANDS

Explanation: One of the following has occurred:

- More than 240 positional and/or keyword operands 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

IEV028 INVALID DISPLACEMENT

Explanation: One of the following has occurred:

- The displacement field of an explicit address is not an absolute value within the range 0 through 4095.
- The displacement field of an S-type address constant is not an absolute value within the range 0 through 4095.

System Action: The statement or constant is assembled as zero.

Programmer Response: Correct the displacement or supply an appropriate USING statement containing an absolute first operand prior to this statement.

Severity: 8

IEV029 INCORRECT REGISTER OR MASK SPECIFICATION

Explanation: The value specifying a register or a mask is not an absolute value within the range 0 through 15; an odd register is used where an even register is required; a register is used where none can be specified; or a register is not specified where one is required.

System Action: For machine instructions and S-type address constants, the statement or constant is assembled as zero. For USING and DROP statements, the invalid register operand is ignored.

Programmer Response: Specify a valid register.

Severity: 8

IEV030 INVALID LITERAL USAGE

Explanation: A literal is used in an assembler instruction, another literal, or a field of a machine instruction where it is not permitted.

System Action: An assembler instruction containing a literal is generally ignored and another message, relative to the operation code of the instruction, appears. A machine instruction is assembled to zero.

Programmer Response: If applicable, replace the literal with the name of a DC statement.

Severity: 8

IEV031 INVALID IMMEDIATE FIELD

Explanation: The value of an immediate operand of a machine instruction requires more than one byte of storage (exceeds 255) or the value of the immediate operand exceeds 9 on an SRP instruction.

System Action: The instruction is assembled as zero.

Programmer Response: Use a valid immediate operand, or specify the immediate information in a DC statement or a literal and change the statement to a nonimmediate type.

Severity: 8

IEV032 RELOCATABLE VALUE FOUND WHERE ABSOLUTE VALUE REQUIRED

Explanation: A relocatable or complex relocatable expression is used where an absolute expression is required.

System Action: A machine instruction is assembled 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.

Severity: 8

IEV033 ALIGNMENT ERROR

Explanation: An address referenced by this statement may not be aligned to the proper boundary for this instruction; for example, the data referenced by a load instruction (L) may be on a halfword boundary, or the address may depend upon an index register.

System Action: The instruction is assembled as written.

Programmer Response: Correct the operand if it is in error. If you are using a System/370 model that does not require alignment or you wish to suppress alignment checking for some other reason, you can specify 'NOALIGN' as an assembler option. If a particular statement is correct, you can suppress this message by writing the statement with an absolute displacement and an explicit base register, as in this example:

```
  L   1,SYM-BASE(,2)
```

Severity: 4

IEV034 ADDRESSABILITY ERROR

Explanation: The address referenced by this statement does not fall within the range of a USING statement, or a base register is specified along with a relocatable displacement.

System Action: The instruction is assembled as zero.

Programmer Response: Insert the appropriate USING statement prior to this statement. Otherwise, check this statement for a misspelled symbol, an unintended term or symbol in an address expression, or a relocatable symbol used as a displacement.

Severity: 8

IEV035 INVALID DELIMITER

Explanation: (1) A required delimiter in a DC, DS, or DXD statement is missing or appears where none should be; the error may be any of these:

- A quotation mark with an address constant.
- A left parenthesis with a nonaddress constant.
- A constant field not started with a quotation mark, left parenthesis, blank, or comma.
- An empty constant field in a DC.
- A missing comma or right parenthesis following an address constant.
- A missing subfield right parenthesis in an S-type address constant.
- A missing right parenthesis in a constant modifier expression.

(2) A parameter in a macro prototype statement was not followed by a valid delimiter: comma, equal sign, or blank.

System Action: The operand or parameter in error and the following operands or parameters are ignored.

Programmer Response: Supply a valid delimiter.

Severity: 12

IEV036 REENTRANT CHECK FAILED

Explanation: A machine instruction that might store data into a control section or common area when executed has been detected. This message is generated only when reentrant checking is requested by the assembler option 'RENT'.

System Action: The statement is assembled as written.

Programmer Response: If you want reentrant code, correct the instruction. Otherwise, you can suppress reentrant checking by specifying 'NORENT' as an assembler option.

Severity: 4

IEV037 ILLEGAL SELF-DEFINING VALUE

Explanation: A decimal, binary (B), hexadecimal (X), or character (C) self-defining term contains invalid characters 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 entire statement is ignored.

Programmer Response: Supply a valid self-defining term.

Severity: 8

IEV038 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. Note that 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 proper section before the ORG statement is executed.

Severity: 12

IEV039 LOCATION COUNTER ERROR

Explanation: The location counter has exceeded 2^{24}-1, the largest address that can be contained in 3 bytes. This occurrence is called location counter wraparound.

System Action: The location counter is 4 bytes long (only 3 bytes appear in the listing and the object deck). The overflow is carried into the high-order byte and the assembly continues. However, the resulting code will probably not execute 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

IEV040 MISSING OPERAND

Explanation: The statement requires an operand, and none is present.

System Action: A machine instruction is assembled as zero. An assembler instruction is ignored.

Programmer Response: Supply the missing operand.

Severity: 12

IEV041 TERM EXPECTED. TEXT IS UNCLASSIFIABLE

Explanation: One of these errors has occurred:

- A term was expected, but the character encountered is not one that starts a term (letter, number, =, +, -, *).
- A letter and a quotation mark did not introduce a valid term; the letter is not L, C, X, or B.

System Action: Another message will accompany an assembler statement. A machine instruction will be assembled as zero.

Programmer Response: Check for missing punctuation, a wrong letter on a self-defining term, a bad attribute request, a leading comma, or a dangling comma. Note that the length attribute is the only one accepted here. If a scale, type, or integer attribute is needed, use a SETA statement and substitute the variable symbol where the attribute is needed.

Severity: 8

IEV042 LENGTH ATTRIBUTE OF UNDEFINED SYMBOL. DEFAULT=1

Explanation: This statement has a length attribute reference to an undefined symbol.

System Action: The L' attribute defaults to 1.

Programmer Response: Define the symbol that was referenced.

Severity: 8

IEV043 PREVIOUSLY DEFINED SYMBOL

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 will be processed. The first occurrence of the symbol will define it.

Programmer Response: Correct a possible spelling error, or change the symbol.

Severity: 8

IEV044 UNDEFINED SYMBOL

Explanation: A symbol in the operand field has not been defined, that is, used in the name field of another statement or the operand field of an EXTRN or WXTRN.

System Action: A machine instruction or an address constant is assembled 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 ignored entirely.

Programmer Response: Define the symbol, or remove the references to it.

Severity: 8

IEV045 REGISTER NOT PREVIOUSLY USED

Explanation: A register specified in a DROP statement has not been previously specified in a USING statement.

System Action: Registers not currently active are ignored.

Programmer Response: Remove the unreferenced registers from the DROP statement. You can drop all active base registers at once by specifying DROP with a blank operand.

Severity: 4

IEV046 FLAG BYTE OPERAND IS NOT A MULTIPLE OF 4

Explanation: Bits 6 and 7 of the flag byte of a channel command word specified by a CCW, CCW0, or CCW1 statement are not all zero.

System Action: The CCW, CCW0, or CCW1 is assembled as zero.

Programmer Response: Set bits 6 and 7 of the flag byte to zero to suppress this message during the next assembly.

Severity: 8

IEV047 SEVERITY CODE TOO LARGE

Explanation: The severity code (first operand) of an MNOTE statement is not * or an unsigned decimal number from 0 to 255.

System Action: The statement is printed in standard format instead of MNOTE format. The MNOTE is given the severity code of this message.

Programmer Response: Choose a severity code of * or a number less than 255, or check for a generated severity code.

Severity: 8

IEV048 ENTRY ERROR

Explanation: One of the following errors was detected in the operand of an ENTRY statement:

- Duplicate symbol (previous ENTRY)
- Symbol defined in a DSECT or COM section
- Symbol defined by a DXD statement
- Undefined symbol
- Symbol defined by an absolute or complex relocatable EQU statement

System Action: The external symbol dictionary output is suppressed for the symbol.

Programmer Response: Define the ENTRY operand correctly.

