GC33-4021-1
File No. S370-21 (OS/VS)

Systems

OS/VS Assembler Programmer's Guide

VS1 Release 2

VS2 Release 1

IBM

Second Edition (May, 1973)

This is a reprint of GC33-4021-0 incorporating changes released in the following Technical Newsletters:

GN33-8146(dated July 25, 1972)
GN33-8150(dated September 29, 1972)
GN33-8159(dated December 1, 1972)

This edition applies to release 2 of OS/VS1, release 1 of OS/VS2 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the IBM System/360 and System/370 Bibliography, Order No. GA22-6822, and the IBM System/370 Advanced Function Bibliography, Order No. GC20-1763, for the editions that are applicable and current.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader,s comments. If the form has been removed, comments may be addressed to IBM Nordic Laboratory, Programming Publications, Box 962, S-181 09 Lidingö, Sweden. Comments become the property of IBM.

© Copyright International Business Machines Corporation 1972, 1973

This Manual and Who It Is For

This manual is for programmers who code in the assembler language. It is intended to help you assemble, link edit, and execute your program; to choose and specify the assembler options you need; and to interpret the listing and the diagnostic messages issued by the assembler. This manual also serves as a guide to information contained in other publications which is of importance to you as an assembler-language programmer. To use this manual you should have a basic understanding of the operating system as described in Introduction to OS, Order No. GC28-6534. You should also have a good understanding of the assembler language as described in OS/VS and DOS/VS Assembler Language, Order No. GC33-4010.

Other Manuals You Will Need

In addition to OS/VS and DOS/VS Assembler Language, you should have the following publications available when using this manual:

System/370 Principles of Operation, Order No. GA22-7000
OS/VS JCL Reference, Order No. GC28-0618
OS/VS Linkage Editor and Loader, Order No. GC26-3803

How This Manual Is Organized

This manual has five main sections and seven appendixes:

Introduction describes the purpose of the VS assembler, its relationship to the operating system, and its input and output. It also describes how the operating system processes your program and reviews the concepts of job, job step, job control language, and cataloged procedures.

Job Control Statements for Assembler Jobs shows you how to invoke the assembler for simple jobs (using cataloged procedures); describes the assembler options and how to specify them; lists the job control statements that make up the four assembler cataloged procedures; and gives examples of how to use the cataloged procedures for more complex jobs.

The Assembler Listing tells you how to interpret the printed listing produced by the assembler.

Programming Considerations serves as a guide to information contained in other programming manuals which you will find useful as an assembler-language programmer. Among the topics discussed are:

- Designing your program
- Specifying the entry point
- Linking with modules written in other languages
- Linking with processing programs

Adding Macro Definitions to a Library tells you how to catalog macro definitions in the system macro library or in a private library.

Appendix A gives definitions of terms used in this manual.
Appendix B gives the listing of the assembler sample program.
Appendix C shows the detailed format of the object deck.
Appendix D tells you how to invoke the assembler dynamically from a problem program.
Appendix E describes the data sets used by the assembler and the assembler's storage requirements.
Appendix F describes the SYSTERM listing.
Appendix G explains the diagnostic messages issued by the assembler.

Contents

INTRODUCTION 7
Purpose of the Assembler 7
Relationship of the Assembler to the Operating System 7
Input 7
Output 7
Compatibility 8
How the Operating System Handles Your Program 8
  Assembler 8
  Linkage Editor 8
  Execution of Your Program 8
  Loader 8
Job Control Language and Cataloged Procedures 10
  Jobs and Job Steps 10
  Job Control Language 11

JOB CONTROL STATEMENTS FOR ASSEMBLER JOBS 13
Simple Assembly and Execution 13
  Assembly 13
  Assembly and Execution 14
Assembler Options 15
  What Assembler Options Are 15
  How to Specify Assembler Options 16
The Assembler Cataloged Procedures 23
  Assembly (ASMFC) 24
  Assembly and Link Editing (ASMFCL) 26
  Assembly, Link Editing, and Execution (ASMFCLG) 28
  Assembly and Loader-Execution (ASMFCG) 30
  Examples 32

THE ASSEMBLER LISTING 34
External Symbol Dictionary (ESD) 36
The Source and Machine Language Statements 38
  Source Statement Fields 38
Relocation Dictionary (RLD) 41
Symbol Cross Reference 42
Literal Cross Reference 44
Diagnostics and Statistics 45

PROGRAMMING CONSIDERATIONS 47
Designing Your Program 47
Specifying the Entry Point into Your Program 48
Linking with Modules Produced by other Language Translators 48
Linking with IBM-Supplied Processing Programs 49

ADDING MACRO DEFINITIONS TO A LIBRARY 50

APPENDIX A. GLOSSARY 51

APPENDIX B. ASSEMBLER SAMPLE PROGRAM 57

APPENDIX C. OBJECT DECK OUTPUT 65
  ESD Card Format 65
  TXT Card Format 66
  RLD Card Format 66
  END Card Format 67
  SYM Card Format 67

APPENDIX D. DYNAMIC INVOCATION OF THE ASSEMBLER 70

APPENDIX E. ASSEMBLER DATA SETS AND STORAGE REQUIREMENTS 72
Assembler Data Sets 72
Assembler Virtual Storage Requirements 73

APPENDIX F. THE SYSTERM LISTING 75

APPENDIX G. ASSEMBLER DIAGNOSTIC ERROR MESSAGES 78
How to Use This Section 78
Recurring Errors 80

INDEX 143

Figures

Figure 1. How the operating system handles your program 9
Figure 2. Jobs and job steps 10
Figure 3. The cataloged procedure concept 11
Figure 4. The assembler options (Part 1 of 5) 18
Figure 5. Cataloged procedure for assembly (ASMFC) 25
Figure 6. Cataloged procedure for assembly and link editing (ASMFCL) 27
Figure 7. Cataloged procedure for assembly, link editing, and execution (ASMFCLG) 29
Figure 8. Cataloged procedure for assembly and loader-execution (ASMFCG) 31
Figure 9. Assembler listing 35
Figure 10. External symbol dictionary 37
Figure 11. Source and machine language statements 40
Figure 12. Relocation dictionary 41
Figure 13. Symbol cross reference 43
Figure 14. Literal cross reference 44
Figure 15. Diagnostics and statistics 46
Figure 16. Minimum requirements for a simple program 48
Figure 17. Assembler sample program (Part 1 of 11) 57
Figure 18. SYM card format 69
Figure 19. Assembler data set characteristics 74
Figure 20. SYSPRINT listing of the source statements used to snow SYSTERM output 76
Figure 21. SYSTERM listing produced for the source statements shown in Figure 20 77

This section describes the purpose of the VS Assembler, its relationship to the operating system, and its input and output. It also tells you how the operating system processes your assembler language program and reviews the concepts of job, job step, job control language, and cataloged procedure.

Purpose of the Assembler

The purpose of the VS Assembler is to translate programs written in the assembler language into object modules, that is, code suitable as input to the linkage editor or loader.

Relationship of the Assembler to the Operating System

The VS Assembler is supplied with the OS/VS control program package. In the same way as the linkage editor or loader, it is executed under control of the OS control program. For a complete description of the relationship between a processing program and the various components of the control program, refer to Introduction to OS.

Input

As input the assembler accepts a program written in the Assembler language as defined in Assembler Language. 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 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 loader. The format of the object module is described in Appendix C.

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 the section "The Assembler Listing".

Compatibility

The language supported by the VS Assembler is compatible with the language supported by the OS Assembler F. All programs which assemble error-free under Assembler F will also assemble error-free under the VS Assembler. However, the resulting object code may in odd cases be different because of the extended features of the language supported by the VS Assembler (the extended attribute reference and SETC facilities).

How the Operating System Handles Your Program

Once you have coded and punched your program, it must be processed by the assembler and the linkage editor or loader before it can be executed. (See Figure 1.)

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 modules 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 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 linkage editing. However, if you need to change your program, you must assemble and linkage 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 Linkage Editor and Loader.

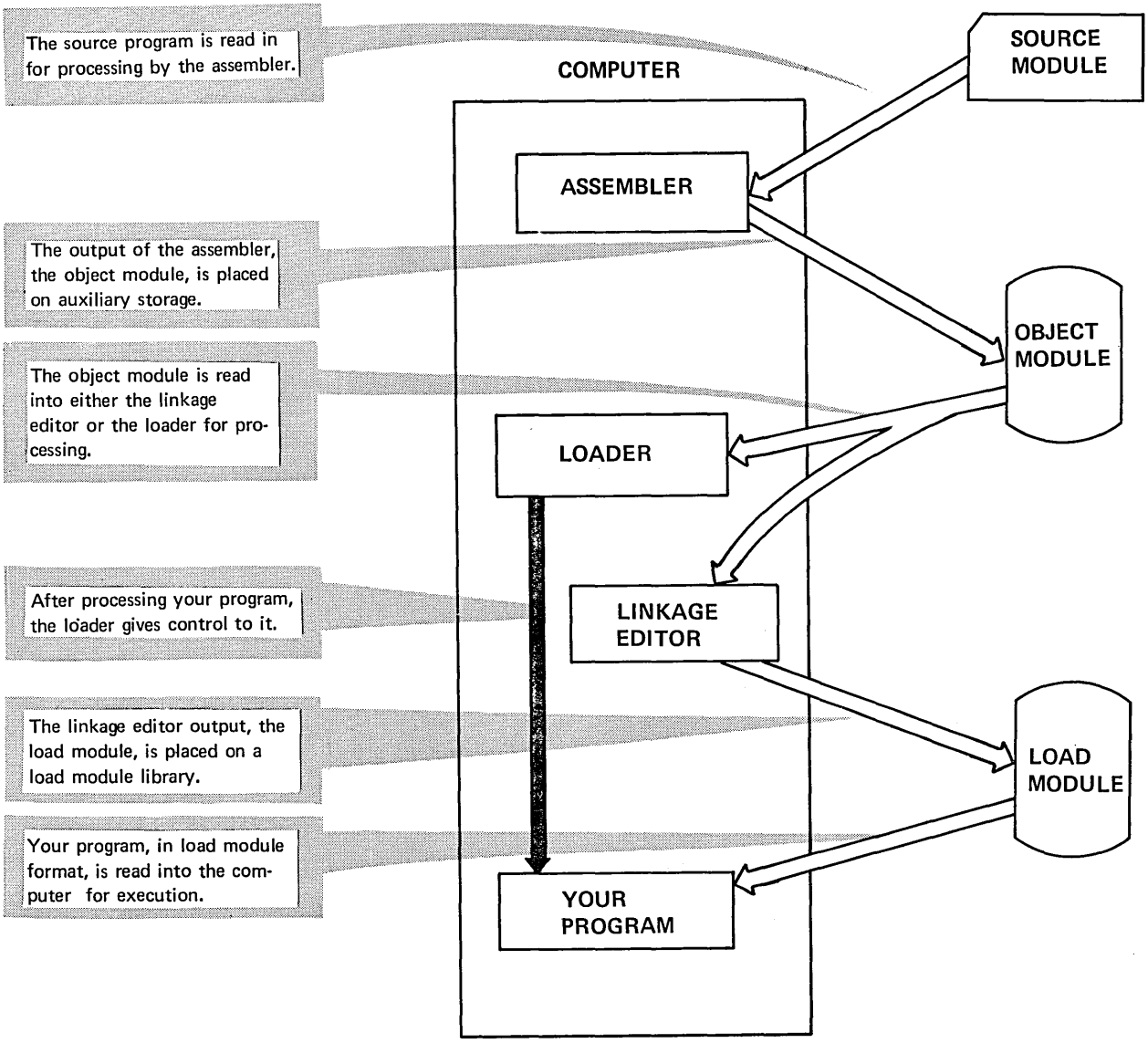


Figure 1. How the operating system handles your program.

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 VS 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 illustrated in Figure 2.

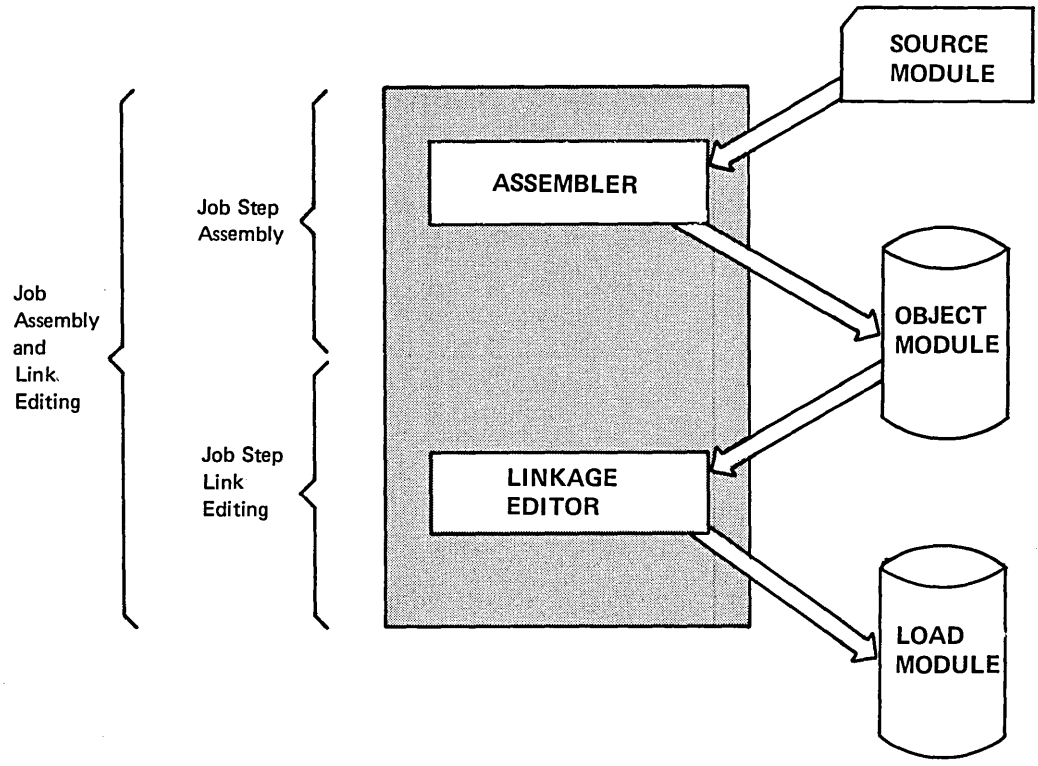


Figure 2. Jobs and job steps

JOB CONTROL LANGUAGE

The job control language is your way of communicating to the operating system control program what services you want performed and what auxiliary devices you want used. Job control language (JCL) statements are usually punched into cards and supplied in the job stream together with your source module and other data needed by the job.

For a detailed discussion of job control language statements, see 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 3 illustrates the concept of a cataloged procedure.

There are several cataloged procedures available for assembler jobs. They are described in the section "Job Control Statements for Assembler Jobs".