Severity: 8

IEV049 ILLEGAL RANGE ON ISEQ

Explanation: If this message is accompanied by another, this one is advisory. If it appears by itself, it indicates one of the following errors:

- An operand value is less than 1 or greater than 80, or the second operand (rightmost column to be checked) is less than the first operand (leftmost column to be checked).
- More or fewer than two operands are present, or an operand is null (empty).
- An operand expression contains an undefined symbol.
- An operand expression is not absolute.
- The statement is too complex. For example, it may have forward references or cause an arithmetic overflow during evaluation.
- The statement is circularly defined.

System Action: Sequence checking is stopped.

Programmer Response: Supply valid ISEQ operands. Also, be sure that the cards following this statement are in order; they have not been sequence checked.

Severity: 4

IEV050 ILLEGAL NAME FIELD. NAME DISCARDED

Explanation: One of these errors has occurred:

- The name field of a macro prototype statement contains an invalid symbolic parameter (variable symbol).
- The name field of a COPY statement in a macro definition contains an entry other than blank or a valid sequence symbol.

System Action: The invalid name field is ignored.

Programmer Response: Correct the invalid name field.

Severity: 8

IEV051 ILLEGAL STATEMENT OUTSIDE A MACRO DEFINITION

Explanation: A MEND, MEXIT, or AREAD statement appears outside a macro definition.

System Action: The statement is ignored.

Programmer Response: Remove the statement or, if a macro definition is intended, insert a MACRO statement.

Severity: 8

IEV052 CARD OUT OF SEQUENCE

Explanation: Input sequence checking, under control of the ISEQ assembler instruction, has determined that this statement is out of sequence. The sequence number of the statement is appended to the message.

System Action: The statement is assembled normally. However, the sequence number of the next statement will be checked relative to this statement.

Programmer Response: Put the statements in proper sequence. If you want a break in sequence, put in a new ISEQ statement and sequence number. ISEQ always resets the sequence number; the card following the ISEQ is not sequence checked.

Severity: 12

IEV053 BLANK SEQUENCE FIELD

Explanation: Input sequence checking, controlled by the ISEQ assembler statement, has detected a statement with a blank sequence field. The sequence number of the last numbered statement is appended to the message.

System Action: The statement is assembled normally. The sequence number of the next statement will be checked relative to the last statement having a nonblank sequence field.

Programmer Response: Put the proper sequence number in the statement or discontinue sequence checking over the blank statements by means of an ISEQ statement with a blank operand.

Severity: 4

IEV054 ILLEGAL CONTINUATION CARD

Explanation: A statement has more than 10 cards or end-of-input has been encountered when a continuation card was expected.

System Action: The cards already read are processed as is. If the statement had more than 10 cards, the next card 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 cards or an extraneous continuation punch.

Severity: 8

IEV055 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

IEV057 UNDEFINED OPERATION CODE

Explanation: One of the following errors has occurred:

- The operation code of this statement is not a valid machine or assembler instruction or macro name.
- In an OPSYN statement, this operand symbol is undefined or illegal or,

if no operand is present, the name field symbol is undefined.

System Action: The statement is ignored. Note that OPSYN does not search the macro library for an undefined operand.

Programmer Response: Correct the statement. In the case of an undefined macro instruction, the wrong data set may have been specified for the macro library. In the case of OPSYN, a previous OPSYN or macro definition may have failed to define the operation code.

Severity: 8

IEV059 ILLEGAL ICTL

Explanation: An ICTL statement has one of the following errors:

- The operation code was created by variable symbol substitution.
- It is not the first statement in the assembly.
- The value of one or more operands is incorrect.
- An operand is missing.
- An invalid character is detected in the operand field.

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

IEV060 COPY CODE NOT FOUND

Explanation: (1) If this message is on a COPY statement and no text is printed with it, one of the following occurred:

- The library member was not found.
- The lookahead phase previously processed the COPY statement and did not find the library member, the copy was recursive, or the operand contains a variable symbol.

(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 the library member may be read by the lookahead phase of the assembler, do not make the library member name a variable symbol.

If COPY member is not defined in any macro library, and is not executed because of an AGO or AIF assembler instruction, add a dummy COPY member with the name to the macro library.

Severity: 12

IEV061 SYMBOL NOT NAME OF DSECT OR DXD

Explanation: The operand of a Q-type address constant is not a symbol or the name of a DSECT or DXD statement.

System Action: The constant is assembled as zero.

Programmer Response: Supply a valid operand.

Severity: 8

IEV062 ILLEGAL OPERAND FORMAT

Explanation: One of the following errors has occurred:

- DROP or USING—more than 16 registers were specified in the operand field.
- PUSH or POP—an operand does not specify a PRINT or USING statement.
- PRINT—an operand specifies an invalid print option.
- MNOTE—the syntax of the severity code (first operand) is invalid.
- AMODE—the operand does not specify 24, 31, or ANY.
- RMODE—the operand does not specify 24 or ANY.

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) will not appear in the cross-reference listing.

Programmer Response: Supply a valid operand field.

Severity: 8

IEV063 NO ENDING APOSTROPHE

Explanation: The quotation mark terminating an operand is missing, or the standard value of a keyword parameter of a macro prototype statement is missing.

System Action: The operand or standard value in error is ignored. If the error is in a macro definition model statement, the entire statement is ignored.

Programmer Response: Supply the missing quotation mark.

Severity: 8

IEV064 FLOATING POINT CHARACTERISTIC OUT OF RANGE

Explanation: A converted floating-point constant is too large or too small for the processor. The allowable range is 7.2×10^{75} to 5.3×10^{-77}.

System Action: The constant is assembled 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

IEV065 UNKNOWN TYPE

Explanation: An unknown constant type has been used in a DC or DS statement or in a literal.

System Action: The operand in error and the following operands are ignored.

Programmer Response: Supply a valid constant. Look for an incorrect type code or incorrect syntax in the duplication factor.

Severity: 8

IEV066 RELOCATABLE Y-TYPE CONSTANT

Explanation: This statement contains a relocatable Y-type address constant. A Y-constant is only 2 bytes long, so addressing errors will occur if this program is loaded at a main storage address greater than 32K (32,768).

System Action: The statement is assembled as written.

Programmer Response: If this program will not be loaded at a main storage address greater than 32K, you can leave the Y-constant.

Severity: 4

IEV067 ILLEGAL DUPLICATION FACTOR

Explanation: One of the following errors has occurred:

- A literal has a zero duplication factor.
- The duplication factor of a constant is greater than $2^{24}-1$.
- A duplication factor expression of a constant is invalid.

System Action: The operand in error and the following operands of a DC, DS, or DXD statement are ignored. The statement containing the literal is assembled 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

IEV068 LENGTH ERROR

Explanation: One of the following errors has occurred:

- The length modifier of a constant is wrong.
- The C, X, B, Z, or P-type constant is too long.
- An operand is longer than $2^{24}-1$ bytes.
- A relocatable address constant has an illegal length.
- The length field in a machine instruction is invalid or out of the permissible range.

System Action: The operand in error and the following operands of the DC, DS, or DXD statement are ignored, except that an address constant with an illegal length is truncated. A machine instruction is assembled as zero.

Programmer Response: Supply a valid length.

Severity: 12

IEV070 SCALE MODIFIER ERROR

Explanation: A scale modifier in a constant is used illegally, is out of range, or is relocatable, or there is an error in a scale modifier expression.

System Action: If the scale modifier is out of range, it defaults to zero. Otherwise, the operand in error and the following operands are ignored.

Programmer Response: Supply a valid scale modifier.

Severity: 8

IEV071 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 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

IEV072 DATA ITEM TOO LARGE

Explanation: A Y-type address constant is larger than $2^{15}-1$ or smaller than -2^{15}, or the value of a decimal constant is greater than the number of bits (integer attribute) allocated to it.

System Action: The constant is truncated. The high-order bits are lost.

Programmer Response: Supply a smaller scale modifier or a longer constant.

Severity: 8

IEV073 PRECISION LOST

Explanation: The scale modifier of a floating-point number was large enough to shift the entire fraction out of the converted constant.

System Action: The constant is assembled with an exponent but with a zero mantissa (fraction).

Programmer Response: Change the scale modifier or use a longer constant. For example, use a D-type constant instead of an E-type constant.