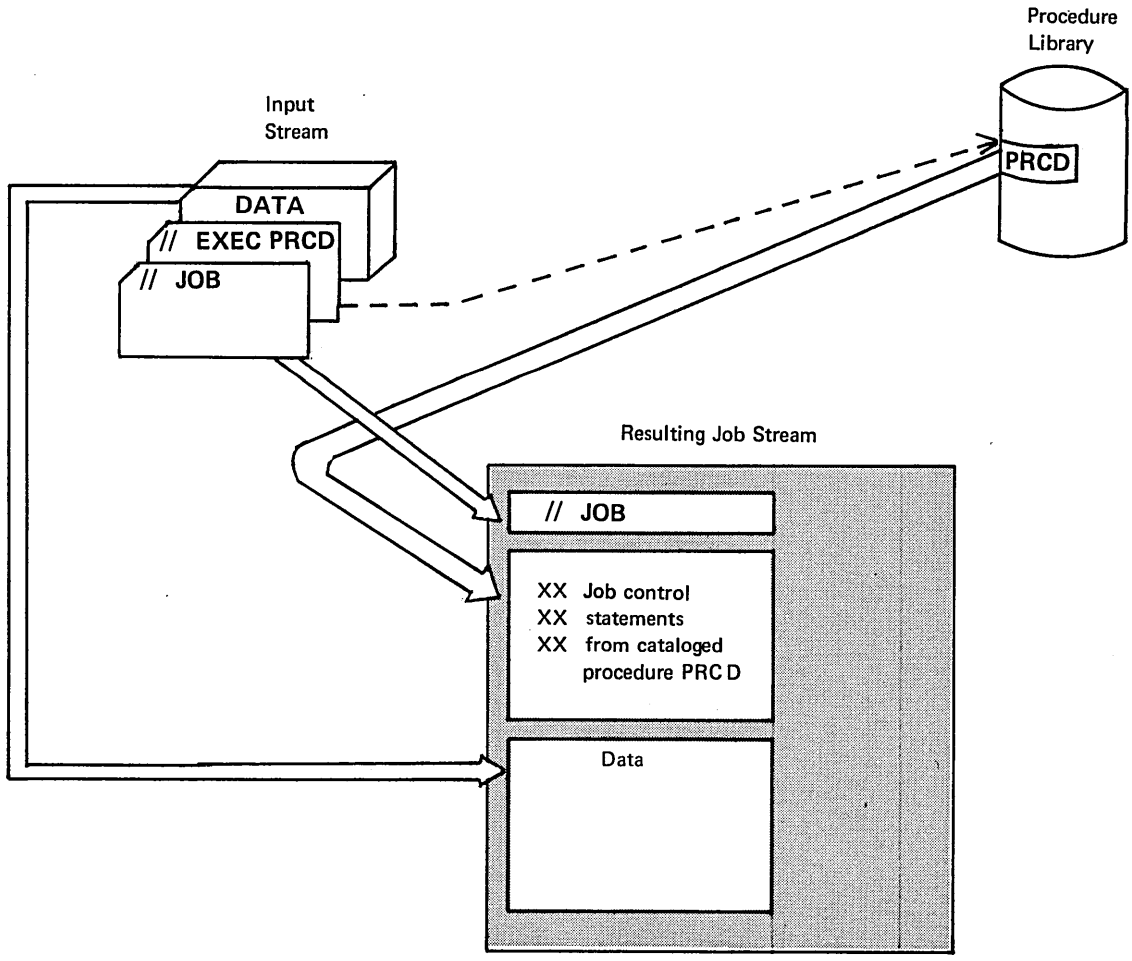


Figure 3. The cataloged procedure concept

The purpose of this section is to:

- Show you how to invoke the assembler for simple jobs (using cataloged procedures).
- Describe the assembler options and how to request them.
- List the job control statements that make up the four assembler cataloged procedures.
- Give examples of how to use the cataloged procedures for more complex jobs.

Simple Assembly and Execution

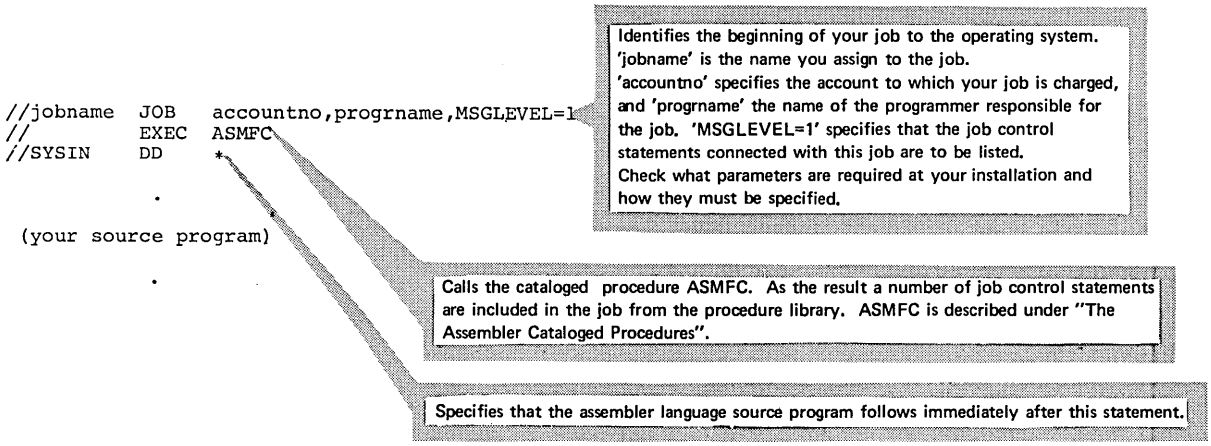
This section gives you the minimum JCL statements needed for two simple assembler jobs:

- 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 job control language (JCL) statements:



These statements cause the assembler to assemble your program and to produce a listing (described in the section "The Assembler Listing") and an object module punched on cards (described in Appendix C).

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  ASMFC,PARM=NODECK
//SYSIN   DD    *
          .
   (your source program)
          .
```

The second parameter (PARM) specifies the assembler option NODECK, telling the assembler not to produce any punched object module. For a full discussion of the assembler options, see "Assembler Options".

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  ASMFCG
//ASM.SYSIN DD  *
          .
    (your source program)
          .

//GO.SYSIN DD   *
          .
    (data, if any, for your program)
          .
```

Calls the procedure ASMFCG, containing job control statements for execution of the assembler (in procedure step ASM) and the loader in (in step GO).

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

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

The first step of the ASMFCG 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  ASMFCG,PARM.ASM=(OBJ,NODECK)
//ASM.SYSIN    DD    *
                .
     (your source program)
                .
//GO.SYSIN     DD    *
                .
     (data for your program)
                .
```

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

Assembler Options

WHAT ASSEMBLER OPTIONS ARE

Assembler options are functions of the assembler that you, as an assembler language programmer, can select. For example, you can use assembler options to specify whether or not you want the assembler to produce an object deck; whether or not you want it to print certain items in the listing; and whether or not you want it to check your program for reenterability.

The assembler options can be divided into four categories:

- Listing control options, which determine the information to be included in the program listing.
- Output control options, which specify the device on which the assembler object module is to be written and the contents of the module.
- SYSTERM options, which determine the information to be included in the listing produced on the SYSTERM data set. This data set is primarily for use by the Time Sharing Option (TSO) of VS2.
- Other assembler options, which specify miscellaneous functions and values for the assembler.

Figure 4 lists all the assembler options. The underlined values are the standard or default values. These values are used by the assembler for options that you do not specify.

As you can see from the figure, the options fall into two format types:

- Simple pairs of keywords: a positive form (for example, DECK) that requests a function, and an alternative negative form (for example, NODECK) that rejects the function.
- Keywords that permit you to assign a value to a function (for example, LINECOUNT(40)).

HOW TO SPECIFY ASSEMBLER OPTIONS

You use the PARM field of the EXEC JCL statement calling the assembler to specify the assembler options. Code PARM= followed by a list of options that you have selected. For example,

```
//STEPA EXEC PGM=IFOX00,PARM='NODECK,FLAG(5),NORLD'
```

IFOX00 is the name of the assembler; three options are specified for the execution of it. Default values are used for the other options.

When you use cataloged procedures, you will notice that most of them contain an option specification in the EXEC statement for the assembly. To override such a specification, include a PARM field with your options in the EXEC statement calling the procedure. If the cataloged procedure contains more than one step, you must add the procedure step name as a qualifier to the PARM operand. For example,

```
//STEP1 EXEC ASMFCG,PARM.ASM='OBJ,NODECK'
```

The .ASM is necessary to indicate the assembly step. As you can see in the section "The Assembler Cataloged Procedures", the stepname for assembly is always ASM. You must also remember that when you override the PARM field in a procedure, the entire PARM field is overridden. The PARM field specification in the cataloged procedure ASMFCG is PARM=OBJ, and the OBJ option must be repeated when you override the PARM field. Otherwise the assembler default value NOOBJ will be used. (For a more detailed description of overriding operands on EXEC statements in cataloged procedures, refer to JCL Reference.

The PARM field is coded according to the following rules:

- Single quotes or parentheses must surround the entire PARM value if you specify two or more options.
- The options must be separated by commas. You may specify as many options as you wish, and in any order. However, the length of the option list must not exceed 100 characters, including separating commas.
- The BUFSIZE, FLAG, LINECOUNT, or SYSPARM options must appear within single quotes.
- 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 quotes must not be continued on another card.

The following examples illustrate these rules:

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

Listing Control Options	
ALOGIC	Conditional assembly statements processed in open code are listed.
NOALOGIC	The ALOGIC option is suppressed.
ESD	The external symbol dictionary (ESD) is listed. (Refer to "The Assembler Listing" for further information on the ESD.)
NOESD	No ESD listing is printed.
FLAG{(nnn) / (0)}	Diagnostic messages and MNOTE messages below severity code nnn will not appear in the listing. Diagnostic messages can have severity codes of 4, 8, 12, 16, or 20 (20 is the most severe), and MNOTE severity codes can be between 0 and 255. For example, FLAG(8) suppresses diagnostic messages with a severity code of 4 and MNOTE messages with severity codes of 0 through 7.
LINECOUNT{(nn) / (55)}	nn specifies the number of lines to be listed per page.
LIST	An assembler listing is produced.
NOLIST	No assembler listing is produced. This option overrides ESD, RLD, and XREF.
MCALL	Inner macro instructions encountered during macro generation are listed following their respective outer macro instructions. The assembler assigns statement numbers to these instructions. The MCALL option is implied by the MLOGIC option; NOMCALL has no effect if MLOGIC is specified.
NOMCALL	The MCALL option is suppressed.
MLOGIC	All statements of a macro definition processed during macro generation are listed after the macro instruction. The assembler assigns statement numbers to them.
NOMLOGIC	The MLOGIC option is suppressed.

Figure 4. The assembler options
(Part 1 of 5)

Listing Control Options (continued)	
RLD	The assembler produces the relocation dictionary as part of the listing. (Refer to "The Assembler Listing" for further information on the relocation dictionary.)
NORLD	The RLD is not printed.
LIBMAC	The macro definitions read from the macro libraries and any assembler statements following the logical END statement are listed after the logical END statement. The logical END statement is the first END statement processed during macro generation. It may appear in a macro or in open code; it may even be created by substitution. The assembler assigns statement numbers to the statements that follow the logical END statement.
NOLIBMAC	The LIBMAC option is suppressed.
XREF(FULL)	The assembler listing will contain a cross reference table of all symbols used in the assembly. This includes symbols that are defined but never referenced. The assembler listing will also contain a cross reference table of literals used in the assembly.
XREF(SHORT)	The assembler listing will contain a cross reference table of all symbols that are referenced in the assembly. Any symbols defined but not referenced are not included in the table. The assembler listing will also contain a cross reference table of literals used in the assembly.
NOXREF	No cross reference tables are printed.

Figure 4. The assembler options
(Part 2 of 5)

Output Control Options	
DECK	The object module is written on the device specified in the SYSPUNCH DD statement. If this option is specified together with the OBJECT option, the object module will be written both on SYSPUNCH and on SYSGO.
NODECK	The DECK option is suppressed.
OBJECT or OBJ	The object module is written on the device specified in the SYSGO DD statement. If this option is specified together with the DECK option, the object module will be written both on SYSGO and on SYSPUNCH.
NOOBJECT or NOOBJ	The OBJECT option is suppressed.
TEST	The special source symbol table (SYM cards) is included in the object module. (See Appendix C for details.)
NOTEST	No SYM cards are produced.

Figure 4. The assembler options
(Part 3 of 5)

SYSTERM Options	
NUMBER or NUM	The line number field (columns 73-80 of the input cards) is written in the SYSTERM listing for statements for which diagnostic information is given. This option is valid only if TERMINAL is specified.
NONUMBER or NONUM	The NUMBER option is suppressed.
STMT	The statement number assigned by the assembler is written in the SYSTERM listing for statements for which diagnostic information is given. This option is valid only if TERMINAL is specified.
NOSTMT	The STMT option is suppressed.
TERMINAL or TERM	The assembler writes diagnostic information on the SYSTERM data set. The diagnostic information, described in detail in Appendix F, consists of the diagnosed statement followed by the error message issued.
NOTERMINAL or NOTERM	The TERMINAL option is suppressed.

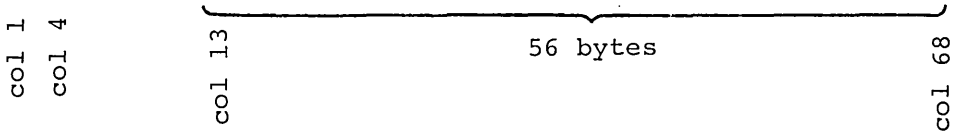
Figure 4. The assembler options.
(Part 4 of 5)

Other Assembler Options	
ALIGN	All data is aligned on the proper boundary in the object module; for example, an F-type constant is aligned on a fullword boundary. In addition, the assembler checks storage addresses used in machine instructions for alignment violations.
NOALIGN	The assembler does not align data areas other than those specified in CCW instructions. The assembler does not skip bytes to align constants on proper boundaries. Alignment violations in machine instructions are not diagnosed.
BUFSIZE(MIN)	The assembler uses the minimum buffer size (790 bytes) for each of the utility data sets (SYSUT1, SYSUT2, and SYSUT3). Storage normally used for buffers is allocated to work space. Because more work space is available, more complex programs can be assembled in a given region; but the speed of the assembly is substantially reduced.
BUFSIZE (STD)	The buffer size that gives optimum performance is chosen. The buffer size depends on the size of the region or partition. Of the assembler working storage in excess of minimum requirements, 37% is allocated to the utility data set buffers, and the rest to macro generation dictionaries. Refer to Appendix E for a more complete description of the effects of BUFSIZE.
RENT	The assembler checks your program for a possible violation of program reentrability. Code that makes your program non-reentrant is identified by an error message.
NORENT	The RENT option is suppressed.
SYSPARM{(string) / () (null string)}	'string' is the value assigned to the system variable symbol &SYSPARM (explained in Assembler Language). Due to JCL restrictions, you cannot specify a SYSPARM value longer than 56 characters (as explained in Note 1 following this figure). Two quotes are needed to represent a single quote, and two ampersands to represent a single ampersand. For example, PARM='OBJECT,SYSPARM((&&AM,''BO).FY)' assigns the following value to &SYSPARM: (&AM,'BO).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.

Figure 4. The assembler options. (Part 5 of 5)

Note 1: The restrictions imposed upon the PARM field limit the maximum length of the SYSPARM value to 56 characters. Consider the following example:

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



Since SYSPARM uses parentheses, it must be surrounded by quotes. Thus, it cannot be continued onto a continuation card. The leftmost column that can be used is column 4 on a continue card. A quote and the keyword must appear on that line as well as the closing quotes. 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).

Note 2: Even though the formats of some of the options previously supported by OS Assembler F have been changed, you can use the old formats for the following options: ALGN (now ALIGN), NOALGN (NOALIGN), LINECNT=nn (LINECOUNT(nn)), LOAD (OBJECT), and NOLOAD (NOOBJECT). This support will, however, be continued only for a limited number of VS releases, so you should change to the new options as soon as possible. The Assembler F option DOS is not supported by the VS Assembler.

The Assembler Cataloged Procedures

This section describes the four assembler cataloged procedures and tells you how to use them. They are:

- ASMFC (assembly)
- ASMFCL (assembly and linkage editing)
- ASMFCG (assembly and loader-execution)
- ASMFCLG (assembly, linkage editing, and execution)

The procedure you choose on each occasion will depend on the type of job you want to run. First, you may want to run an assembly to correct your coding and keypunching errors. For this, you would use the ASMFC procedure with the option NODECK specified. In the next run you may want to assemble and execute your program, in which case you can use ASMFCG (or possibly ASMFCLG, if you use linkage editor features not supported by the loader). When you have debugged your program, you may want to include it in a load module library using ASMFCL.

The examples given in this section assume that the cataloged procedures you are using are identical to the cataloged procedures delivered by IBM. Therefore, you should first make sure that your installation has not modified the procedures after they were delivered.

The ASMFC procedure contains only one job step: assembly. You use the name ASFMC to call this procedure. The result of execution is an object module, in punched card form, and an assembler listing.

To call the procedure use the following statements:

```
//jobname    JOB       parameters
//stepname   EXEC    {ASMFC      }
                     {PROC=ASMFC }
//SYSIN      DD        *
             .
             .
    source module
             .
```

The statements of the ASMFC procedure shown in Figure 5 are read from the procedure library and merged into your input stream. The SYSIN statement specifies that the input to the assembler (that is, your source program) follows immediately after the statement.