Severity: 8

IEV074 ILLEGAL SYNTAX IN EXPRESSION

Explanation: An expression has two terms or two operations in succession, or invalid 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 entire statement is ignored. A machine instruction is assembled as zero.

Programmer Response: Check the expression for keypunch errors, or for missing or invalid terms or characters.

Severity: 8

IEV075 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 is assembled as zero. An assembler instruction is ignored.

Programmer Response: Change the expression.

Severity: 8

IEV076 STATEMENT COMPLEXITY EXCEEDED

Explanation: The complexity of this statement caused the assembler's expression evaluation work area to overflow.

System Action: A machine instruction is assembled 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

IEV077 CIRCULAR DEFINITION

Explanation: The value of a symbol in an expression is dependent on itself, either directly or indirectly, via one or more EQU statements. For 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

IEV079 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

IEV080 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

IEV081 CREATED SET SYMBOL EXCEEDS 63 CHARACTERS

Explanation: A SET symbol created by variable symbol substitution is longer than 63 characters (including the ampersand as the first character).

System Action: If the symbol is in the operand field of a SET, AIF, or AGO statement, its value is set to zero or null, and the type attribute is set to undefined (U). If the symbol is in the operand field of a GBL, or LCL statement or the name field of a SET statement, the macro is aborted.

Programmer Response: Shorten the symbol.

Severity: 8

IEV082 CREATED SET SYMBOL IS NULL

Explanation: A SET symbol created by variable symbol substitution is null (empty string).

System Action: If the symbol is in the operand field of a SET, AIF, or AGO statement, its value is set to zero or null, and the type attribute is set to undefined (U). If the symbol is in the operand field of a GBL, or LCL statement or the name field of a SET statement, the macro is aborted.

Programmer Response: Supply a valid symbol.

Severity: 8

IEV083 CREATED SET SYMBOL IS NOT A VALID SYMBOL

Explanation: A SET symbol created by variable symbol substitution or concatenation does not consist of an ampersand followed by up to 62 alphameric 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, the macro is aborted.

Programmer Response: Supply a valid symbol.

Severity: 8

IEV084 GENERATED NAME FIELD EXCEEDS 63 CHARACTERS. DISCARDED

Explanation: The name field on a generated statement is longer than 63 characters.

System Action: The name field is not generated. The rest of the statement is assembled normally.

Programmer Response: Shorten the generated name to 63 characters or fewer.

Severity: 12

IEV085 GENERATED OPERAND FIELD IS NULL

Explanation: The operand field of a generated statement is null (empty).

System Action: The statement is assembled as though no operand were specified.

Programmer Response: Provide a nonempty operand field. If you want the statement assembled with no operand, substitute a comma rather than leave the operand blank.

Severity: 0

IEV086 MISSING MEND GENERATED

Explanation: A macro definition, appearing in the source program or being read from a library by a macro call or a COPY statement, ends before a MEND statement is encountered to terminate it.

System Action: A MEND statement is generated. The portion of the macro definition read in will be processed.

Programmer Response: Insert the MEND statement if it was left out. Otherwise, check if all the macro definition is on the library.

Severity: 12

IEV087 GENERATED OPERATION CODE IS NULL

Explanation: The operation code of a generated statement is null (blank).

System Action: The generated statement is printed but not assembled.

Programmer Response: Provide a valid operation code.

Severity: 12

IEV088 UNBALANCED PARENTHESES IN MACRO CALL OPERAND

Explanation: Excess left or 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

IEV089 ARITHMETIC EXPRESSION CONTAINS ILLEGAL DELIMITER OR ENDS PREMATURELY

Explanation: An arithmetic expression contains an invalid character or an arithmetic subscript ends without sufficient right parentheses.

System Action: The statement is ignored.

Programmer Response: Supply a valid expression.

Severity: 8

IEV090 EXCESS RIGHT PARENTHESIS IN MACRO CALL OPERAND

Explanation: A right parenthesis without a corresponding left parenthesis was detected in an operand of a macro instruction.

System Action: The excess right parenthesis is ignored. The macro expansion may be incorrect.

Programmer Response: Insert the proper parenthesis.

Severity: 8

IEV091 SETC OR CHARACTER RELATIONAL OPERAND OVER 255 CHARACTERS. TRUNCATED TO 255 CHARACTERS

Explanation: The value of the operand of a SETC statement or the character relational operand of an AIF statement is longer than 255 characters. This may occur before substrings are evaluated.

System Action: The first 255 characters are used.

Programmer Response: Shorten the SETC expression value or the operand value.

Severity: 8

IEV092 SUBSTRING EXPRESSION 1 POINTS PAST STRING END DEFAULT=NULL

Explanation: The first arithmetic expression of a SETC substring points beyond the end of the expression character string.

System Action: The substring is given a null value.

Programmer Response: Supply a valid expression.

Severity: 0

IEV093 SUBSTRING EXPRESSION 1 LESS THAN 1. DEFAULT=NULL

Explanation: The first arithmetic expression of a SETC substring is less than one; that is, it points before the expression character string.

System Action: The substring expression defaults to null.

Programmer Response: Supply a valid expression.

Severity: 8

IEV094 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.

Severity: 0

IEV095 SUBSTRING EXPRESSION 2 LESS THAN 0. DEFAULT=NULL

Explanation: The second arithmetic expression of a SETC substring is less than or equal to zero.

System Action: No characters (a null string) from the substring character expression are used.

Programmer Response: Supply a valid expression.

Severity: 4

IEV096 UNSUBSCRIPTED SYSLIST. DEFAULT=SYSLIST(1)

Explanation: The system variable symbol, &SYSLIST, is not subscripted. &SYSLIST(n) refers to the nth positional parameter in a macro instruction. Note that N'&SYSLIST does not have to be subscripted.

System Action: The subscript defaults to one so that the first positional parameter will be referred to.

Programmer Response: Supply an appropriate subscript.

Severity: 8

IEV097 INVALID ATTRIBUTE REFERENCE TO SETA OR SETB SYMBOL. DEFAULT=U OR 0

Explanation: A type (T'), 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: T'=U, L'=0, S'=0, and D'=0.

Programmer Response: Change or remove the attribute reference.

Severity: 8

IEV098 ATTRIBUTE REFERENCE TO INVALID SYMBOL. DEFAULT=U OR 0

Explanation: An attribute attempted to reference an invalid symbol. (A valid symbol is 1 to 63 alphameric 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

IEV099 WRONG TYPE OF CONSTANT FOR S' OR I' ATTRIBUTE REFERENCE. DEFAULT=0

Explanation: An integer (I') or scaling (S') attribute references a symbol whose type is other than floating-point (E,D,L), decimal (P,Z), or fixed-point (H,F).

System Action: The integer or scaling attribute defaults to zero.

Programmer Response: Remove the integer or scaling attribute reference or change the constant type.

Severity: 4

IEV100 SUBSCRIPT LESS THAN 1. DEFAULT TO SUBSCRIPT = 1.

Explanation: The subscript of a subscripted SET symbol in the name field of a SET statement, the operand field of a GBL or LCL statement, or an &SYSLIST statement is less than 1.

System Action: The subscript defaults to 1.

Programmer Response: Supply the correct subscript.

Severity: 8

IEV101 SUBSCRIPT LESS THAN 1. DEFAULT TO VALUE=0 OR NULL

Explanation: The subscript of a SET symbol in the operand field is less than 1.

System Action: The subscript is set to 1.

Programmer Response: Supply a valid subscript.

Severity: 8

IEV102 ARITHMETIC TERM IS NOT SELF-DEFINING TERM. DEFAULT=0

Explanation: A SETC term or expression used as an arithmetic term is not a 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. Note that the C, X, or B and the quotation marks must be part of the SETC value.

Severity: 8

IEV103 MULTIPLICATION OVERFLOW. DEFAULT PRODUCT=1

Explanation: A multiplication overflow occurred in a macro definition statement.

System Action: The value of the expression up to the point of overflow is set to one; evaluation is resumed.

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

IEV105 ARITHMETIC EXPRESSION TOO COMPLEX

Explanation: An arithmetic expression in a macro definition statement caused an overflow because it is too complex; that is, it has too many terms and/or levels.

System Action: The assembly is terminated.

Programmer Response: Simplify the expression or break it into two or more expressions.