```
[1] //ASMFC     PROC  MAC='SYS1.MACLIB',MAC1='SYS1.MACLIB'
[2] //ASM      EXEC  PGM=IFOX00,REGION=128K
[3] //SYSLIB   DD    DSN=&MAC,DISP=SHR
    //         DD    DSN=&MAC1,DISP=SHR
[4] //SYSUT1   DD    DSN=&&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(600,100)),
    //             SEP=(SYSLIB)
    //SYSUT2   DD    DSN=&&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(300,50)),
    //             SEP=(SYSLIB,SYSUT1)
    //SYSUT3   DD    DSN=&&SYSUT3,UNIT=SYSSQ,SPACE=(1700,(300,50))
[5] //SYSPRINT DD    SYSOUT=A,DCB=BLKSIZE=1089
[6] //SYSPUNCH DD    SYSOUT=B
```

[1] This statement names the procedure and gives default values to the symbolic parameters MAC and MAC1.

[2] This statement specifies that the program to be executed is IFOX00, which is the name of the assembler.
The REGION parameter specifies the virtual storage region that gives best performance. It is possible to run the assembler in 64K, in which case you must change the region size parameter.
You can also add COND and PARM parameters.

[3] This statement identifies the macro library data set. The succeeding statement concatenates another macro library with it. The default values for the DSN parameters of both data sets are SYS1.MACLIB, the system macro library. You can change either or both of the data sets in the EXEC statement calling the procedure. For example, to concatenate your own macro library with SYS1.MACLIB, code your EXEC statement as follows:

```
//   EXEC     ASMFC,MAC1=MYMACS
```

DISP=SHR indicates that the data set can be used simultaneously by other jobs in the system.

[4] SYSUT1, SYSUT2, and SYSUT3 specify the assembler work data sets. The device classname SYSSQ represents either a direct access device or a tape drive. The I/O units assigned to the classnames are specified by your installation during system generation. Instead of a classname you can specify a unit name, such as 2314. The DSN parameters guarantee dedicated work data sets, if this is supported by your installation. The SEP and SPACE parameters are effective only if SYSSQ is a direct access device. The space required depends on the source program.

[5] This statement defines the standard system output class as the destination of the assembler listing. You can specify any blocksize that is a multiple of 121.

[6] This statement describes the data set that will receive the punched object module.

Figure 5. Cataloged prccedure for assembly (ASMFC)

ASSEMBLY AND LINK EDITING (ASMFCL)

The ASMFCL procedure consists of two job steps: assembly and link editing. It produces an assembler listing, a linkage editor listing, and a load module.

SYSGO contains the output from the assembly step and the input to the linkage editor step. It can be concatenated with additional input to the linkage editor. This additional input can be linkage editor control statements or other object modules.

To call the procedure, use the following statements:

```
//jobname     JOB
//stepname    EXEC       ASMFCL
//ASM.SYSIN   DD         *
              .
              .
    source program statements
              .
              .
/*
//LKED.SYSIN DD          *
              .
              .
    object module or      }   necessary only if linkage
    linkage editor        }   editor is to combine modules
    control statements    }   or read linkage editor control
              .           }   information from the job stream
              .           }
/*
```

Figure 6 shows the statements that make up the ASMFCL procedure. Only those statements not previously discussed are explained.

```
  //ASMFCL   PROC MAC='SYS1.MACLIB',MAC1='SYS1.MACLIB'
  //ASM      EXEC PGM=IFOX00,PARM=OBJ,REGION=128K
  //SYSLIB   DD   DSN=&MAC,DISP=SHR
  //         DD   DSN=&MAC1,DISP=SHR
  //SYSUT1   DD   DSN=&&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(600,100)),
  //              SEP=(SYSLIB)
  //SYSUT2   DD   DSN=&&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(300,50)),
  //              SEP=(SYSLIB,SYSUT1)
  //SYSUT3   DD   DSN=&&SYSUT3,UNIT=SYSSQ,SPACE=(1700,(300,50))
  //SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=1089
  //SYSPUNCH DD   SYSOUT=B
[1]//SYSGO    DD   DSN=&&OBJSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
  //              DISP=(MOD,PASS)
[2]//LKED     EXEC PGM=IEWL,PARM=(XREF,LET,LIST,NCAL),REGION=128K,
  //              COND=(8,LT,ASM)
[3]//SYSLIN   DD   DSN=&&OBJSET,DISP=(OLD,DELETE)
[4]//         DD   DDNAME=SYSIN
[5]//SYSLMOD  DD   DSN=&&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
  //              DISP=(MOD,PASS)
[6]//SYSUT1   DD   DSN=&&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
  //              SPACE=(1024,(50,20))
[7]//SYSPRINT DD   SYSOUT=A
```

[1] The SYSGO DD statement describes a temporary data set--the object module--which is to be 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, 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 produced as output by the assembler.

[4] This statement is used to concatenate any input to the linkage editor from the input stream 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. If the output of the linkage editor is to be retained, the DSN parameter must specify a library name and member name designating where 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.

Figure 6. Cataloged procedure for assembly and link editing (ASMFCL)

ASSEMBLY, LINK EDITING AND EXECUTION (ASMFCLG)

The ASMFCLG consists of three job steps: assembly, link editing and execution. 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       ASMFCLG
//ASM.SYSIN   DD         *
              .
              .
    source program statements
              .
              .
/*
//LKED.SYSIN  DD         *  ┐
              .            │
              .            │
    object module or       │  necessary only if linkage
    linkage editor         │  editor is to combine modules
    control statements     │  or read linkage editor control
              .            │  information from the job stream
              .            ┘
/*
//GO.ddname   DD         parameters ┐
//GO.ddname   DD         parameters │
//GO.ddname   DD         *          │ only
              .                    │ if
              .                    │ necessary
    problem program input          │
              .                    │
              .                    ┘
/*
```

Figure 7 shows the statements that make up the ASMFCLG procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMFCLG  PROC MAC='SYS1.MACLIB',MAC1='SYS1.MACLIB'
//ASM      EXEC PGM=IFOX00,PARM=OBJ,REGION=128K
//SYSLIB   DD   DSN=&MAC,DISP=SHR
//         DD   DSN=&MAC1,DISP=SHR
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(600,100)),
//              SEP=(SYSLIB)
//SYSUT2   DD   DSN=&&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(300,50)),
//              SEP=(SYSLIB,SYSUT1)
//SYSUT3   DD   DSN=&&SYSUT3,UNIT=SYSSQ,SPACE=(1700,(300,50))
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=1089
//SYSPUNCH DD   SYSOUT=B
//SYSGO    DD   DSN=&&OBJSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
//              DISP=(MOD,PASS)
[1] //LKED    EXEC PGM=IEWL,PARM=(XREF,LET,LIST,NCAL),REGION=128K,
//              COND=(8,LT,ASM)
//SYSLIN   DD   DSN=&&OBJSET,DISP=(OLD,DELETE)
//         DD   DDNAME=SYSIN
[2] //SYSLMOD  DD   DSN=&&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//              DISP=(MOD,PASS)
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//              SPACE=(1024,(50,20))
//SYSPRINT DD   SYSOUT=A
[3] //GO       EXEC PGM=*.LKED.SYSLMOD,COND=((8,LT,ASM),(4,LT,LKED))
```

[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 succeeding job step.

[3] This statement initiates execution of the assembled and linkage edited program. The notation *.LKED.SYSLMOD identifies the program to be executed as being in the data set described in job step LKED by the DD statement named SYSLMOD.

Figure 7. Cataloged procedure for assembly, link editing, and execution (ASMFCLG)

ASSEMBLY AND LOADER-EXECUTION (ASMFCG)

The ASMFCG procedure contains two job steps: assembly and loader-execution. The loader link-edits, loads, and passes control to the program for execution.

Both assembler and a loader listing are produced, but the load module is not included in a library.

To call the procedure use the following statements:

```
//jobname    JOB
//stepname   EXEC      ASMFCG
//ASM.SYSIN  DD        *
             .
             .
    source program
             .
             .
/*                                    ┐
//GC.ddname  DD        parameters      │
//GO.ddname  DD        parameters      │
//GC.ddname  DD        *               │ only
             .                        │ if
             .                        │ necessary
    problem program input             │
             .                        │
             .                        ┘
/*
```

Figure 8 shows the statements that make up the ASMFCG procedure. Only those statements not previously discussed are explained in the figure.

```
//ASMFCG   PROC MAC='SYS1.MACLIB',MAC1='SYS1.MACLIB'
//ASM      EXEC PGM=IFOX00,PARM=OBJ,REGION=128K
//SYSLIB   DD   DSN=&MAC,DISP=SHR
//         DD   DSN=&MAC1,DISP=SHR
//SYSUT1   DD   DSN=&&SYSUT1,UNIT=SYSSQ,SPACE=(1700,(600,100)),
//              SEP=(SYSLIB)
//SYSUT2   DD   DSN=&&SYSUT2,UNIT=SYSSQ,SPACE=(1700,(300,50)),
//              SEP=(SYSLIB,SYSUT1)
//SYSUT3   DD   DSN=&&SYSUT3,UNIT=SYSSQ,SPACE=(1700,(300,50))
//SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=1089
//SYSPUNCH DD   SYSOUT=B
//SYSGO    DD   DSN=&&OBJSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
//              DISP=(MOD,PASS)
[1] //GO      EXEC PGM=LOADER,PARM='MAP,PRINT,NOCALL,LET',
//              COND=(8,LT,ASM)
[2] //SYSLIN   DD   DSN=&&OBJSET,DISP=(OLD,DELETE)
[3] //SYSLOUT  DD   SYSOUT=A
```

[1] This statement initiates the 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 linkage editor and the LET option is the same as for 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.

Figure 8. Cataloged procedure for assembly and loader-execution (ASMFCG)

EXAMPLES

The following examples demonstrate the use of the assembler cataloged procedures. Normally, you will want to change or add parameters to the procedures you use. The examples illustrate how you use the EXEC statement calling the procedure to change or add parameters to EXEC statements in the procedure; and how you add DD statements after the EXEC statement calling the procedure to change or add DD statement parameters. The rules for overriding parts of cataloged procedures for the duration of a job are explained in JCL Reference.

Example 1:

In the procedure ASMFC, the punched object deck can be suppressed and the UNIT and SPACE parameters of data set SYSUT1 can be respecified by coding the following statements:

```
//stepname   EXEC     ASMFC,PARM=NODECK
//SYSUT1     DD       UNIT=2311,SPACE=(200,(300,40))
//SYSIN      DD       *
             .
             source statements
             .
/*
```

Example 2:

In the procedure ASMFCLG, the assembler listing can be suppressed and the COND parameter, which sets conditions for execution of the linkage editor, can be changed by the following statements:

```
//stepname   EXEC     ASMFCLG,PARM.ASM=(NOLIST,OBJECT),
//                    COND.LKED=(8,LT,PREVSTEP.ASM)
//ASM.SYSIN  DD       *
             .
             source statements
             .
/*
```

Here PREVSTEP is the name of a previous EXEC statement calling an assembler procedure in the same job.
Note: You cannot override individual options in the PARM field. The whole PARM field is always overridden. Therefore, you must repeat OBJECT in the example above.

Example 3:

The following example shows the use of the procedure ASMFCL to:

- Read input from a unlabeled nine-track tape on tape drive 282. The tape has a blocking factor of ten.
- Put the output listing on a tape labeled VOLID=TAPE10, with a data set name of PROG1 and a blocking factor of five (605 divided by 121, the record size for the assembler listing).
- Block the SYSGO output of the assembler and use it as input to the linkage editor with a blocking factor of five.
- Link-edit the module only if there are no errors in the assembly (COND=0).

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

```
//                EXEC  ASMFCL,COND.LKED=(0,NE,ASM)
//ASM.SYSPRINT    DD    DSN=PROGR1,UNIT=TAPE,DISP=(NEW,KEEP),
//                      VOL=SER=TAPE10,DCB=BLKSIZE=605
//ASM.SYSGO       DD    DCB=BLKSIZE=400
//ASM.SYSIN       DD    UNIT=282,LABEL=(,NL),DISP=OLD,
//                      DCB=(RECFM=FSB,BLKSIZE=800)
//LKED.SYSLMOD    DD    DSN=USER.LIBRARY(PROG),DISP=OLD
```

Note: The order in which the overriding DD statements are specified corresponds to the order of DD statements in the procedure. For example, SYSPRINT precedes SYSGO in step ASM. The DD name ASM.SYSIN is placed last among the overriding statements for step ASM, because SYSIN does not exist in step ASM of the procedure.

Example 4:

The following example shows assembly of two programs, link editing of the two object modules into one load module, and execution of the load module:

```
//STEP1         EXEC    ASMFC,PARM.ASM=OBJ
//ASM.SYSGO     DD      DSN=&&OBJSET,UNIT=SYSSQ,SPACE=(80,(200,50)),
                        DISP=(MOD,PASS),DCB=BLKSIZE=400
//ASM.SYSIN     DD      *
                .
                source module 1
                .
/*
//STEP2         EXEC    ASMFCLG
//ASM.SYSGO     DD      DCB=BLKSIZE=400,DISP=(MOD,PASS)
//ASM.SYSIN     DD      *
                .
                source module 2
                .
/*
//LKED.SYSLIN   DD      DCB=BLKSIZE=400
//LKED.SYSIN    DD      *
                ENTRY   PROG
/*
//GO.ddname     DD
//               .              (dd cards for GO step)
```

The LKED.SYSIN statement indicates that input to the linkage editor follows. In this case it is a linkage editor control statement. ENTRY, which identifies PROG, an external symbol in one of the two modules, as the entry point into the load module. When the load module is executed, that point in the module gets control first.

JCL Reference provides additional information on overriding techniques.

The Assembler Listing

This section 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".

The six parts of the assembler listing are:

- External symbol dictionary (ESD)
- Source and object program
- Relocation dictionary (RLD)
- Symbol cross reference
- Literal cross reference
- Diagnostics and statistics

Figure 9 shows the different parts of the listing. The function and purpose of each of them, as well as the individual details, are explained in the following text and illustrations.

```
EXAM                                          EXTERNAL SYMBOL DICTIONARY                                    PAGE    1

SYMBOL   TYPE  ID   ADDR  LENGTH LDID                                                          ASM 0100 09.46 01/05/72

          PC  0001 000000 0001C0
SEARCH    LD       000024        0001
```

```
                                                                                                          PAGE    3
EXAM     SAMPLE PROGRAM

                                   STMT  SOURCE STATEMENT                                    ASM 0100 09.46 01/05/72
  LOC  OBJECT CODE     ADDR1 ADDR2
                                   52 ************************************************************************ SAMPL050
                                   53 *         MAIN ROUTINE                                                 * SAMPL051
                                   54 ************************************************************************ SAMPL052
                                   55           CSECT                                                          SAMPL053
000000                             56           ENTRY SEARCH                                                   SAMPL054
                                   57 BEGIN     BALR  R12,0                ESTABLISH ADDRESSABILITY OF PROGRAM SAMPL055
000000 05C0                        58           USING *,R12                AND TELL THE ASSEMBLER              SAMPL056
                             00002 59           LM    R5,R7,=A(LISTAREA,16,LISTEND)      LOAD LIST AREA PARAMS SAMPL057
000002 9857 C1A6       001A8       60           USING LIST,R5              REGISTER 5 POINTS TO THE LIST       SAMPL058
                             00000 61 MORE      BAL   R14,SEARCH           FIND LIST ENTRY IN TABLE            SAMPL059
000006 45E0 C022       00024       62           TM    SWITCH,NONE          CHECK TO SEE IF NAME WAS FOUND      SAMPL060
00000A 9180 C020       00022       63           BO    NOTTHERE             BRANCH IF NOT                       SAMPL061
00000E 4710 C018       0001A       64           USING TABLE,R1             REGISTER 1 NOW POINTS TO TABLE ENTRY SAMPL062
                             00000 65           MOVE  TSWITCH,LSWITCH      MOVE FUNCTIONS                      SAMPL063

         *** ERROR ***             66+     1,IMPROPER OPERAND TYPES, NO STATEMENTS GENERATED
                                   67           MOUE  TNUMBER,LNUMBER      FROM LIST ENTRY                     SAMPL066

         *** ERROR ***             68           MOVE  TADDRESS,LADDRESS    TO TABLE ENTRY                      SAMPL069
                                   69++         NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO                   SAMPL028
                                   70+          L          2,LADDRESS                                          SAMPL029
000012 5820 500C       0000C       71+          ST         2,TADDRESS                                          SAMPL030
000016 5020 1004       00004       72 NOTTHERE  OI    LSWITCH,NONE         TURN ON SWITCH IN LIST ENTRY        SAMPL075
00001A 9680 5008       00008       73           BXLE  R5,R6,MORE           LOOP THROUGH THE LIST               SAMPL076
00001E 8756 C004       00006       74           EOJ                        END OF PROGRAM, USER LIBRARY MACRO  SAMPL077
```

```
EXAM                                          RELOCATION DICTIONARY                                         PAGE    6