Severity: 20

IEV106 WRONG TARGET SYMBOL TYPE. VALUE LEFT UNCHANGED

Explanation: The SET symbol in the name field does not match its declared type (does not match the operation code): SETA, SETB, or SETC.

System Action: The statement is ignored.

Programmer Response: Make the declaration agree with the SET statement type. If you want to store across types, store first into a SET symbol of matching type.

Severity: 8

IEV107 INCONSISTENT DIMENSION ON TARGET SYMBOL. SUBSCRIPT IGNORED OR 1 USED

Explanation: The SET symbol in the name field is dimensioned (subscripted), but was not declared in a GBL or LCL statement as dimensioned, or vice versa.

System Action: The subscript is ignored or a subscript of 1 is used, in accordance with the declaration.

Programmer Response: Make the declaration and the usage compatible. Note that you can declare a local SET symbol as dimensioned by using it, subscripted, in the name field of a SET statement.

Severity: 8

IEV108 INCONSISTENT DIMENSION ON SET SYMBOL REFERENCE. DEFAULT = 0, NULL, OR TYPE = U

Explanation: A SET symbol in the operand field is dimensioned (subscripted), but was not declared in a GBL or LCL statement as dimensioned, or vice versa.

System Action: A value of zero or null is used for the subscript. If the type attribute of the SET symbol is being requested, it is set to U.

Programmer Response: Make the declaration and the usage compatible. Note that you can declare a SET symbol as dimensioned by using it, subscripted, in the name field of a SET statement.

Severity: 8

IEV109 MULTIPLE OPERANDS FOR UNDIMENSIONED SET SYMBOL. GETS LAST OPERAND

Explanation: Multiple operands were assigned to an undimensioned (unsubscripted) SET symbol.

System Action: The SET symbol is given the value of the last operand.

Programmer Response: Declare the SET symbol as dimensioned, or assign only one operand to it.

Severity: 8

IEV110 LIBRARY MACRO 1ST 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

IEV111 INVALID AIF OR SETB OPERAND FIELD

Explanation: The operand of an AIF or SETB statement either does not begin with a left parenthesis or is missing altogether.

System Action: The statement is ignored.

Programmer Response: Supply a valid operand.

Severity: 12

IEV112 INVALID SEQUENCE SYMBOL

Explanation: One of the following errors has occurred:

- A sequence symbol doesn't begin with a period followed by one to 62 alphameric characters, the first being alphabetic.
- A sequence symbol in the name field was created by substitution.
- A sequence symbol contains an underscore character.

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 entire statement to be ignored.

Programmer Response: Supply a valid sequence symbol.

Severity: 12

IEV113 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 card, but the continue column (normally column 16) of this card is blank.

System Action: This card and any following cards of the statement are ignored. Any SET symbols appearing entirely on the previous card(s) are processed normally.

Programmer Response: Begin this card in the continuation column.

Severity: 12

IEV114 INVALID COPY OPERAND

Explanation: The operand of a COPY statement is not a symbol of 1 to 8 alphameric characters, the first being alphabetic.

System Action: The COPY statement is ignored.

Programmer Response: Supply a valid operand.

Severity: 12

IEV115 COPY OPERAND TOO LONG

Explanation: The symbol in the operand field of a COPY statement is more than 8 characters long.

System Action: The COPY statement is ignored.

Programmer Response: Supply a valid operand.

Severity: 12

IEV116 ILLEGAL SET SYMBOL

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 alphameric characters, the first being alphabetic.

System Action: The invalid SET symbol and all following SET symbols in a GBL or LCL statement are ignored. The entire SET statement is ignored.

Programmer Response: Supply a SET symbol.

Severity: 8

IEV117 ILLEGAL SUBSCRIPT

Explanation: The subscript following a SET symbol contained unbalanced parentheses or an invalid 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

IEV118 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. This terminated the definition.

System Action: The MEND statement is ignored. No more COPY code is read. The statements brought in before the end of the COPY code are processed. The macro definition is resumed with the statement following the COPY statement.

Programmer Response: Make sure that each library member to be used as COPY code contains balanced MACRO and MEND statements.

Severity: 12

IEV119 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 terminate 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

IEV120 EOD WHERE CONTINUE CARD EXPECTED

Explanation: An end-of-data occurred when a continuation card was expected.

System Action: The portion of the statement read in is assembled. The assembly is terminated if the end-of-data is on SYSIN. 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

IEV121 INSUFFICIENT CORE 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 is terminated.

Programmer Response: Split the assembly into two or more parts or give the macro editor more working storage. This can be done by increasing the region size for the assembler, decreasing blocking factor or block size on the assembler data sets, or a combination of both.

Severity: 12

IEV122 ILLEGAL OPERATION CODE FORMAT

Explanation: The operation code is not followed by a blank or is missing altogether, or the first card 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 cards of the statement are present.

Severity: 12

IEV123 VARIABLE SYMBOL TOO LONG

Explanation: A SET symbol, symbolic parameter, or sequence symbol contains more than 62 characters following the ampersand or period.

System Action: This statement is ignored.

Programmer Response: Shorten the variable symbol or sequence symbol.

Severity: 12

IEV124 ILLEGAL USE OF PARAMETER

Explanation: A symbolic parameter was used in the operand field of a GBL or LCL statement or in the name field of a SET statement. In other words, a variable symbol has been used both as a symbolic parameter and as a SET symbol.

System Action: The statement is ignored.

Programmer Response: Change the variable symbol to one that is not a symbolic parameter.

Severity: 12

IEV125 ILLEGAL MACRO NAME - MACRO UNCALLABLE

Explanation: The operation code of a macro prototype statement is not a valid symbol; that is, one to 63 alphameric characters, the first alphabetic.

System Action: The macro definition is edited. However, since the macro name is invalid, the macro cannot be called.

Programmer Response: Supply a valid macro name.

Severity: 12

IEV126 LIBRARY MACRO NAME INCORRECT

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 will usually require listing the macro definition from the library.

Severity: 12

IEV127 ILLEGAL USE OF AMPERSAND

Explanation: One of the following errors has occurred:

- An ampersand was found where all substitution should have already been performed.
- The standard value of a keyword parameter in a macro prototype statement contained a single ampersand or a string of ampersands whose length was odd.
- 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 constants occur in pairs. Also, avoid substituting an ampersand into a statement unless there is a double ampersand.

Severity: 12

IEV128 EXCESS RIGHT PARENTHESIS

Explanation: An unpaired right parenthesis has been found.

System Action: A machine instruction is assembled 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

IEV129 INSUFFICIENT RIGHT PARENTHESES

Explanation: An unpaired left parenthesis has been found. Note that parentheses must balance at each comma in a multiple operand statement.

System Action: A machine instruction is assembled as zero. An assembler instruction is ignored and an additional message relative to the statement type will appear. 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

IEV130 ILLEGAL ATTRIBUTE REFERENCE

Explanation: One of the following errors has occurred:

- The symbol following a D, I, L, S, or T attribute reference is not a valid variable symbol or ordinary symbol.
- The symbol following a K or N attribute reference is not a valid variable symbol.
- The quote is missing from a T attribute reference.

System Action: The statement is ignored.

Programmer Response: Supply a valid attribute reference.

Severity: 12

IEV131 PARENTHESIS NESTING DEPTH EXCEEDS 255

Explanation: There are more than 255 levels of parentheses in a SETA expression.

System Action: The statement is ignored.

Programmer Response: Rewrite the SETA statement using several statements to regroup the subexpressions in the expression.

Severity: 12

IEV132 INVALID SETB EXPRESSION

Explanation: A SETB expression in the operand field of a SETB statement or an AIF statement does not consist of valid character relational expressions, arithmetic relational expressions, and single SETB symbols, connected by logical operators.

System Action: The statement is ignored.

Programmer Response: Supply a valid SETB expression.

Severity: 12

IEV133 ILLEGAL SUBSTRING REFERENCE

Explanation: A substring expression following a SETC expression does not consist of two valid SETA expressions separated by a comma and enclosed in parentheses.

System Action: The statement is ignored.

Programmer Response: Supply a valid substring expression.

Severity: 12

IEV134 INVALID RELATIONAL OPERATOR

Explanation: Characters other than EQ, NE, LT, GT, LE, or GE are used in a SETB expression where a relational operator is expected.

System Action: The statement is ignored.

Programmer Response: Supply a valid relational operator.