POS.ID   REL.ID   FLAGS   ADDRESS                                                                ASM 0100 09.46 01/05/72

 0001     0001      0C     000154
 0001     0001      0C     000164
 0001     0001      0C     000174
 0001     0001      0C     0001A8
 0001     0001      0C     0001B0
```

```
EXAM                                          CROSS-REFERENCE                                               PAGE    7

SYMBOL    LEN   VALUE   DEFN    REFERENCES                                                   ASM 0100 09.46 01/05/72

BEGIN     00002 00000000 00057  00135 00143 00183
HIGHER    00002 0000004A 00090  00085
LADDRESS  00004 0000000C 00174  00070
LIST      00001 00000000 00170  00060
LISTAREA  00008 00000148 00132  00184
LISTEND   00008 00000198 00152  00184
LNAME     00008 00000000 00171  00084
LNUMBER   00003 00000009 00173
LOOP      00004 00000030 00083  00088 00091 00139
```

```
EXAM                                          LITERAL CROSS-REFERENCE                                       PAGE    8

SYMBOL    LEN   VALUE   DEFN    REFERENCES                                                   ASM 0100 09.46 01/05/72

=A(LISTAREA,16,LISTEND)
          00004 000001A8 00184  00059
=F'128,4,128'
          00004 000001B4 00185  00081
```

```
EXAM                                          ASSEMBLER DIAGNOSTICS AND STATISTICS                          PAGE    9

STMT  ERROR CODE   MESSAGE                                                                ASM 0100 09.46 01/05/72

    0  IFO076      SEQUENCE SYMBOL .TYPECGK IS UNDEFINED IN MACRO MOVE
   36  IFO016      ILLEGAL OR INVALID NAME FIELD
   65  IFO090      UNDEFINED SEQUENCE SYMBOL ENCOUNTERED DURING CONDITIONAL ASSEMBLY
   66  IFO197      *** MNOTE ***
```

Figure 9. Assembler listing

External Symbol Dictionary (ESD)

The external symbol dictionary (ESD) describes the contents of the ESD records included in the object module produced by the assembler. It describes to the linkage editor or loader the control sections and external symbols defined in the module.

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

The ESD is explained in detail in Figure 10. For a full understanding of the terms and concepts used in the figure, refer to "Section E: Program Sectioning" and "Section F: Addressing" in Assembler Language.

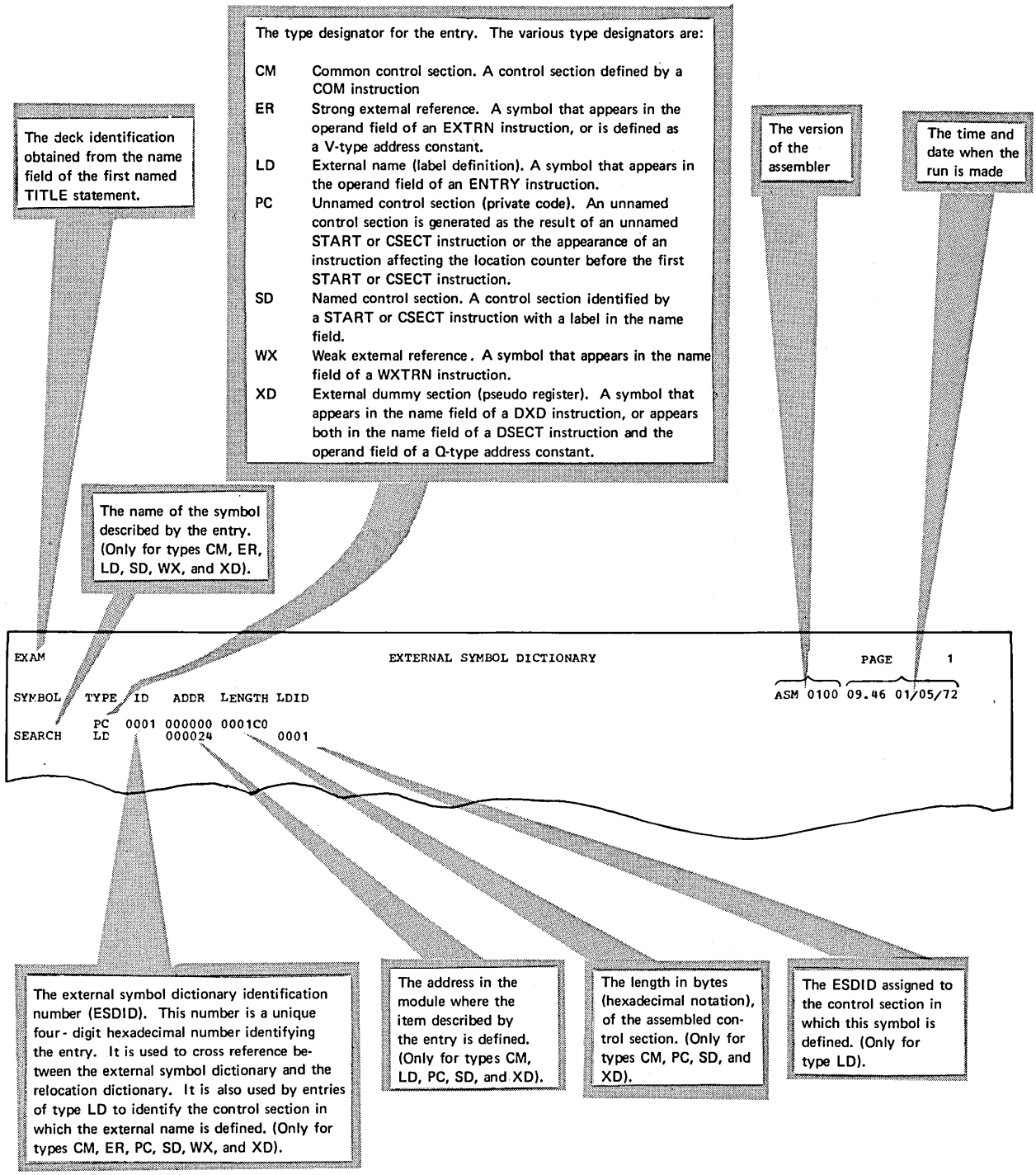


Figure 10. External symbol dictionary

The Source and Machine Language Statements

The second section of the listing contains a copy of the source statements of the module together with a copy of the object code produced by the assembler for each of the source statements.

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 together 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.

The source and machine language statements section is described in detail in Figure 11. For terms that you are unfamiliar with, refer to Assembler Language.

SOURCE STATEMENT FIELDS

The contents of the source statement fields in the listing (see Figure 11) are as follows:

- All source statements except listing control statements are listed, including statements generated from macros and inserted by COPY instructions.
- The definitions of library macros that are called by the program are listed only if the LIBMAC option has been specified.
- The statements generated as the result of a macro instruction are listed after the macro instruction in the listing unless PRINT NOGEN is in effect.
- Unless the NOALOGIC option has been specified, assembler and machine instructions with variable symbols in open code are listed both as they appear in the input to the assembler and with values substituted for the variable symbols.
- When the assembler detects an error, it normally inserts an error indicator in the listing after the statement in error, and prints an error message in the diagnostics and statistics section. Using the FLAG option you can suppress error messages below a severity code that you choose.
- MNOTE messages appear inline where they are generated. MNOTE messages can be suppressed in the same way as error messages using the FLAG option.
- Literals that have not been assigned locations by LTORG instructions appear after the END instruction.
- A generated statement has the same format as the statement from which it was generated, unless a substituted value is longer than the variable symbol used in the model statement.
- Any statement in which the assembler finds an error is listed, even if it would not otherwise be listed. (For example, an AIF statement

in a called library macro definition).

- For a statement generated from a macro definition, columns 73-80 contain the columns from the model statement from which it was generated.

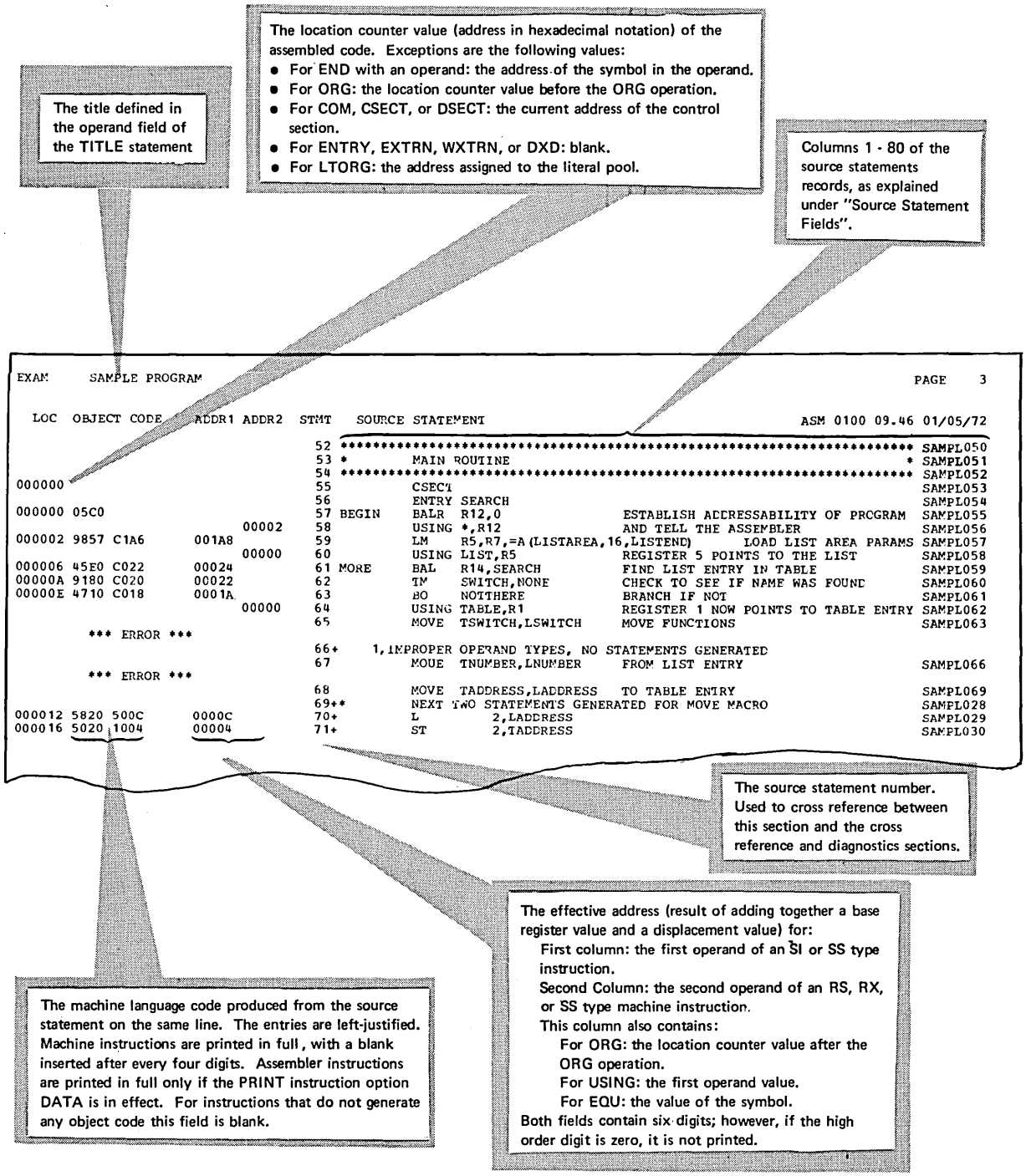


Figure 11. Source and machine language statements

Relocation Dictionary (RLD)

The relocation dictionary (RLD) describes the contents of the RLD records passed to the linkage editor or loader in the object module. The entries describe those address constants in the module that are affected by program relocation.

The section helps you find the relocatable constants in your program.

The RLD section is described in detail in Figure 12. For a description of the different address constants mentioned in the figure, refer to the section "G3 -- Defining Data", in Assembler Language.

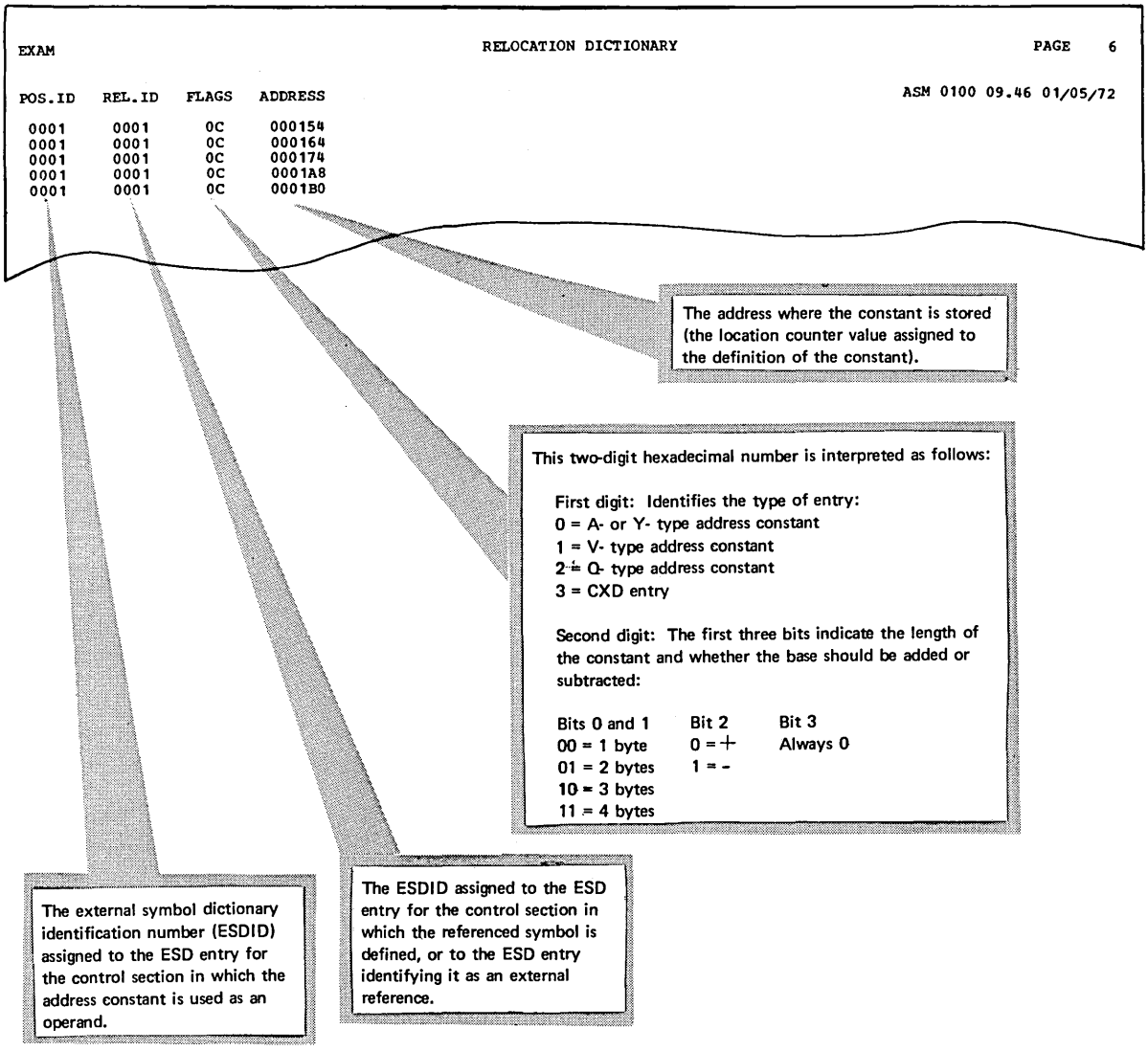


Figure 12. Relocation dictionary

Symbol Cross Reference

The symbol cross reference section of the listing lists the symbols used in the module, indicating both where they are defined and where they are referenced. 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.

The symbol cross reference section contains all symbols in the module, except those appearing in the operand field of V-type address constants. Thus, symbols that are not listed in the source and machine language statements section because of a PRINT OFF or PRINT NOGEN instruction will appear in the cross reference table. (For a description of V-type address constants and the PRINT instruction, refer to Assembler Language.

Symbols that are undefined but referenced will also be listed, and identified as undefined. Duplicate definitions will also be identified in the table.

Figure 13 describes in detail the items of the cross reference table.

Note: The cross reference entry for a symbol used in a literal refers to the assembled literal in the literal pool. Look up the literal cross reference table to find where the symbol is used.

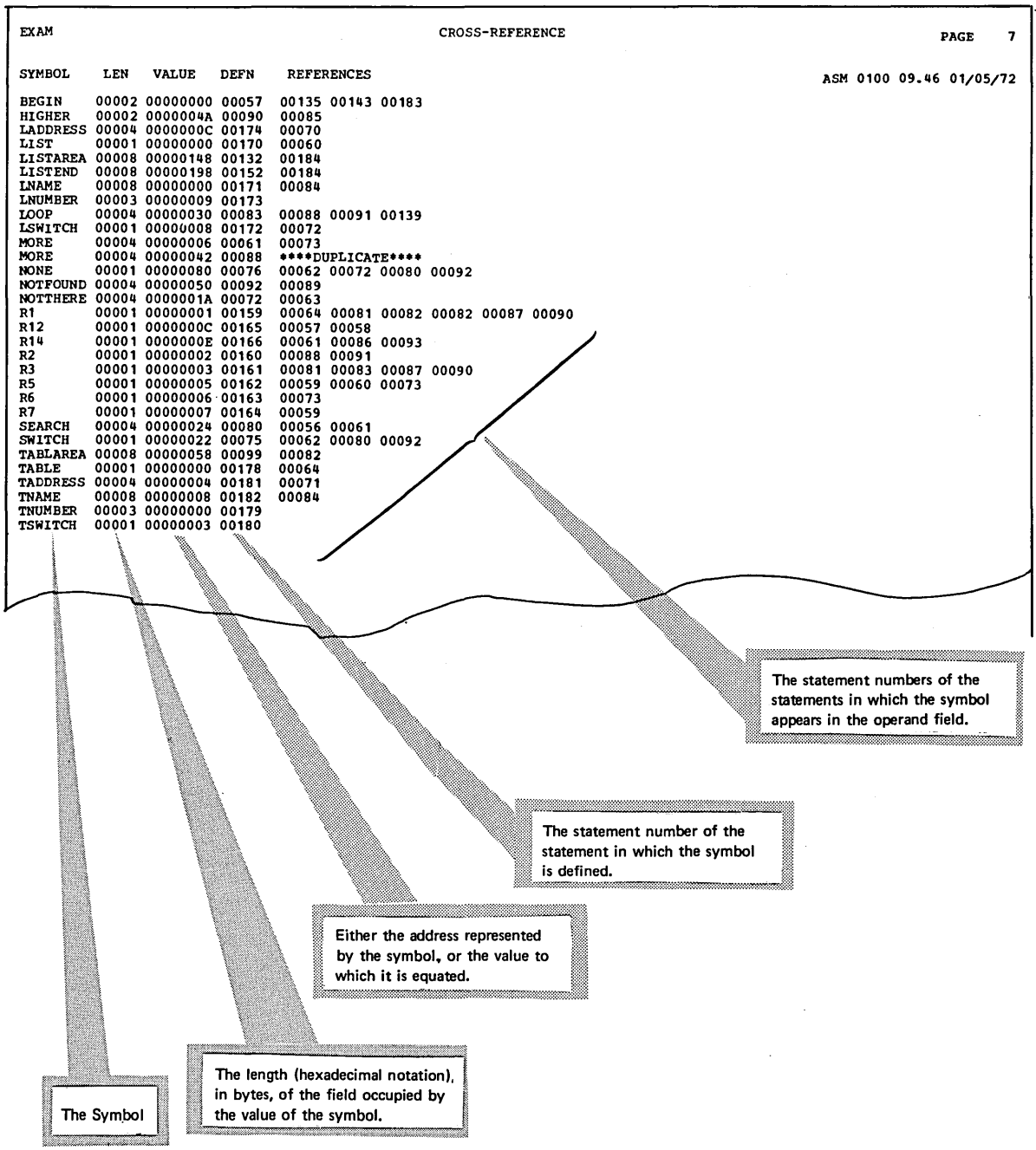


Figure 13. Symbol cross reference

Literal Cross Reference

The literal cross reference section lists all the literals that are used in the program.

Figure 14 gives a detailed explanation of the items of the literal cross reference table.

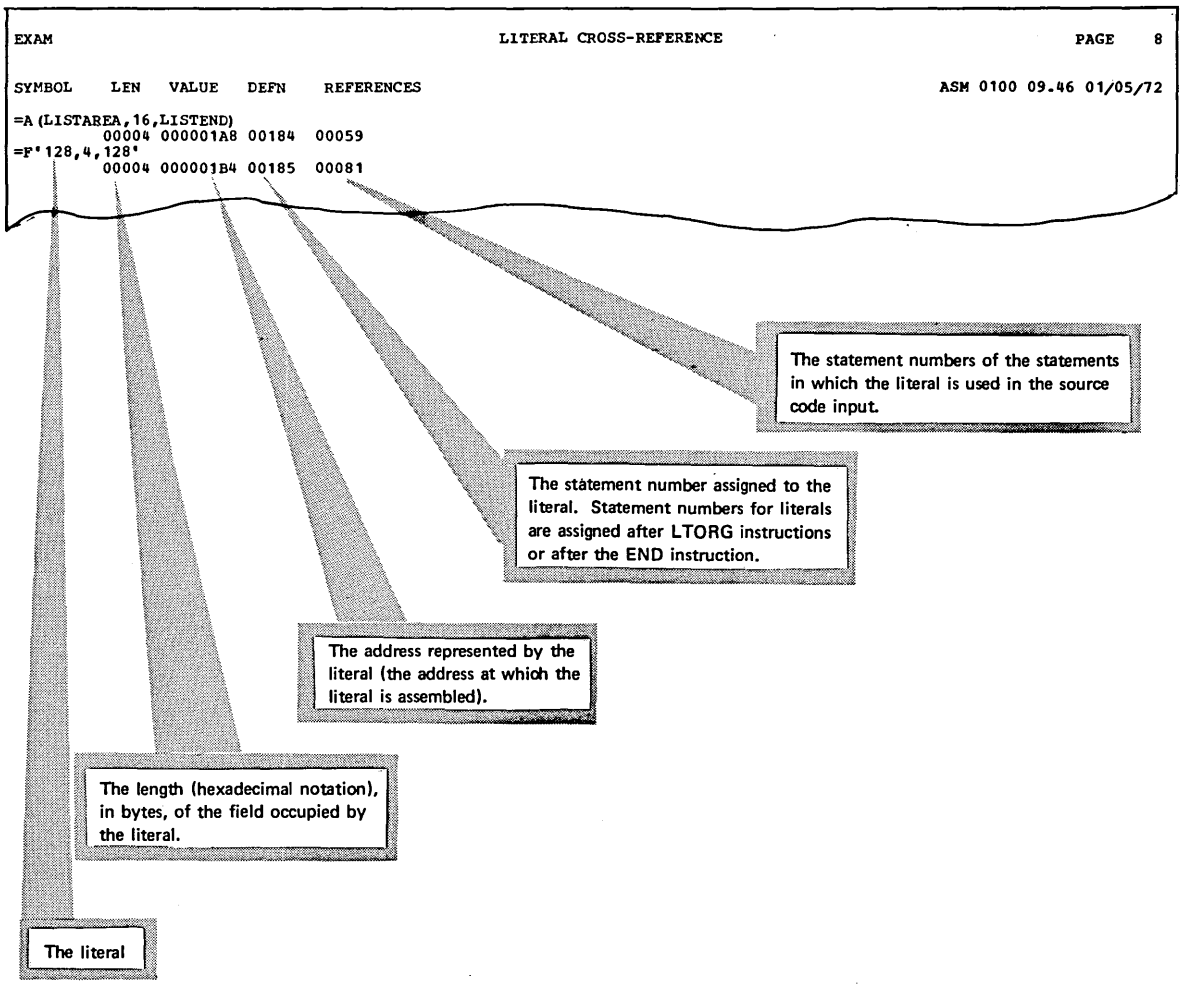


Figure 14. Literal cross reference

Diagnostics and Statistics

Figure 15 gives a detailed explanation of the diagnostics and statistics section of the listing. The following information may also be helpful in interpreting this section.

The diagnostic messages issued by the assembler are fully documented in Appendix G of this manual.

Error messages with the text IFO197 ***MNOTE*** indicate that an MNOTE message has been written in the source statement section of the listing. The MNOTE message is given a statement number which is indicated together with this diagnostic message.

Errors encountered during the processing of library macro definitions reference the END statement. (This is because library macros are read in by the assembler after the source code.) However, if you specify the LIBMAC assembler option, all system macro definitions will be listed after the END statement; an error will then reference the statement within the macro definition that caused the error.

To suppress error messages and MNOTE messages below a specified severity level, you can use the FLAG option.

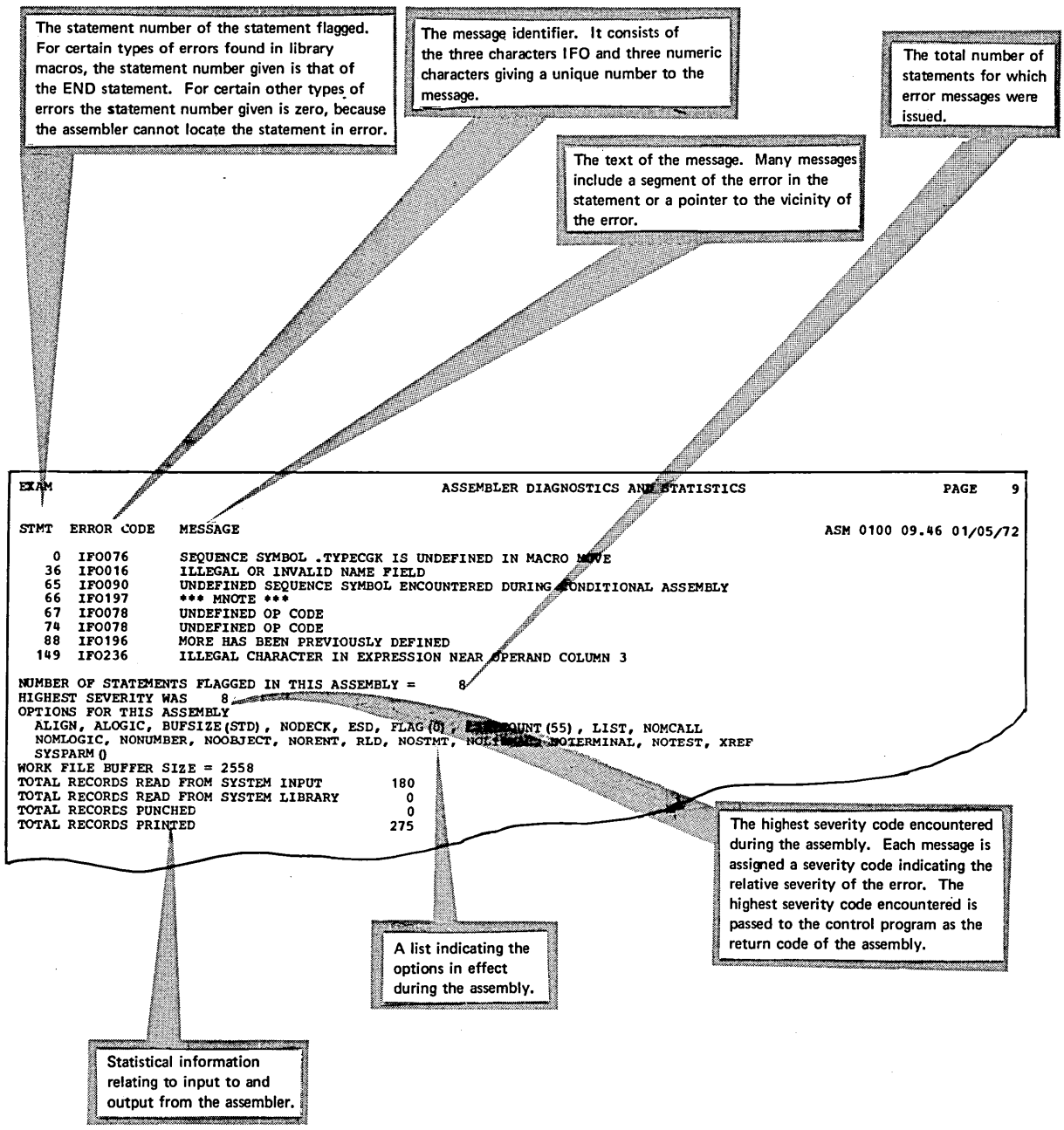


Figure 15. Diagnostics and statistics

Programming Considerations

The purpose of this section is to serve as a bridge between Assembler Language and other manuals that you will use frequently when programming in the assembler language. Among the topics discussed are:

- Designing your program
- Specifying the entry point into your program.
- Linking with modules written in other programming languages.
- Linking with processing programs.

Designing Your Program

When you design your program to run under VS, you must make sure that it follows the conventions required by that operating system. The minimum requirements for a very simple program are given in Figure 16.

However, you will hardly ever write such a simple program and will therefore want to refer to the section "Program Design" in OS/VS Supervisor Services and Macro Instructions. Among the topics covered there are:

- The linkage registers that the operating system uses in passing control between various components of the control program, and between the control program and your problem program. You should use the same registers when calling your own programs.
- Acquiring the information in the PARM field of the EXEC statement. In the same way as the assembler checks the options you specify for it in the PARM field, you can have your own program check the contents of that field.
- Saving the calling program's registers, so that they are not modified by the called program.
- Establishing a base register.
- Providing a save area, so that any programs called by your program can save the contents of your registers and restore the contents upon return. Note that certain system macro instructions (such as GET or PUT) call subroutines that assume that your program has provided a save area.
- Virtual storage considerations.
- Task creation.

The following coding shows the minimum number of instructions you need for a simple program. The program will be less than 4096 bytes long and will consist of only one control section. It will not call any subroutines or use any other IBM-supplied macros than SAVE and RETURN.