Severity: 12

IEV135 INVALID LOGICAL OPERATOR

Explanation: Characters other than AND, OR, or NOT 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

IEV136 ILLEGAL LOGICAL/RELATIONAL OPERATOR

Explanation: Characters other than a valid logical or relational operator are used in a SETB expression where a logical or relational operator is expected.

System Action: The statement is ignored.

Programmer Response: Supply a valid logical or relational operator.

Severity: 12

IEV137 ILLEGAL SETC EXPRESSION

Explanation: The operand of a SETC statement or the character value used in a character relation is erroneous. It must be a valid type attribute (T') reference or a valid character expression enclosed in quotation marks.

System Action: The statement is ignored.

Programmer Response: Supply a valid expression.

Severity: 12

IEV139 EOD DURING REPRO PROCESSING

Explanation: A REPRO statement was immediately followed by an end-of-data so that no valid card could be punched. The REPRO is either the last card of source input or the last card of a COPY member.

System Action: The REPRO statement is ignored.

Programmer Response: Remove the REPRO or ensure that it is followed by a card to be punched.

Severity: 12

IEV140 END CARD MISSING

Explanation: End-of-file on the source input data set occurred before an END statement was read. One of the following has occurred:

- The END statement was omitted or misspelled.
- The END operation code was changed or deleted by OPSYN or by definition of a macro named END. The lookahead phase of the assembler marks what it thinks is the END statement. If an OPSYN statement or a macro definition redefines the END statement, premature end-of-input may occur because the assembler will 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 will be processed as usual following the END statement.

Programmer Response: Check for lost cards. Supply a valid END statement; or, if you use OPSYN to define another symbol as END, place it prior to possible entry into the lookahead phase.

Severity: 4

IEV141 BAD CHARACTER IN OPERATION CODE

Explanation: The operation code contains a nonalphameric character, that is, a character other than A to Z, 0 to 9, $, #, or @. Embedded blanks are not allowed.

System Action: The statement is ignored.

Programmer Response: Supply a valid operation code. If the operation code is formed by variable symbol substitution, check the statements leading to substitution.

Severity: 8

IEV142 OPERATION CODE NOT COMPLETE ON FIRST CARD

Explanation: The entire name and operation code, including a trailing blank, is not contained on the first card (before the continue column—usually column 72) of a continued statement.

System Action: The statement is ignored.

Programmer Response: Shorten the name and/or the operation code or simplify the statement by using a separate SETC statement to create the name or operation code by substitution.

Severity: 8

IEV143 BAD CHARACTER IN NAME FIELD

Explanation: The name field contains a nonalphameric character, that is, a character other than A to Z, 0 to 9, $, #, @, or _. (**Note:** _ is invalid for external names or in the name field of an OPSYN instruction.)

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

IEV144 BEGIN-TO-CONTINUE COLUMNS NOT BLANK

Explanation: On a continuation card, one or more columns between the begin column (usually column 1) and the continue column (usually column 16) are not blank.

System Action: The extraneous characters are ignored.

Programmer Response: Check whether the operand started in the wrong column or whether the preceding card contained an erroneous continue punch.

Severity: 8

IEV145 OPERATOR, RIGHT PARENTHESIS, OR END-OF-EXPRESSION EXPECTED

Explanation: One of the following has occurred:

- A letter, number, equal sign, quotation mark, or undefined character occurred following a term where a right parenthesis, an operator, a comma, or a blank ending the expression was expected.
- In an assembler instruction, a left parenthesis followed a term.

System Action: A machine instruction is assembled as zero. An assembler instruction is ignored and another message, relative to the operation code, is issued.

Programmer Response: Check for an omitted or mispunched operator. Subscripting is not allowed on this statement.

Severity: 8

IEV146 SELF-DEFINING TERM TOO LONG OR VALUE TOO LARGE

Explanation: A self-defining term is longer than 4 bytes, (8 hexadecimal digits, 32 bits, or 4 characters), or the value of a decimal self-defining term is greater than $2^{31}-1$.

System Action: A machine instruction is assembled 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

IEV147 SYMBOL TOO LONG, OR 1ST CHARACTER NOT A LETTER

Explanation: A symbol does not begin with a letter or is longer than 63 characters.

System Action: If the symbol is in the name field, the statement is processed as unnamed. If the symbol is in the operand field, an assembler operation or a macro definition model statement is ignored and a machine operation is assembled as zero.

Programmer Response: Supply a valid symbol.

Severity: 8

IEV148 SELF-DEFINING TERM LACKS ENDING QUOTE OR HAS BAD CHARACTER

Explanation: A hexadecimal or binary self-defining term contains an invalid character or is missing the final quotation mark.

System Action: A machine operation is assembled as zero. An assembler operation is ignored and another message, relative to the operation code, is issued.

Programmer Response: Correct the invalid term.

Severity: 8

IEV149 LITERAL LENGTH EXCEEDS 256 CHARACTERS, INCLUDING EQUAL SIGN

Explanation: A literal is longer than 256 characters.

System Action: The instruction is assembled as zero.

Programmer Response: Shorten the literal, or change it to a DC statement.

Severity: 8

IEV150 SYMBOL HAS NON-ALPHAMERIC CHARACTER OR INVALID DELIMITER

Explanation: The first character following a symbol is not a valid delimiter (plus sign, minus sign, asterisk, slash, left or right parenthesis, comma, or blank).

System Action: A machine operation is assembled 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 nonalphameric character or that it is followed by a valid delimiter.

Severity: 8

IEV151 LITERAL EXPRESSION MODIFIERS MUST BE ABSOLUTE AND PREDEFINED

Explanation: The duplication factor or length modifier in a literal is not (1) a self-defining term or (2) an expression using self-defining terms or previously defined symbols.

System Action: The statement is assembled 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

IEV152 EXTERNAL SYMBOL TOO LONG OR UNACCEPTABLE CHARACTER

Explanation: One of the following errors has occurred:

- An external symbol is longer than 8 characters, or contains a bad character. An external symbol might be the name of a CSECT, START, DXD, AMODE, RMODE, or COM statement, or the operand of an ENTRY, EXTRN, or WXTRN statement or a Q-type or V-type address constant.
- The operand of an ENTRY, EXTRN, or WXTRN statement or a Q-type or V-type address constant is an expression instead of a single term, or contains a bad character.

System Action: The symbol does not appear in the external symbol dictionary. If the error is in the name field, an attempt is made to process the statement as unnamed. If the error is in the operand field, the bad operand is ignored and, if possible, the following operands are processed. A bad constant is assembled as zero.

Programmer Response: Supply a shorter name or replace the expression with a term.

Severity: 12

IEV153 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

IEV154 OPERAND MUST BE ABSOLUTE, PREDEFINED SYMBOLS. SET TO 0

Explanation: The operand on a START or MHELP statement is invalid. 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 and/or predefined symbols.
- The statement is too complex. For example, it may 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. Note that paired relocatable symbols in different LOCTRs, even though in the same CSECT or DSECT, are not valid where an absolute, predefined value is required.

Severity: 8

IEV155 PREVIOUS USE OF SYMBOL IS NOT THIS SECTION TYPE

Explanation: The name on a CSECT, DSECT, COM, or LOCTR statement has been used previously, on a different type of statement. For example, the name on a CSECT has been used before on a statement other than CSECT, such as a machine instruction or a LOCTR.

System Action: This name is ignored, and the statement is processed as unnamed.

Programmer Response: Correct the misspelled name, or change the name to one that does not conflict.

Severity: 12

IEV156 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 alphameric 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 will be processed 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

IEV157 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 may also be that the END operand symbol is not in a CSECT.

System Action: An ORG statement or the operand of an END statement is ignored.

Programmer Response: If an error exists, supply a correct expression. Note that paired relocatable symbols in different LOCTRs, even though in the same CSECT or DSECT, may cause circular definition when used in an ORG statement.

Severity: 12

IEV158 OPERAND 1 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

IEV159 OPERANDS MUST BE ABSOLUTE, PROPER MULTIPLES OF 2 OR 4

Explanation: The combination of operands of a CNOP statement is not one of the following valid combinations:

0,4	2,4
0,8	2,8
4,8	6,8

System Action: The statement is ignored. However, the location counter is adjusted to a halfword boundary.

Programmer Response: Supply a valid combination of CNOP operands.

Severity: 12

IEV161 ONLY ONE TITLE CARD 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

IEV162 PUNCH OPERAND EXCEEDS 80 COLUMNS. IGNORED

Explanation: A PUNCH statement attempted to punch more than 80 characters into a card.

System Action: The statement is ignored. The card is not punched.

Programmer Response: Shorten the operand to 80 characters or fewer or use more than one PUNCH statement.

Severity: 12

IEV163 OPERAND NOT PROPERLY ENCLOSED IN QUOTES

Explanation: The operand of a PUNCH or TITLE statement does not begin with a quotation mark, or the operand of a PUNCH, MNOTE, or TITLE statement does not end with a quotation mark, or the ending quotation mark is not followed by a blank.

System Action: The statement is ignored.

Programmer Response: Supply the missing quotation mark. Be sure that a quotation mark to be punched as data is represented as two quotation marks.

Severity: 8

IEV164 OPERAND IS A NULL STRING - CARD NOT PUNCHED

Explanation: A PUNCH statement does not have any characters between its two single quotation marks, or a single quotation mark to be punched as data is not represented by two single quotation marks.

System Action: The statement is ignored.

Programmer Response: Correct the operand. If you want to "punch" a blank card, the operand of the PUNCH statement should be a blank enclosed in single quotation marks.

Severity: 12

IEV165 UNEXPECTED NAME FIELD

Explanation: The assembler operation has a name and the name field should be blank.

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. Check that the period was not omitted from a sequence symbol.

Severity: 4

IEV166 SEQUENCE SYMBOL TOO LONG

Explanation: A sequence symbol contains more than 62 characters following the period.

System Action: If the sequence symbol is in the name field, the statement is processed without a name. If it is in the operand field of an AIF or AGO statement, the entire statement is ignored.

Programmer Response: Shorten the sequence symbol.

Severity: 12

IEV167 REQUIRED NAME MISSING

Explanation: This statement requires a name and has none. The name field may be blank because an error occurred during an attempt to create the name by substitution or because a sequence symbol was used as the name.

System Action: The statement is ignored.

Programmer Response: Supply a valid name or ensure that a valid name is created by substitution. If a sequence symbol is needed, put it on an ANOP statement ahead of this one and put a name on this statement.

Severity: 8

IEV168 UNDEFINED SEQUENCE SYMBOL

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 appropriate 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. Note that, 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 lookahead.

Severity: 16

IEV170 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 will not be 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

IEV171 STANDARD VALUE TOO LONG

Explanation: The standard (default) value of a keyword parameter on a macro prototype statement is longer than 255 characters.

System Action: The parameter in error and the following parameters are ignored.

Programmer Response: Shorten the standard value.

Severity: 12

IEV172 NEGATIVE DUPLICATION FACTOR. DEFAULT = 1

Explanation: The duplication factor of a SETC statement is negative.

System Action: The duplication factor is given a default value of 1.

Programmer Response: Supply a positive duplication factor.

Severity: 8

IEV173 DELIMITER ERROR, EXPECT BLANK

Explanation: Another character, such as a comma or a quotation mark, is used where a blank (end of operand) is required.

System Action: A machine instruction is assembled as zero. An ORG statement is ignored. For an EQU or END statement, the invalid delimiter is ignored and the operand is processed normally. For a CNOP statement, the location counter is aligned to a halfword boundary.

Programmer Response: Replace the invalid delimiter with a blank. Look for an extra operand or a missing left parenthesis.

Severity: 12

IEV174 DELIMITER ERROR, EXPECT BLANK OR COMMA

Explanation: Another character, such as a quotation mark or ampersand, is used where a blank or a comma is required.

System Action: A machine instruction is assembled as zero. For a USING or DROP statement, the invalid delimiter is ignored and the operand is processed normally.

Programmer Response: Replace the invalid delimiter with a blank or a comma. Look for an extra operand or a missing left parenthesis.

Severity: 12

IEV175 DELIMITER ERROR, EXPECT COMMA

Explanation: Another character, such as a blank or a parenthesis, is used where a comma is required.

System Action: A machine instruction is assembled as zero. For a CNOP statement, the location counter is aligned to a halfword boundary.

Programmer Response: Replace the invalid delimiter with a comma. Be sure each expression is syntactically correct and that no parentheses are omitted.

Severity: 12

IEV176 DELIMITER ERROR, EXPECT COMMA OR LEFT PARENTHESIS

Explanation: Another character, such as a blank or a right parenthesis, is used in a machine instruction where a comma or a left parenthesis is required.

System Action: The machine instruction is assembled as zero.

Programmer Response: Replace the invalid delimiter with a comma or a left parenthesis. Look for invalid syntax or invalid base or length fields on the first operand.

Severity: 12

IEV177 DELIMITER ERROR, EXPECT BLANK OR LEFT PARENTHESIS

Explanation: Another character, such as a comma or a right parenthesis, is used in a machine instruction when a blank or a left parenthesis is required.

System Action: The machine instruction is assembled as zero.

Programmer Response: Replace the invalid delimiter with a blank or a left parenthesis. Look for invalid punctuation or invalid length, index, or base field.

Severity: 12

IEV178 DELIMITER ERROR, EXPECT COMMA OR RIGHT PARENTHESIS

Explanation: Another character, such as a blank or a left parenthesis, is used in a machine instruction when a comma or a right parenthesis is required.

System Action: The machine instruction is assembled as zero.

Programmer Response: Replace the invalid delimiter with a comma or a right parenthesis. Look for a missing base field.

Severity: 12

IEV179 DELIMITER ERROR, EXPECT RIGHT PARENTHESIS

Explanation: Another character, such as a blank or a comma, is used in a machine instruction when a right parenthesis is required.

System Action: The machine instruction is assembled as zero.

Programmer Response: Replace the invalid delimiter with a right parenthesis. Look for an index field used where it is not allowed.

Severity: 12

IEV180 OPERAND MUST BE ABSOLUTE

Explanation: The operand of a SPACE statement or the first, third, or fourth operand of a CCW statement is not an absolute term.

System Action: A SPACE statement is ignored. A CCW statement is assembled as zero.

Programmer Response: Supply an absolute operand. Note that paired relocatable terms may span LOCTRs but must be in the same control section.

Severity: 12

IEV181 CCW OPERAND VALUE IS OUTSIDE ALLOWABLE RANGE

Explanation: One or more operands of a CCW statement are not within the following limits:

- 1st operand—0 to 255
- 2nd operand—0 to 16 777 215 (CCW, CCW0); or 0 to 2 147 483 647 (CCW1)
- 3rd operand—0-255 and a multiple of 8
- 4th operand—0-65,535

System Action: The CCW is assembled as zero.

Programmer Response: Supply valid operands.

Severity: 12

IEV182 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.

System Action: Operand 2 is ignored, and the length attribute of the first operand is used. If the third operand is present, it will be processed normally.

Programmer Response: Correct the error if it exists. Note that paired relocatable symbols in different LOCTRs, even though in the same CSECT, are not valid where an absolute, predefined value is required.

Severity: 8

IEV183 OPERAND 3 MUST BE ABSOLUTE, 0-255. IGNORED

Explanation: If there is another message with this statement, this message is advisory. If this message appears alone, the third operand of an EQU statement contains one of the following errors:

- It is not an absolute term or expression whose value is within the range of 0 to 255.
- It contains a symbol that is not previously defined.
- It is circularly defined.
- It is too complex; for example, it causes an arithmetic overflow during evaluation.

System Action: The third operand is ignored, and the type attribute of the EQU statement is set to U.

Programmer Response: Correct the error if it exists. Note that paired relocatable symbols in different LOCTRs, even though in the same CSECT, are not valid where an absolute, predefined value is required.

Severity: 8

IEV184 COPY DISASTER

Explanation: The assembler copied a library member (executed 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 "swallowed" by an AREAD statement, and the COPY should not have been executed. (Lookahead phase ignores OPSYN statements.) This message will follow the first card of the COPY code.

System Action: The library member will be assembled. If it included an ICTL statement, the format of that ICTL will be used.

Programmer Response: Move COPY statements, or OPSYN statements that modify the meaning of COPY, to a point in the assembly prior to possible entry into lookahead mode.