```
CSA         SAVE      (14,12)      Save registers for calling routine
            USING     CSA,15       The control program passes control
                                   to the routine using register 15;
                                   use that register as a base
            ST        13,SAVE13    Store address of calling routine's
                                   save area
            .
            (your program)
            .
            L         13,SAVE13    Reload address of save area
            RETURN    (14,12)      Return to calling routine in AOS
SAVE13      DS        F            Space to save address of calling
            .                      routine's save area
            (your constants and data areas)
            .
            END
```

Figure 16. Minimum requirements for a simple program

Specifying the Entry Point into Your Program

When your object module is link edited, either alone, or together with other modules, the entry point into the load module produced is determined by the linkage editor. (The entry point is the address in the load module to which control is given by the control program, when the load module is to be executed.)

You can use the assembler END instruction or the linkage editor ENTRY control statement to specify the entry point to the linkage editor, as explained under "Output From The Linkage Editor" in Linkage Editor and Loader.

Linking with Modules Produced by other Language Translators

The modules produced by the assembler can be combined with other modules by the linkage editor. These modules can be object modules or load modules, and may have been originally written in any of the languages supported by the operating system. 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.

However, when linking between modules produced by different language translators you must make sure that each module conforms to the data formats and linkage conventions required. If input/output operations are performed, you must also make sure that the appropriate DD statements are supplied for the data sets used in the different modules. For information on the requirements for linking between modules written in the assembler language and the problem-oriented languages, refer to the programmer's guide for the particular compiler you are using.

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. When calling a program dynamically, make sure you follow the OS/VS linking conventions described under "Program Design" in OS/VS Supervisor Services and Macro Instructions. 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. Appendix D 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.

Adding Macro Definitions to a Library

You can include your own macro definitions or other sections of often-used source code in the system macro library or in a private library that you concatenate with the system macro library. A macro library can consist of both macro definitions and sections of code to be inserted by the COPY assembler instruction.

You use the IEBUPDTE program to add members to a macro library. For further information on IEBUPDTE and the utility control statements needed, refer to OS/VS Utilities, Order No. GC35-0005. The following example shows how a new macro definition, NEWMAC, is added to the system macro library (SYS1.MACLIB).

```
//CATMAC    JOB       12345,BROWN.JR,...
//STEP1     EXEC      PGM=IEBUPDTE,PARM=MOD
//SYSUT1    DD        DSN=SYS1.MACLIB,DISP=OLD
//SYSUT2    DD        DSN=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.

Appendix A. Glossary

The following terms are defined as they are used in this manual. If you do not find the term you are looking for, refer to the Index or to the IBM Data Processing Glossary, Order No. GC20-1699.