Severity: 16

IEV185 OPERAND NO. 2 IS ERRONEOUS

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

IEV186 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

IEV187 THE NAME FIELD IS INVALID

Explanation: The name field of an AMODE instruction does not refer to a valid control section in this assembly, or the name field of an RMODE instruction does not refer to a valid control section in this assembly.

System Action: The instruction in error is ignored, and the name field will not appear in the cross-reference listing.

Programmer Response: Specify a valid control section in the name field of the AMODE or RMODE instruction.

Severity: 8

IEV188 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

IEV253 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 no more errors are detected for this statement, the messages and/or annotated text is discarded.

Programmer Response: Correct the indicated errors, and rerun the assembly. If there are more errors on this statement, they will be detected in the next assembly.

Severity: 16

IEV254 * 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 will often cause unsuccessful execution of the program.

Severity: An MNOTE is assigned a severity code of 0 to 255 by the writer of the MNOTE statement.

ABNORMAL ASSEMBLY TERMINATION MESSAGES

Whenever an assembly cannot be completed, Assembler H provides a message and, in some cases, a specially formatted dump for diagnostic information. This may indicate an assembler malfunction or it may indicate a programmer error. The statement causing the error is identified and, if possible, the assembly listing up to the point of the error is printed. The messages in this book give enough information to enable you to correct the error and reassemble your program, or to determine that the error is an assembler malfunction.

MESSAGES

IEV950	END OF STATEMENT FLAG WAS EXPECTED IN MACRO EDITED TEXT, BUT WAS NOT FOUND - MACRO EDITOR IS SUSPECT
IEV951	THE MACRO GENERATOR HAS ENCOUNTERED UNTRANSLATABLE MACRO EDITED TEXT
IEV952	BAD SET SYMBOL NAME FIELD OR LCL/GBL OPERAND - CHECK THE MACRO EDITED TEXT

IEV953	BAD SUBSCRIPT ON SET SYMBOL - CHECK THE MACRO EDITED TEXT
IEV954	CHARACTER EXPRESSION FOLLOWED BY BAD SUBSCRIPTS - CHECK THE MACRO EDITED TEXT
IEV955	A RIGHT PARENTHESIS WITH NO MATCHING LEFT PARENTHESIS WAS FOUND IN AN EXPRESSION - CHECK THE MACRO EDITED TEXT
IEV956	MULTIPLE SUBSCRIPTS OR BAD SET SYMBOL TERMINATOR - CHECK THE MACRO EDITED TEXT
IEV957	BAD TERMINATOR ON CREATED SET SYMBOL - CHECK THE MACRO EDITED TEXT
IEV958	BAD TERMINATOR ON PARAMETER - CHECK THE MACRO EDITED TEXT
IEV959	UNEXPECTED END OF DATA ON H-ASSEMBLER WORK FILE (SYSUT1) - INTERNAL CORE MANAGEMENT IS SUSPECT
IEV960	A BAD INTERNAL FILE NUMBER HAS BEEN PASSED TO THE xxxxx INTERNAL CORE MANAGEMENT ROUTINE
IEV961	AN INVALID CORE REQUEST HAS BEEN MADE, OR THE FREE CORE CHAIN POINTERS HAVE BEEN DESTROYED

Explanation: The assembly is terminated because of one of the errors described in IEV950 through IEV961. This usually is caused by a bug in the assembler itself. Under certain conditions, however, the assembly can be rerun successfully.

System Action: A special abnormal termination dump (Assembler H interrupt and diagnostic dump) follows the message. Depending on where the error occurred, the assembly listing up to the bad statement may also be produced. The dump usually indicates which statement caused termination. It also may 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: Check the statement that caused termination. Correct any errors in it or, especially if the statement is long or complex, rewrite it. Reassemble the program; it may assemble correctly. However, even if it reassembles without error, there may be a bug in the assembler. Save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and contact your IBM level-1 support center. Also, if the program assembles correctly, submit a copy of the listing and the input deck of the correct assembly. This information may be helpful in diagnosing and fixing the assembler bug.

Severity: 20

IEV970 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 (Assembler H 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 may show the last statement generated in the macro. The dump may 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 will not be needed unless the error persists. This information may be helpful in diagnosing and fixing an assembler bug.

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 may be an assembler malfunction. Save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and give them to your IBM program support representative.

Severity: 20

IEV971 INSUFFICIENT CORE AVAILABLE FOR MACRO EDITOR WORK AREA
IEV972 NO AVAILABLE STORAGE REMAINS - ALLOCATE MORE CORE OR BREAK THE INPUT INTO MULTIPLE ASSEMBLIES

Explanation: The assembler work areas are full and none of the contents can be spilled onto the auxiliary data set (SYSUT1). Note that the load modules and fixed data areas of the assembler require about 96K bytes of main storage. The rest of the assembler's region is used for data set buffers, assembler internal files, and work areas. Some of the internal files, like the symbol table, must remain in main storage throughout the assembly.

System Action: A special abnormal termination dump (Assembler H interrupt and diagnostic dump) follows the message. Depending on where the error occurred, the assembly listing up to the bad statement may also be produced. The dump usually indicates the statement being processed when the assembler ran out of main storage. The other information in the dump, such as register and work area contents, is not needed.

Programmer Response: Increase the region size or split the assembly into two or more assemblies. Check for loops in open code that cause the symbol table to overflow. Complete information on these and other remedies, such as decreasing the storage used for data set buffers, is in "Chapter 6. Calculating Storage Requirements" and "Chapter 7. Assembler Language Programming under CMS."

Severity: 20

IEV980 SYSUT1 IS REQUIRED TO BE ASSIGNED TO A DIRECT ACCESS DEVICE, BUT WAS NOT
IEV981 THE DD STATEMENTS FOR SYSIN AND SYSUT1 WERE MISSING OR INVALID
IEV982 THE DD STATEMENT FOR SYSIN WAS MISSING OR INVALID
IEV983 THE DD STATEMENT FOR SYSUT1 WAS MISSING OR INVALID

Explanation: The DD statements for the data sets indicated in IEV980 through IEV983 have not been included in the job control language for the assembly job step or are invalid.

System Action: The assembly is not done because the assembler does not have the required data sets. This message appears alone, without any other abnormal termination dump information.

Programmer Response: Supply a valid DD statement and rerun the assembly. "Chapter 1. Introduction" describes the assembler data sets and the standard DD statements (in the IBM-supplied cataloged procedures) for them. Be sure to check whether your installation has changed the ddname (for example, SYSUT1 to SYSWORK1) or one or more parameters in the cataloged procedure statement.

Severity: 20

IEV998 THE ASSEMBLER COULD NOT RESUME READING A SYSLIB MEMBER BECAUSE IT COULD NOT FIND THE MEMBER AGAIN

Explanation: The assembly is terminated, because the assembler cannot find a COPY member that it has already read. This usually is caused by a bug 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 (Assembler H interrupt and diagnostic dump) follows the message. The dump usually indicates which statement caused termination. It also may 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 may assemble correctly. If it does not reassemble without error, save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and contact your IBM level-1 support center.

Severity: 20

IEV999(I) ASSEMBLY TERMINATED - SYNAD EXIT TAKEN - PERMANENT I/O ERROR ON xxxxx DATA SET

Explanation: The assembly was terminated because of a permanent I/O error on the data set indicated in the message. This is usually caused by a machine or an operating system error. The assembly usually can be rerun successfully. This message will also appear on the console output device.

System Action: A special abnormal termination dump (Assembler H interrupt and diagnostic dump) follows the message. Depending on where the error occurred, the assembly listing up to the bad statement may also be produced. The dump usually indicates which statement caused termination. It also may 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 may have concatenated the input or library data sets incorrectly. Make sure that the DD statement for the data set with the largest block size (BLKSIZE) is placed in the JCL before the DD statements of the data sets concatenated to it. Also, make sure that all input or library data sets have the same device class (all DASD or all tape).

Reassemble the program; it may assemble correctly. If it does not reassemble without error, save the abnormal termination dump, the assembly listing (if one was produced), and the input deck and give them to your IBM customer engineer. Also, if the program assembles correctly, submit a copy of the listing and input deck of the correct assembly.

Severity: 20

Note: The following table is referred to in "Severity Code" under "Message Descriptions" on page 100.

Severity Code	Explanation
0	No errors detected
4	Minor errors detected; successful program execution is probable
8	Errors detected; unsuccessful program execution is possible
12	Serious errors detected; unsuccessful program execution is probable
16	Critical errors detected; normal execution is impossible
20	I/O error from which the system could not recover occurred during assembly, or data sets are missing; assembly terminated

APPENDIX E. ASSEMBLER H VERSION 2 INCOMPATIBILITY WITH OS/VS ASSEMBLER

Assembler H has the following incompatibilities with the OS/VS Assembler:

- TEST option

  The TEST option in the OS/VS Assembler generates entries in the source symbol table for simply relocatable EQUs, named LTORGs, named CNOPs, and named ORGs. Assembler H does not generate source symbol table entries for these assembler instructions.

- COPY

  Assembler H scans a COPY member as a part of "lookahead" processing <u>even though conditional assembly logic (AIF or AGO) may subsequently cause a COPY instruction to be bypassed</u>. This processing occurs regardless of whether or not the COPY member is a macro or source code segment and—for macros—whether or not the macro is defined in a source module or macro library.

  Lookahead is a sequential, statement-by-statement, forward scan over the source text; it is performed by Assembler H but <u>not</u> by the OS/VS Assembler. During lookahead processing, no macro expansion or open-code substitution is performed, and no AIF or AGO branches are taken.

  If the COPY member does not exist in a referenced macro library, Assembler H issues error message IEV060, 'COPY CODE NOT FOUND', <u>even though conditional assembly logic may subsequently cause the COPY instruction to be bypassed</u>. If the COPY member does exist and contains errors, those errors will be diagnosed and the appropriate error messages issued <u>only</u> if the COPY member is actually assembled.

  The OS/VS Assembler executes a COPY assembler instruction and scans the COPY member <u>only</u> if conditional assembly logic causes it to be executed. If the COPY member does not exist and conditional assembly logic causes the COPY instruction to be bypassed, no error message will be issued. The one exception to this rule occurs when a macro is defined within the source module and that macro contains a COPY statement; if the COPY member does not exist in any referenced macro library, the OS/VS Assembler issues message IF0068, 'COPY MEMBER xxxxxxxx NOT FOUND IN LIBRARY'.

GLOSSARY

This glossary has three main types of definitions that apply:

- To the assembler language in particular (usually distinguished by reference to the words "assembler," "assembly," etc.)
- To programming in general
- To data processing as a whole

If you do not understand the meaning of a data processing term used in any of the definitions below, refer to Vocabulary for Data Processing, Telecommunications, and Office Systems, GC20-1699.

IBM is grateful to the American National Standards Institute (ANSI) for permission to reprint its definitions from the American National Standard Vocabulary for Information Processing, which was prepared by Subcommittee X3K5 on Terminology and Glossary of American National Standards Committee X3. ANSI definitions are preceded by an asterisk (*).

addressing mode (24-bit). A System/370 addressing mode of the extended architecture that allows a program to execute using 24-bit addresses. When operating in 24-bit mode, S/370 addressing architecture is applied. Other facilities of the extended architecture (see below) may be utilized. Only the low-order 24 bits of an address are used; the high-order bits are ignored.

addressing mode (31-bit). An extended architecture addressing mode (AMODE) that allows a program to execute using 31-bit addresses and/or other facilities of the extended architecture. 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 perform 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.

bimodal program execution. A function of the extended architecture (see "addressing mode (31-bit)") that allows a program to execute in 24-bit or 31-bit addressing mode. The addressing mode is under program control.

control program. A program that is designed to schedule and supervise the performance of data processing work by a computing 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.

***diagnostic.** Pertaining to the detection and isolation of a malfunction or mistake.

dummy control section (DSECT). A control section that an assembler can use to format an area of storage without producing any object code. 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.

***entry point.** A location in a module to which control can be passed from another module or from the control program.

extended architecture. A hardware architecture for the IBM 3081 processor. A major characteristic is 31-bit addressing. See also "addressing mode (31-bit)."

external symbol dictionary (ESD). Control information associated with an object or load module which identifies the external symbols in the module.

global dictionary. An internal table used by the assembler during macro generation to contain the current values of all unique global SETA, SETB, and SETC variables from all text segments.

global vector table. A table of pointers in the skeleton dictionary of each text segment showing where the global variables are located in the global dictionary.

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 for execution. It combines separately produced object or load modules; resolves symbolic cross references among them; replaces, deletes, and adds control sections; and generates overlay structures on request; and produces executable code (a load module) that is ready to be fetched into main storage and executed.

load module. The output of a single linkage editor execution. A load module is in a format suitable for loading into virtual storage for execution.

loader. A processing program that performs the basic editing functions of the linkage editor, and also fetches and gives control to the processed program, all in one job step. It accepts object modules and load modules created by the linkage editor and generates executable code directly in storage. The loader does not produce load modules for program libraries.

local dictionary. An internal table used by the assembler during macro generation to contain the current values of all local SET symbols. There is one local dictionary for open code, and one for each macro definition.

location counter. A counter whose value indicates the assembled address of a machine instruction or a constant or the address of an area of reserved storage, relative to the beginning of the control section.

***machine instruction.** An instruction that a machine can recognize and execute.

***machine language.** A language that is used directly by the machine.

macro definition. A set of statements that defines the name of, format of, and conditions for generating a sequence of assembler language statements from a single source statement. This statement is a macro instruction that calls the definition. (See also "library macro definition" and "source macro definition.")

macro generation (macro expansion). An operation in which the assembler generates a sequence of assembler language statements from a single macro instruction, under conditions described by a macro definition.

macro instruction (macro call). An assembler language statement that causes the assembler to process a predefined set of statements (called a macro definition). The statements normally produced from the macro definition replace the macro instruction in the source program.

macro library. A library containing macro definitions. The supervisor and data management macro definitions supplied by IBM (GET, LINK, etc.) are contained in the system macro library. Private macro libraries can be concatenated with the system macro library.

main storage. All program addressable storage from which instructions may be executed and from which data can be loaded directly into registers.

object module. The machine-language output of a single execution of an assembler or a compiler. An object module is used as input to the linkage editor or loader.

open code. The portion of a source module that lies outside of and after any source macro definitions that may be specified.

***operating system.** Software which controls the execution of computer programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services.

ordinary symbol attribute reference dictionary. A dictionary used by the assembler. The assembler puts an entry in it for each ordinary symbol encountered in the name field of a statement. The entry contains the attributes (type, length, etc.) of the symbol.

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 perform a specific function.

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.

***relocation dictionary.** The part of an object or load module that identifies all addresses that must be adjusted when a relocation occurs.

residence mode. An extended architecture addressing mode (RMODE) that allows a program to specify the residence mode (below 16 megabytes or anywhere) to be associated with a control section.

return code. A value placed in the return code register at the completion of a program. The value is established by the user and may be used to influence the execution of succeeding programs or, in the case of an abnormal end of task, may simply be printed for programmer analysis.

severity code. A code assigned by the assembler to each error detected in the source code. The highest code encountered during assembly becomes the return code of the assembly step.

skeleton dictionary. A dictionary built by the assembler for each text segment. It contains the global vector, 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 symbols of a programming language.

***statement.** A meaningful expression or generalized instruction in a source language.

symbol file. A data set used by the assembler for symbol definitions and references and literals.

symbolic parameter. In assembler programming, a variable symbol declared in the prototype statement of a macro definition.

system macro definition. Loosely, an IBM-supplied library macro definition which provides access to operating system facilities.

text segment. The range over which a local dictionary has meaning. The source module is divided into text segments with a segment for open code and one for each macro definition.

***translate.** To transform statements from one language into another without significantly changing the meaning.

virtual storage. Address space appearing to the user as real storage from which instructions and data are mapped into real storage locations. The size of virtual storage is limited by the addressing scheme of the computing system and by the amount of auxiliary storage available, rather than by the actual number of real storage locations.

INDEX

Reader's Comment Form

Assembler H Version 2
Application Programming: Guide
SC26-4036-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ________________

Previous TNL ________________

Fold on two lines, tape, and mail. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Thank you for your cooperation.

Please use pressure sensitive or other gummed tape to seal this form.

SC26-4036-0

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

SC26-4036-0

Assembler H Version 2 Application Programming: Guide (File No. S370-21) Printed in U.S.A. SC26-4036-0

SC26-4036-0