The terms are of three different kinds:

- Definitions made by the American National Standards Institute (ANSI). Such definitions are marked by an asterisk (*).
- Definitions valid for OS. Such definitions are marked by an O.
- Definitions of terms that are used in describing the logic of the OS Assembler. They are included here only because they are used in the assembler diagnostic messages. For further information on these terms, refer to OS/VS Assembler Logic, SY33-8041. Such definitions are marked by an A.

> 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.

This glossary does not explain terms pertaining to the assembler language. Such terms are covered in the glossary of Assembler Language.

O 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.

O assembler instruction: An assembler language source statement that causes the assembler to perform a specific operation. Assembler instructions are not translated into machine instructions.

O 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.

O assembler option: A function of the assembler requested for a particular job step.

O auxiliary storage: Online storage other than main storage; for example storage on magnetic tapes or on direct access devices.

O catalged procedure: A set of job control statements that has been placed in a partitioned data set called the procedure library, and can be retrieved by naming it in an execute (EXEC) statement or started by the START command.

O concatenated data sets: A group of logically connected data sets that are treated as one data set for the duration of a job step.

* control program: A program that is designed to schedule and supervise the performance of data processing work by a computing system.

O control section: 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.

O data set: The major unit of data storage and retrieval in the operating system, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

★ diagnostic: Pertaining to the detection and isolation of a malfunction or mistake.

A edited text: Source statements modified by the assembler for internal use. The initial processing of the assembler is referred to as editing.

O entry point: A location in a module to which control can be passed from another module or from the control program.

ESD: (See external symbol dictionary)

O execute (EXEC) statement: A job control language (JCL) statement that marks the beginning of a job step and identifies the program to be executed or the cataloged or in-stream procedure to be used.

O external symbol dictionary (ESD): Control information associated with an object or load module which identifies the external symbols in the module.

A 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.

A 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.

O input stream: The sequence of job control statements and data submitted to an operating system on an input unit especially activated for this purpose by the operator.

instruction:

★ 1. A statement that specifies an operation and the values and locations of its operands.

2. (See assembler instruction, machine instruction, and macro instruction)

JCL: (See job control language)

★ job: A specified group of tasks prescribed as a unit of work for a computer. By extension, a job usually includes all necessary computer programs, linkages, files, and instructions to the operating system.

O 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.

job step:

★ 1. The execution of a computer program explicitly identified by a job control statement. A job may specify that several job steps be executed.

o 2. A unit of work associated with one processing program or one cataloged procedure and related data. A job consists of one or more job steps.

o jobname: The name assigned to the JOB statement; it identifies the job to the system.

★ language: A set of representations, conventions, and rules used to convey information.

o 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: (See partitioned data set)

o 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.

o 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.

o linking conventions: A set of conventions for passing control between different routines of the operating system.

o 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.

o load module library: A partitioned data set that is used to store and retrieve load modules.

o 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.

A 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.

o 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: (See macro instruction and macro definition)

macro call: (See macro instruction)

o 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 expansion: (See macro generation)

o 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.

o macro instruction (macro call): An assembler language statement that causes the assembler to process a predefined set of statements called a macro definition.

o 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.

o main storage: All program addressable storage from which instructions may be executed and from which data can be loaded directly into registers.

module: (see load module, object module, and source module)

o 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.

★ online storage: Storage under the control of the central processing unit.

o 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.

A 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.

option: (See assembler option)

o partitioned data set (library): A data set in direct access storage that is divided into partitions, called members, each of which can contain a program or a part of a program. Each partitioned data set contains a directory (or index) that the control program can use to locate a program in the partitioned data set.

o procedure step: A unit of work associated with one processing program and related data within a cataloged procedure. A cataloged procedure consists of one or more procedure steps.

o 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.

o program:

1. 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.

2. To write a program.

programmer macro definition: (See source macro definition)

o 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.

o 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.

o sequential data set: A data set whose records are organized on the basis of their successive physical positions such as on magnetic tape.

o 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.

A 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.

o source macro definition: A macro definition included in a source module, either physically or as the result of a COPY instruction.

o source module: The source statements that constitute the input to a language translator for a particular translation.

o source statement: A statement written in symbols of a programming language.

★ statement: A meaningful expression or generalized instruction in a source language.

step: (See job step and procedure step)

o stepname: The name assigned to an execute (EXEC) statement. It identifies a job step within a job.

o symbolic parameter:

1. In JCL, a symbol preceded by an ampersand that appears in a cataloged procedure. Values are assigned to symbolic parameters when the procedure in which they appear is called.

2. In assembler programming, a variable symbol declared in the prototype statement of a macro definition.

A symbol file: A data set used by the assembler for symbol definitions and references and literals.

o system macro definition: Loosely, an IBM-supplied library macro definition which provides access to operating system facilities.

★ terminal. A point in a system or communication network at which data can either enter or leave or both.

A 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.

★ transform. To change the form of data according to specific rules.

★ translate. To transform statements from one language into another without significantly changing the meaning.

O 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.

Appendix B. Assembler Sample Program

The sample program shown in Figure 17 can be used as a test of the functioning of the assembler after your system has been generated (see OS/VS1 System Generation Reference, Order No. GC26-3791). It also serves serves as a good example of assembler language coding and of the listing produced by the assembler.

The program illustrates the definition and use of user-written macro instructions, use of IBM-supplied macro instructions, use of dummy control sections, and the method of saving and restoring registers upon entry to and exit from a program.

The data to be processed is assembled as part of the program. It consists of a table and a list of entries that are compared with the table. Each item in the table contains an argument name (such as ALPHA) and space in which information concerning the name is to be placed. Each entry in the list contains an argument name and function values. The formats of the table entries and the list entries are different, and both formats are described in dummy sections.

The program searches the table for an argument name in the list. If a match is found, the function values are reformatted and moved to the appropriate table entry. If an argument name in the list cannot be found in the table, a switch is set in the list entry. After all the list entries have been processed, the list area and the table area are compared with a table and a list containing the predefined results. If the tables and lists are equal, the routine executed properly, and a message is written on the operator's console to indicate this.

```
IFOSAMP                          EXTERNAL SYMBOL DICTIONARY          PAGE    1

SYMBOL   TYPE  ID   ADDR   LENGTH LDID                          ASM 0100 15.00 01/03/72

SAMPLR    SD   0001 000000 0003C0
```

Figure 17. Assembler sample program (Part 1 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                                    PAGE    2

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                ASM 0100 15.00 01/03/72

                                     2          PRINT DATA                                                    SAMPL002
                                     3 *                                                                      SAMPL003
                                     4 *        THIS IS THE MACRO DEFINITION                                  SAMPL004
                                     5 *                                                                      SAMPL005
                                     6          MACRO                                                         SAMPL006
                                     7          MOVE  &TO,&FROM                                               SAMPL007
                                     8 .*                                                                     SAMPL008
                                     9 .*       DEFINE SETC SYMBOL                                            SAMPL009
                                    10 .*                                                                     SAMPL010
                                    11          LCLC  &TYPE                                                   SAMPL011
                                    12 .*                                                                     SAMPL012
                                    13 .*       CHECK NUMBER OF OPERANDS                                      SAMPL013
                                    14 .*                                                                     SAMPL014
                                    15          AIF   (N'&SYSLIST NE 2).ERROR1                                SAMPL015
                                    16 .*                                                                     SAMPL016
                                    17 .*       CHECK TYPE ATTRIBUTES OF OPERANDS                             SAMPL017
                                    18 .*                                                                     SAMPL018
                                    19          AIF   (T'&TO NE T'&FROM).ERROR2                               SAMPL019
                                    20          AIF   (T'&TO EQ 'C' OR T'&TO EQ 'G' OR T'&TO EQ 'K').TYPECGK  SAMPL020
                                    21          AIF   (T'&TO EQ 'D' OR T'&TO EQ 'E' OR T'&TO EQ 'H').TYPEDEH  SAMPL021
                                    22          AIF   (T'&TO EQ 'F').MOVE                                     SAMPL022
                                    23          AGO   .ERROR3                                                 SAMPL023
                                    24 .TYPEDEH ANOP                                                          SAMPL024
                                    25 .*                                                                     SAMPL025
                                    26 .*       ASSIGN TYPE ATTRIBUTE TO SETC SYMBOL                          SAMPL026
                                    27 .*                                                                     SAMPL027
                                    28 &TYPE    SETC  T'&TO                                                   SAMPL028
                                    29 .MOVE    ANOP                                                          SAMPL029
                                    30 *        NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO                  SAMPL030
                                    31          L&TYPE   2,&FROM                                              SAMPL031
                                    32          ST&TYPE  2,&TO                                                SAMPL032
                                    33          MEXIT                                                         SAMPL033
                                    34 .*                                                                     SAMPL034
                                    35 .*       CHECK LENGTH ATTRIBUTES OF OPERANDS                           SAMPL035
                                    36 .*                                                                     SAMPL036
                                    37 .TYPECGK AIF   (L'&TO NE L'&FROM OR L'&TO GT 256).ERROR4               SAMPL037
                                    38 *        NEXT STATEMENT GENERATED FOR MOVE MACRO                       SAMPL038
                                    39          MVC   &TO,&FROM                                               SAMPL039
                                    40          MEXIT                                                         SAMPL040
                                    41 .*                                                                     SAMPL041
                                    42 .*       ERROR MESSAGES FOR INVALID MOVE MACRO INSTRUCTIONS            SAMPL042
                                    43 .*                                                                     SAMPL043
                                    44 .ERROR1  MNOTE 1,'IMPROPER NUMBER OF OPERANDS, NO STATEMENTS GENERATED' SAMPL044
                                    45          MEXIT                                                         SAMPL045
                                    46 .ERROR2  MNOTE 1,'OPERAND TYPES DIFFERENT, NO STATEMENTS GENERATED'    SAMPL046
                                    47          MEXIT                                                         SAMPL047
                                    48 .ERROR3  MNOTE 1,'IMPROPER OPERAND TYPES, NO STATEMENTS GENERATED'     SAMPL048
                                    49          MEXIT                                                         SAMPL049
                                    50 .ERROR4  MNOTE 1,'IMPROPER OPERAND LENGTHS, NO STATEMENTS GENERATED'   SAMPL050
                                    51          MEND                                                          SAMPL051
                                    52 *                                                                      SAMPL052
                                    53 *        MAIN ROUTINE                                                  SAMPL053
                                    54 *                                                                      SAMPL054
000000                              55 SAMPLR   CSECT                                                         SAMPL055
                                    56 BEGIN    SAVE  (14,12),,*                                              SAMPL056
```

Figure 17. Assembler sample program
(Part 2 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                                                    PAGE    3

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                ASM 0100 15.00 01/03/72

000000 47F0 F00A       0000A          57+BEGIN    B     10(0,15)                                BRANCH AROUND ID        00860000
000004 05                             58+         DC    AL1(5)                                                          00880000
000005 C2C5C7C9D5                      59+         DC    CL5'BEGIN'                              IDENTIFIER              00900000
00000A 90EC D00C       0000C          60+         STM   14,12,12(13)                            SAVE REGISTERS          01180000
00000E 05C0                           61          BALR  R12,0          ESTABLISH ADDRESSABILITY OF PROGRAM          SAMPL057
                                00010  62          USING *,R12          AND TELL THE ASSEMELER WHAT BASE TO USE      SAMPL058
000010 50D0 C0C0       000D0          63          ST    13,SAVE13                                                     SAMPL059
000014 9857 C398       003A8          64          LM    R5,R7,=A(LISTAREA,16,LISTEND)  LOAD LIST AREA PARAMETERS     SAMPL060
                                00000  65          USING LIST,R5       REGISTER 5 POINTS TO THE LIST                 SAMPL061
000018 45E0 C0C6       000D6          66 MORE     BAL   R14,SEARCH    FIND LIST ENTRY IN TABLE                      SAMPL062
00001C 9180 C0C4       000D4          67          TM    SWITCH,NONE   CHECK TO SEE IF NAME WAS FOUND               SAMPL063
000020 4710 C0B6       000C6          68          BO    NOTTHERE      BRANCH IF NOT                                 SAMPL064
                                00000  69          USING TABLE,R1      REGISTER 1 NOW POINTS TO TABLE ENTRY        SAMPL065
                                       70          MOVE  TSWITCH,LSWITCH          MOVE FUNCTIONS                 SAMPL066
                                       71+*        NEXT STATEMENT GENERATED FOR MOVE MACRO                       SAMPL038
000024 D200 1003 5008 00003 00008      72+         MVC   TSWITCH,LSWITCH                                         SAMPL039
                                       73          MOVE  TNUMBER,LNUMBER                FROM LIST ENTRY          SAMPL067
                                       74+*        NEXT STATEMENT GENERATED FOR MOVE MACRO                       SAMPL038
00002A D202 1000 5009 00000 00009      75+         MVC   TNUMBER,LNUMBER                                         SAMPL039
                                       76          MOVE  TADDRESS,LADDRESS                TO TABLE ENTRY         SAMPL068
                                       77+*        NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO                  SAMPL030
000030 5820 500C       0000C          78+         L       2,LADDRESS                                            SAMPL031
000034 5020 1004       00004          79+         ST      2,TADDRESS                                            SAMPL032
000038 8756 C008       00018          80 LISTLOOP BXLE  R5,R6,MORE    LOOP THROUGH THE LIST                        SAMPL069
00003C D5EF C248 C0F8 00258 00108      81          CLC   TESTTABL(240),TABLAREA                                  SAMPL070
000042 4770 C080       00090          82          BNE   NOTRIGHT                                               SAMPL071
000046 D55F C338 C1E8 00348 001F8      83          CLC   TESTLIST(96),LISTAREA                                   SAMPL072
00004C 4770 C080       00090          84          BNE   NOTRIGHT                                               SAMPL073
                                       85          WTO   'SAMPLE PROGRAM IFOSAMP SUCCESSFUL',ROUTCDE=(2,11)      SAMPL074
000050                                 86+         CNOP  0,4                                                     00480000
000050 4510 C06E       0007E          87+         BAL   1,IHB0005A                      BRANCH AROUND MESSAGE   00500000
000054 0025                           88+         DC    AL2(IHB0005-*)    MESSAGE LENGTH                        00428018
000056 8000                           89+         DC    B'1000000000000000'     MCSFLAGS FIELD                  00428818
000058 E2C1D4D7D3C540D7               90+         DC    C'SAMPLE PROGRAM IFOSAMP SUCCESSFUL' MESSAGE            00429618
000060 D9D6C7D9C1D440C9
000068 C6D6E2C1D4D740E2
000070 E4C3C3C5E2E2C6E4
000078 D3
                                00079  91+IHB0005  EQU   *                                                       00430418
000079 0000                           92+         DC    B'0000000000000000'     DESCRIPTOR CODES                00432018
00007B 4020                           93+         DC    B'0100000000100000'     ROUTING CODES                   00432818
00007E                                94+IHB0005A DS    0H                                                      00560000
00007E 0A23                           95+         SVC   35                                    ISSUE SVC         00600000
000080 58D0 C0C0       000D0          96 EXIT     L     R13,SAVE13                                              SAMPL075
                                      97          RETURN (14,12),RC=0                                           SAMPL076
000084 98EC D00C       0000C          98+         LM    14,12,12(13)                          RESTORE THE REGISTERS   00260000
000088 41F0 0000       00000          99+         LA    15,0(0,0)                             LOAD RETURN CODE        00700000
00008C 07FE                          100+         BR    14                                    RETURN            00800000
                                     101 *                                                                       SAMPL077
                                     102 NOTRIGHT WTO   'SAMPLE PROGRAM IFOSAMP UNSUCCESSFUL',ROUTCDE=(2,11)   SAMPL078
00008E 0700                          103+         CNOP  0,4                                                     00480000
000090 4510 C0B0       000C0         104+NOTRIGHT BAL   1,IHB0007A                      BRANCH AROUND MESSAGE   00500000
000094 0027                          105+         DC    AL2(IHB0007-*)    MESSAGE LENGTH                        00428018
000096 8000                          106+         DC    B'1000000000000000'     MCSFLAGS FIELD                  00428818
000098 E2C1D4D7D3C540D7              107+         DC    C'SAMPLE PROGRAM IFOSAMP UNSUCCESSFUL' MESSAGE          00429618
```

Figure 17. Assembler sample program
(Part 3 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                                                  PAGE    4

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          ASM 0100 15.00 01/03/72

0000A0 D9D6C7D9C1D440C9
0000A8 C6D6E2C1D4D740E4
0000B0 D5E2E4C3C3C5E2E2
0000B8 C6E4D3
                            000BB   108+IHB0007  EQU   *                                                          00430418
0000BB 0000                         109+         DC    B'0000000000000000'     DESCRIPTOR CODES                  00432018
0000BD 4020                         110+         DC    B'0100000000100000'     ROUTING CODES                     00432818
0000C0                              111+IHB0007A DS    0H                                                         00560000
0000C0 0A23                         112+         SVC   35                                 ISSUE SVC                00600000
0000C2 47F0 C070      00080         113          B     EXIT                                                       SAMPL079
0000C6 9680 5008      00008         114 NOTTHERE OI    LSWITCH,NONE TURN ON SWITCH IN LIST ENTRY                 SAMPL080
0000CA 47F0 C028      00038         115          B     LISTLOOP     GO BACK AND LOOP                              SAMPL081
0000CE 0000
0000D0 00000000                     116 SAVE13   DC    F'0'                                                       SAMPL082
0000D4 00                           117 SWITCH   DC    X'00'                                                      SAMPL083
                            00080   118 NONE     EQU   X'80'                                                      SAMPL084
                                    119 *                                                                          SAMPL085
                                    120 *        BINARY SEARCH ROUTINE                                             SAMPL086
                                    121 *                                                                          SAMPL087
0000D5 00
0000D6 947F C0C4      000D4         122 SEARCH   NI    SWITCH,255-NONE TURN OFF NOT FOUND SWITCH                  SAMPL088
0000DA 9813 C3A4      003B4         123          LM    R1,R3,=F'128,4,128' LOAD TABLE PARAMETERS                  SAMPL089
0000DE 4111 C0E8      000F8         124          LA    R1,TABLAREA-16(R1)  GET ADDRESS OF MIDDLE ENTRY            SAMPL090
0000E2 8830 0001      00001         125 LOOP     SRL   R3,1                DIVIDE INCREMENT BY 2                  SAMPL091
0000E6 D507 5000 1008 00000 00008   126          CLC   LNAME,TNAME         COMPARE LIST ENTRY WITH TABLE ENTRY    SAMPL092
0000EC 4720 C0EC      000FC         127          BH    HIGHER              BRANCH IF SHOULD BE HIGHER IN TABLE    SAMPL093
0000F0 078E                         128          BCR   8,R14               EXIT IF FOUND                          SAMPL094
0000F2 1B13                         129          SR    R1,R3               OTHERWISE IT IS LOWER IN THE TABLE    XSAMPL095
                                                                             SO SUBTRACT INCREMENT                 SAMPL096
0000F4 4620 C0D2      000E2         130          BCT   R2,LOOP             LOOP 4 TIMES                           SAMPL097
0000F8 47F0 C0F2      00102         131          B     NOTFOUND            ARGUMENT IS NOT IN THE TABLE           SAMPL098
0000FC 1A13                         132 HIGHER   AR    R1,R3               ADD INCREMENT                          SAMPL099
0000FE 4620 C0D2      000E2         133          BCT   R2,LOOP             LOOP 4 TIMES                           SAMPL100
000102 9680 C0C4      000D4         134 NOTFOUND OI    SWITCH,NONE         TURN ON NOT FOUND SWITCH               SAMPL101
000106 07FE                         135          BR    R14                 EXIT                                   SAMPL102
                                    136 *                                                                          SAMPL103
                                    137 *        THIS IS THE TABLE                                                 SAMPL104
                                    138 *                                                                          SAMPL105
000108                              139          DS    0D                                                         SAMPL106
000108 0000000000000000             140 TABLAREA DC    XL8'0',CL8'ALPHA'                                          SAMPL107
000110 C1D3D7C8C1404040
000118 0000000000000000             141          DC    XL8'0',CL8'BETA'                                           SAMPL108
000120 C2C5E3C140404040
000128 0000000000000000             142          DC    XL8'0',CL8'DELTA'                                          SAMPL109
000130 C4C5D3E3C1404040
000138 0000000000000000             143          DC    XL8'0',CL8'EPSILON'                                        SAMPL110
000140 C5D7E2C9D3D6D540
000148 0000000000000000             144          DC    XL8'0',CL8'ETA'                                            SAMPL111
000150 C5E3C14040404040
000158 0000000000000000             145          DC    XL8'0',CL8'GAMMA'                                          SAMPL112
000160 C7C1D4D4C1404040
000168 0000000000000000             146          DC    XL8'0',CL8'IOTA'                                           SAMPL113
000170 C9D6E3C140404040
000178 0000000000000000             147          DC    XL8'0',CL8'KAPPA'                                          SAMPL114
000180 D2C1D7D7C1404040
```

Figure 17. Assembler sample program
(Part 4 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                    PAGE    5

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                ASM 0100 15.00 01/03/72

000188 0000000000000000              148          DC    XL8'0',CL8'LAMBDA'                    SAMPL115
000190 D3C1D4C2C4C14040
000198 0000000000000000              149          DC    XL8'0',CL8'MU'                        SAMPL116
0001A0 D4E4404040404040
0001A8 0000000000000000              150          DC    XL8'0',CL8'NU'                        SAMPL117
0001B0 D5E4404040404040
0001B8 0000000000000000              151          DC    XL8'0',CL8'OMICRON'                   SAMPL118
0001C0 D6D4C9C3D9D6D540
0001C8 0000000000000000              152          DC    XL8'0',CL8'PHI'                       SAMPL119
0001D0 D7C8C94040404040
0001D8 0000000000000000              153          DC    XL8'0',CL8'SIGMA'                     SAMPL120
0001E0 E2C9C7D4C1404040
0001E8 0000000000000000              154          DC    XL8'0',CL8'ZETA'                      SAMPL121
0001F0 E9C5E3C140404040
                                     155 *                                                    SAMPL122
                                     156 *        THIS IS THE LIST                            SAMPL123
                                     157 *                                                    SAMPL124
0001F8 D3C1D4C2C4C14040              158 LISTAREA DC    CL8'LAMBDA',X'0A',FL3'29',A(BEGIN)    SAMPL125
000200 0A00001D00000000
000208 E9C5E3C140404040              159          DC    CL8'ZETA',X'05',FL3'5',A(LOOP)        SAMPL126
000210 050000050000000E2
000218 E3C8C5E3C1404040              160          DC    CL8'THETA',X'02',FL3'45',A(BEGIN)     SAMPL127
000220 0200002D00000000
000228 E3C1E44040404040              161          DC    CL8'TAU',X'00',FL3'0',A(1)            SAMPL128
000230 0000000000000001
000238 D3C9E2E340404040              162          DC    CL8'LIST',X'1F',FL3'465',A(0)         SAMPL129
000240 1F0001D100000000
000248 C1D3D7C8C1404040              163 LISTEND  DC    CL8'ALPHA',X'00',FL3'1',A(123)        SAMPL130
000250 000000010000007B
                                     164 *                                                    SAMPL131
                                     165 *        THIS IS THE CONTROL TABLE                   SAMPL132
                                     166 *                                                    SAMPL133
000258                               167          DS    0D                                    SAMPL134
000258 000001000000007B              168 TESTTABL DC    FL3'1',X'00',A(123),CL8'ALPHA'        SAMPL135
000260 C1D3D7C8C1404040
000268 0000000000000000              169          DC    XL8'0',CL8'BETA'                      SAMPL136
000270 C2C5E3C140404040
000278 0000000000000000              170          DC    XL8'0',CL8'DELTA'                     SAMPL137
000280 C4C5D3E3C1404040
000288 0000000000000000              171          DC    XL8'0',CL8'EPSILON'                   SAMPL138
000290 C5D7E2C9D3D6D540
000298 0000000000000000              172          DC    XL8'0',CL8'ETA'                       SAMPL139
0002A0 C5E3C14040404040
0002A8 0000000000000000              173          DC    XL8'0',CL8'GAMMA'                     SAMPL140
0002B0 C7C1D4D4C1404040
0002B8 0000000000000000              174          DC    XL8'0',CL8'IOTA'                      SAMPL141
0002C0 C9D6E3C140404040
0002C8 0000000000000000              175          DC    XL8'0',CL8'KAPPA'                     SAMPL142
0002D0 D2C1D7D7C1404040
0002D8 00001D0A00000000              176          DC    FL3'29',X'0A',A(BEGIN),CL8'LAMBDA'    SAMPL143
0002E0 D3C1D4C2C4C14040
0002E8 0000000000000000              177          DC    XL8'0',CL8'MU'                        SAMPL144
0002F0 D4E4404040404040
0002F8 0000000000000000              178          DC    XL8'0',CL8'NU'                        SAMPL145
000300 D5E4404040404040
```

Figure 17. Assembler sample program
(Part 5 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                    PAGE    6

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                    ASM 0100 15.00 01/03/72

000308 0000000000000000                179          DC    XL8'0',CL8'OMICRON'                      SAMPL146
000310 D6D4C9C3D9D6D540
000318 0000000000000000                180          DC    XL8'0',CL8'PHI'                          SAMPL147
000320 D7C8C94040404040
000328 0000000000000000                181          DC    XL8'0',CL8'SIGMA'                        SAMPL148
000330 E2C9C7D4C1404040
000338 00000505000000E2                182          DC    FL3'5',X'05',A(LOOP),CL8'ZETA'           SAMPL149
000340 E9C5E3C140404040
                                       183 *                                                        SAMPL150
                                       184 *        THIS IS THE CONTROL LIST                        SAMPL151
                                       185 *                                                        SAMPL152
000348 D3C1D4C2C4C14040                186 TESTLIST DC    CL8'LAMBDA',X'0A',FL3'29',A(BEGIN)       SAMPL153
000350 0A00001D00000000
000358 E9C5E3C140404040                187          DC    CL8'ZETA',X'05',FL3'5',A(LOOP)           SAMPL154
000360 05000005000000E2
000368 E3C8C5E3C1404040                188          DC    CL8'THETA',X'82',FL3'45',A(BEGIN)        SAMPL155
000370 8200002D00000000
000378 E3C1E44040404040                189          DC    CL8'TAU',X'80',FL3'0',A(1)               SAMPL156
000380 8000000000000001
000388 D3C9E2E340404040                190          DC    CL8'LIST',X'9F',FL3'465',A(0)            SAMPL157
000390 9F0001D100000000
000398 C1D3D7C8C1404040                191          DC    CL8'ALPHA',X'00',FL3'1',A(123)           SAMPL158
0003A0 000000010000007B
                                       192 *                                                        SAMPL159
                                       193 *        THESE ARE THE SYMBOLIC REGISTERS                SAMPL160
                                       194 *                                                        SAMPL161
                              00000    195 R0       EQU   0                                        SAMPL162
                              00001    196 R1       EQU   1                                        SAMPL163
                              00002    197 R2       EQU   2                                        SAMPL164
                              00003    198 R3       EQU   3                                        SAMPL165
                              00005    199 R5       EQU   5                                        SAMPL166
                              00006    200 R6       EQU   6                                        SAMPL167
                              00007    201 R7       EQU   7                                        SAMPL168
                              0000C    202 R12      EQU   12                                       SAMPL169
                              0000D    203 R13      EQU   13                                       SAMPL170
                              0000E    204 R14      EQU   14                                       SAMPL171
                              0000F    205 R15      EQU   15                                       SAMPL172
                                       206 *                                                        SAMPL173
                                       207 *        THIS IS THE FORMAT DEFINITION OF LIST ENTRYS    SAMPL174
                                       208 *                                                        SAMPL175
000000                                 209 LIST     DSECT                                          SAMPL176
000000                                 210 LNAME    DS    CL8                                      SAMPL177
000008                                 211 LSWITCH  DS    C                                        SAMPL178
000009                                 212 LNUMBER  DS    FL3                                      SAMPL179
00000C                                 213 LADDRESS DS    F                                        SAMPL180
                                       214 *                                                        SAMPL181
                                       215 *        THIS IS THE FORMAT DEFINITION OF TABLE ENTRYS   SAMPL182
                                       216 *                                                        SAMPL183
000000                                 217 TABLE    DSECT                                          SAMPL184
000000                                 218 TNUMBER  DS    FL3                                      SAMPL185
000003                                 219 TSWITCH  DS    C                                        SAMPL186
000004                                 220 TADDRESS DS    F                                        SAMPL187
000008                                 221 TNAME    DS    CL8                                      SAMPL188
000000                                 222          END   BEGIN                                    SAMPL189
0003A8 000001F800000010                223                =A(LISTAREA,16,LISTEND)
```

Figure 17. Assembler sample program
(Part 6 of 11)

```
IFOSAMP   - SAMPLE PROGRAM                                                    PAGE    7

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                  ASM 0100 15.00 01/03/72

0003B0 00000248
0003B4 0000008000000004             224               =F'128,4,128'
0003BC 00000080
```

Figure 17. Assembler sample program
(Part 7 of 11)

```
IFOSAMP                                    RELOCATION DICTIONARY                    PAGE    8

POS.ID   REL.ID   FLAGS   ADDRESS                                  ASM 0100 15.00 01/03/72

 0001     0001      0C     000204
 0001     0001      0C     000214
 0001     0001      0C     000224
 0001     0001      0C     0002DC
 0001     0001      0C     00033C
 0001     0001      0C     000354
 0001     0001      0C     000364
 0001     0001      0C     000374
 0001     0001      0C     0003A8
 0001     0001      0C     0003B0
```

Figure 17. Assembler sample program
(Part 8 of 11)

```
IFOSAMP                                    CROSS-REFERENCE                          PAGE    9

SYMBOL    LEN    VALUE    DEFN    REFERENCES                               ASM 0100 15.00 01/03/72

BEGIN    00004 00000000 00057  00158 00160 00176 00186 00188 00222
EXIT     00004 00000080 00096  00113
HIGHER   00002 000000FC 00132  00127
IHB0005  00001 00000079 00091  00088
IHB0005A 00002 0000007E 00094  00087
IHB0007  00001 000000BB 00108  00105
IHB0007A 00002 000000C0 00111  00104
LADDRESS 00004 0000000C 00213  00078
LIST     00001 00000000 00209  00065
LISTAREA 00008 000001F8 00158  00083 00223
LISTEND  00008 00000248 00163  00223
LISTLOOP 00004 00000038 00080  00115
LNAME    00008 00000000 00210  00126
LNUMBER  00003 00000009 00212  00075
LOOP     00004 000000E2 00125  00130 00133 00159 00182 00187
LSWITCH  00001 00000008 00211  00072 00114
MORE     00004 00000018 00066  00080
NONE     00001 00000080 00118  00067 00114 00122 00134
NOTFOUND 00004 00000102 00134  00131
NOTRIGHT 00004 00000090 00104  00082 00084
NOTTHERE 00004 000000C6 00114  00068
R0       00001 00000000 00195
R1       00001 00000001 00196  00069 00123 00124 00124 00129 00132
R12      00001 0000000C 00202  00061 00062
R13      00001 0000000D 00203  00096
R14      00001 0000000E 00204  00066 00128 00135
R15      00001 0000000F 00205
R2       00001 00000002 00197  00130 00133
R3       00001 00000003 00198  00123 00125 00129 00132
R5       00001 00000005 00199  00064 00065 00080
R6       00001 00000006 00200  00080
R7       00001 00000007 00201  00064
SAMPLR   00001 00000000 00055
SAVE13   00004 000000D0 00116  00063 00096
SEARCH   00004 000000D6 00122  00066
SWITCH   00001 000000D4 00117  00067 00122 00134
TABLAREA 00008 00000108 00140  00081 00124
TABLE    00001 00000000 00217  00069
TADDRESS 00004 00000004 00220  00079
TESTLIST 00008 00000348 00186  00083
TESTTABL 00003 00000258 00168  00081
TNAME    00008 00000008 00221  00126
TNUMBER  00003 00000000 00218  00075
TSWITCH  00001 00000003 00219  00072
```

Figure 17. Assembler sample program
(Part 9 of 11)

```
IFOSAMP                                 LITERAL CROSS-REFERENCE                 PAGE   10

SYMBOL    LEN   VALUE    DEFN    REFERENCES                          ASM 0100 15.00 01/03/72

=A(LISTAREA,16,LISTEND)
          00004 000003A8 00223   00064
=F'128,4,128'
          00004 000003B4 00224   00123
```

Figure 17. Assembler sample program
(Part 10 of 11)

```
IFOSAMP                                 ASSEMBLER DIAGNOSTICS AND STATISTICS    PAGE   11

                                                                    ASM 0100 15.00 01/03/72

NO STATEMENTS FLAGGED IN THIS ASSEMBLY
HIGHEST SEVERITY WAS    0
OPTIONS FOR THIS ASSEMBLY
  ALIGN, ALOGIC, BUFSIZE(STD), NODECK, ESD, FLAG(0), LINECOUNT(55), LIST, NOMCALL
  NOMLOGIC, NONUMBER, NOOBJECT, NORENT, RLD, NOSTMT, NOLIBMAC, NOTERMINAL, NOTEST, XREF
  SYSPARM()
WORK FILE BUFFER SIZE = 2558
TOTAL RECORDS READ FROM SYSTEM INPUT        189
TOTAL RECORDS READ FROM SYSTEM LIBRARY      833
TOTAL RECORDS PUNCHED                         0
TOTAL RECORDS PRINTED                       373
```

Figure 17. Assembler sample program
(Part 11 of 11)

Appendix C. Object Deck Output

ESD CARD FORMAT

Columns	Contents
1	12-2-9 punch
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, padded with blanks 1 byte -- ESD type code The HEX 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; otherwise blank 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 named TITLE statement. The name can be one to eight alphameric characters long. If the name is less than eight characters long or if there is no name, the remaining columns contain a card sequence number. (Coulmns 73-80 of cards produced by PUNCH or REPRO statements do not contain a deck ID or a sequence number.)

TXT CARD FORMAT

Columns	Contents
1	12-2-9 punch
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 named TITLE statement. The name can be one to eight alphameric characters long. If the name is less than eight 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.)

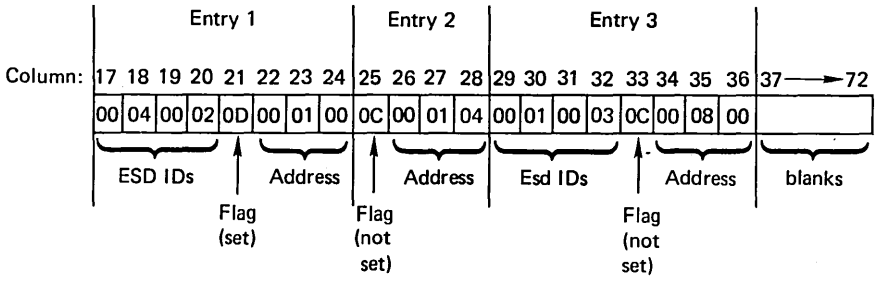
RLD CARD FORMAT

Columns	Contents
1	12-2-9 punch
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 named TITLE statement. The name can be one to eight alphameric characters long. If the name is less than eight 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:

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



END CARD FORMAT

Columns	Contents
1	12-2-9 punch
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	1 or 2
34-43	Order number of the assembler: 5741SC103
44-45	Version level of the assembler
46-47	Modification level of the assembler
48-49	Last two digits of the year in which the assembly was run
50-52	Day of the year (counted sequentially: Jan 3 = 3, Feb 3 = 34, etc) in which the assembly was run
53-72	Normally not used
73-80	Deck ID and/or sequence number. The deck ID is the name field from the first named TITLE statement. The name can be one to eight alphameric characters long. If there is no name or the name is less than eight characters long, 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.)

SYM CARD FORMAT

If you specify the TEST assembler option, the assembler punches out symbolic information concerning the assembled program. This output appears ahead of the object module. The format of the card images for SYM output is as follows:

Columns	Contents
1	12-2-9 punch
2-4	SYM
5-10	Blank
11-12	Variable field count -- 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 named TITLE statement. The name can be one to eight alphameric characters long. If the name is less than eight 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 fifty-six bytes of SYM text. The items making up the text are packed together; consequently, only the last card may contain less than fifty-six bytes of text in the variable field. The formats of a text card and an individual text item are shown in Figure 18. The contents of the fields within an individual entry are as follows:

1. Organization (one byte)
   Bit 0:
   0 = non-data type
   1 = data type

   Bits 1-3 (if non-data type):
   000 = space
   001 = control section
   010 = dummy control section
   011 = common
   100 = machine instruction
   101 = CCW
   110 = Simply relocatable EQU, named LTORG, named CNOP, or named ORG

   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 (three bytes) -- displacement from base of control section

3. Symbol Name (zero to eight bytes) -- symbolic name of particular item

Note: The follwing fields are present cnly for data-type items.

4. Data Type (one byte) -- contents in hexadecimal

   00 = C-type data
   04 = X-type data
   08 = B-type data
   10 = F-type data
   14 = H-type data
   18 = E-type data
   1C = D-type data
   20 = A-type or Q-type data
   24 = Y-type data
   28 = S-type data
   2C = V-type data
   30 = P-type data
   34 = Z-type data
   38 = L-type data

5. Length (two bytes for character, hexadecimal , or binary items; one byte for other types) -- length of data item minus 1

6. Multiplicity - M field (three bytes) -- equals 1 if not present

7. Scale - signed integer - S field (two bytes) -- present only for F, H, E, D, L, P and Z type data, and cnly if scale is ncn-zero.

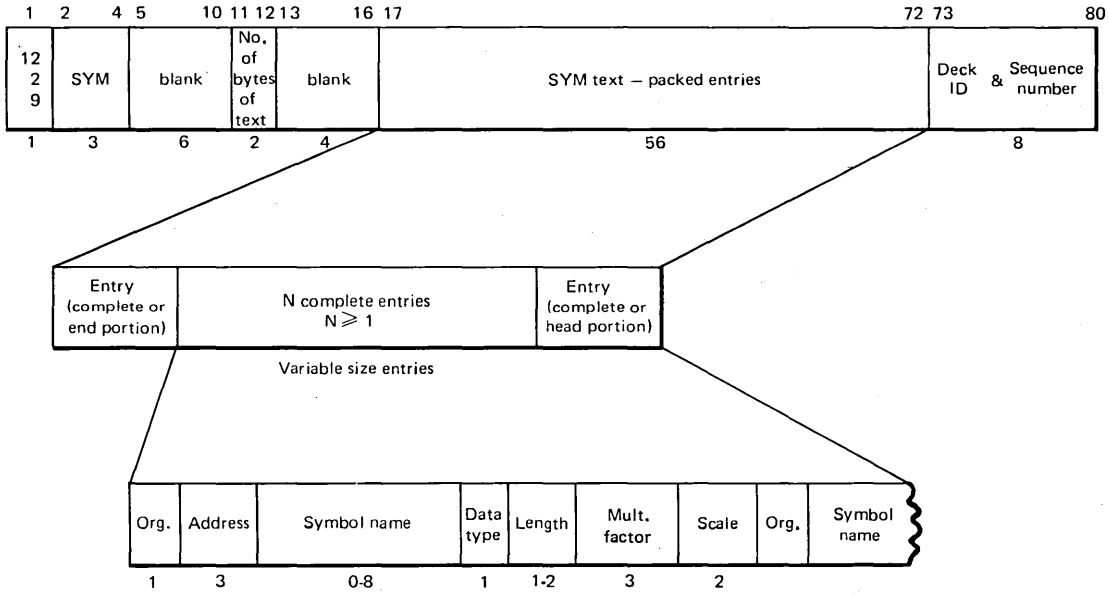


Figure 18. SYM Card Fcrmat

Appendix D. Dynamic Invocation of the Assembler

You can invcke the assembler from your problem program when it is executed, by using the CALL, LINK, XCTL, or ATTACH macrc instruction. If you use the XCTL instruction, you cannot specify any assembler options. The assembler will use the standard or default cpticns. If you use CALL, LINK, cr ATTACH, you can specify both the assembler options and DD names of the data sets to be used by the assembler. The formats of these macrcs are:

Name	Operation	Operand
[symbol]	CALL	IFOX00, (optionlist [,ddnamelist]), VL
	{ LINK / ATTACH }	EP=IFOX00, PARAM=(optionlist [,ddnamelist]), VL=1

EP -- specifies the symbolic name of the assembler (IFOX00).

PARAM -- specifies, as a sublist, address parameters tc 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 nc option list is prcvided.

The option list must begin on a halfword bcundary. The first two bytes contain a ccunt of the number of bytes in the remainder cf 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 ccmma. No blanks or zeros shculd appear in the list.

ddnamelist -- specifies the address of a variable length list containing alternate DDnames for the data sets used during assembler prccessing. If standard DDnames are used, this operand can 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 cf the list. Each name of less than eight bytes must be left-justified and padded with blanks. If an alternate DDname is cmitted, the standard name will be assumed. If the name is omitted within the list, the eight-byte entry must contain binary zeros. Names can be cmitted frcm the end merely by shcrtening the list. The sequence of the eight-byte entries in the DDname list is as follows:

Entry	Standard Name
1	not applicable
2	not applicable
3	not applicable
4	SYSLIB
5	SYSIN
6	SYSPRINT
7	SYSPUNCH
8	SYSUT1
9	SYSUT2
10	SYSUT3
11	SYSGO
12	SYSTERM

VL -- specifies that the high-order bit is to be set to 1 in the last word of the list of address parameters in the macro expansion. The assembler checks this bit to find out if a DDname list is specified or not.

Note: If you invoke the assembler more than once from the same program, make sure that RECFM=S is not specified for the SYSPRINT data set.

Appendix E. Assembler Data Sets and Storage Requirements

This appendix describes the data sets used by the assembler (see Figure 19). 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 or partition size or data set parameters (such as buffer size). A more detailed description of assembler storage requirements appears in OS/VS1 Storage Estimates, Order No. GC28-0604.

ASSEMBLER DATA SETS

DDname SYSUT1, SYSUT2, and SYSUT3

The assembler uses the utility data sets as intermediate external storage devices when processing the source program. These data sets must be organized sequentially, and the devices assigned to them must be direct access devices, magnetic tape units, or a combination of both. The assembler does not support multivolume utility data sets. For optimum performance, SYSUT1 should be on a direct access device.

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 that can 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 name being the macro instruction mnemonic or COPY code name.

The data set may be SYS1.MACLIB or a private macro library. SYS1.MACLIB contains macro definitions for the IBM-supplied macro instructions. Private libraries and SYS1.MACLIB can be concatenated with each other in any order. Concatenated libraries must have the same record length, but the blocking factors may be different. However, a library with a high blocking factor must always come before a library with a low blocking factor.

DDname SYSPRINT

This data set is used by the assembler to produce a listing. Output may be directed to a printer, magnetic tape, or direct-access storage device. The assembler uses the ASCII carriage-control characters for this data set. The smallest blocksize recommended is 1089 (blocking factor of 9).

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. This output can be used as input to the linkage editor.

DDname SYSGO

This is a direct-access storage device or 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 linkage editor.

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 ASCII carriage control characters for this data set. The smallest blocksize recommended is 1089 (blocking factor of 9).

ASSEMBLER VIRTUAL STORAGE REQUIREMENTS

The minimum virtual storage partition or region required by the assembler is 64K bytes. However, better performance is generally achieved if the assembler is run in 128K bytes of virtual storage. This region size is recommended and is specified in the assembler cataloged procedures.

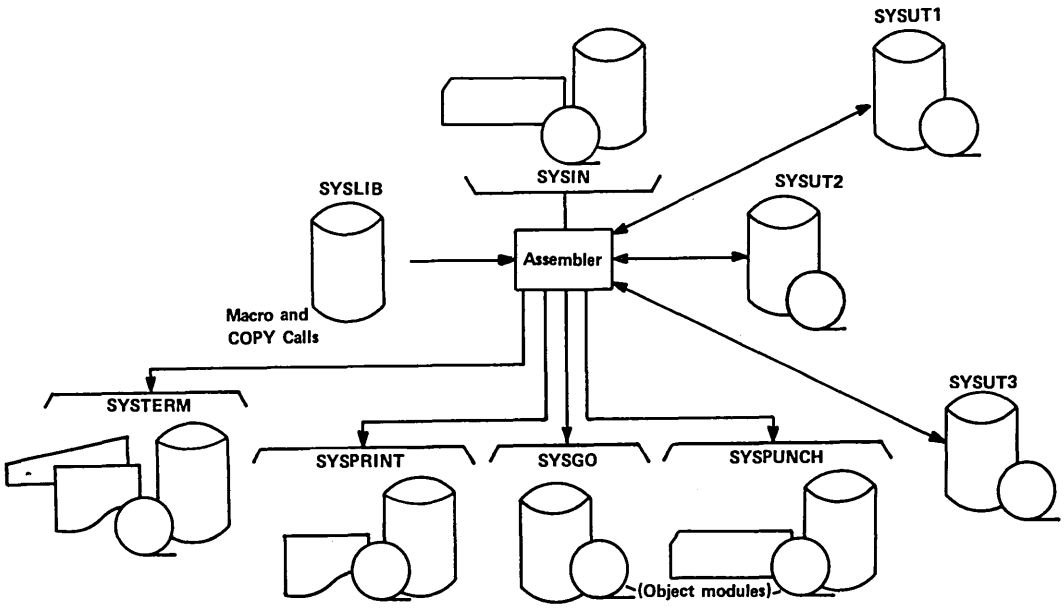
If more storage is allocated to the assembler, the size of buffers and work space can be increased. The amount of storage allocated to buffers and work space determines assembler speed and capacity. Generally, as more storage is allocated to buffers, a given assembly will run faster; as more storage is allocated to work space, larger and more complex macro definitions can be handled.

You can control the buffer sizes of SYSIN, SYSLIB, SYSPRINT, SYSPUNCH, and SYSGO by specifying the blocksize (BLKSIZE) and number of buffers (BUFNO) as shown in Figure 19.

You can control the buffer sizes for the assembler utility data sets (SYSUT1, SYSUT2, and SYSUT3) and the size of the work space used during macro processing, by specifying the BUFSIZE assembler option. Of the storage given to the assembler, the assembler first allocates storage for the SYSIN and SYSLIB buffers according to the specifications in the DD statements or the labels of the data sets. It then allocates storage for the modules of the assembler. The remainder of the partition or region is allocated to utility data set buffers and macro generation dictionaries according to the BUFSIZE option specified:

BUFSIZE(STD): 37% is allocated to buffers, and 63% to work space. This is the default chosen, if you do not specify any BUFSIZE option.

BUFSIZE(MIN): Each utility data set is allocated a single 790-byte buffer. The remaining storage is allocated to work space. This allows relatively complex macro definitions to be processed in a given region or partition size, but the speed of the assembly is substantially reduced.



	SYSIN	SYSLIB	SYSPRINT SYSTERM	SYSPUNCH	SYSGO	SYSUT1 SYSUT2 SYSUT3
LRECL	Fixed at 80	Fixed at 80	Fixed at 121	Fixed at 80	Fixed at 80	N/A
RECFM ①	You must specify in LABEL or DD card F,FS,FBS,FB, FBST, FBT	You must specify in LABEL or DD card F,FS,FBS,FB, FBST,FBT	F and A set by assembler. B set by assembler except when F is specified and BLKSIZE is not specified. You may add S or T FA,FAB,FAS,FAT FABS,FABT	F set by assembler, you may specify B and/or T in label or DD card F,FB,FT, FBT	F set by assembler, you may specify B and/or T in label or DD card F,FB,FT, FBT	Set by assembler to U
BLKSIZE ②	You must specify in LABEL or DD card, must be a multiple of LRECL	You must specify in LABEL or DD card, must be a multiple of LRECL	Optional, but must be a multiple of LRECL; if omitted BLKSIZE=LRECL	Optional, but must be a multiple of LRECL; if omitted BLKSIZE=LRECL	Optional, but must be a multiple LRECL; if omitted BLKSIZE=LRECL	If BUFSIZE (STD) in effect, a value between 790 and 8192 is chosen. If BUFSIZ E (MIN) in effect, 790 is chosen
BUFNO	Optional; if omitted 2 is used	Set by assembler to 1	Optional; if omitted 2 is used	Optional; if omitted 3 is used for unit record and 2 for other devices	Optional; if omitted 3 is used for unit record and 2 for other devices	Set by assembler to either 1 or 2

① U = undefined, F = fixed length records, B= blocked records, S= standard blocks, T = track overflow, A = ASCII code carriage control

② Blocking is not allowed on unit record devices. Blocking on other direct access can not be greater than the track size unless T is specified on RECFM. If the BLKSIZE specified is not a multiple of LRECL, the assembler truncates it to a multiple. For example, if LRECL = 80, a BLKSIZE of 850 is truncated to 800.

Figure 19. Assembler data set characteristics

Appendix F. The SYSTERM Listing

The SYSTERM data set, which gives you rapid access to the diagnostic messages issued during an assembly, is primarily designed for the user of the Time Sharing Option (TSO) of VS2. However, the data set can also be directed to a printer, a magnetic tape, or a direct-access device.

You use the assembler option TERMINAL to specify that you want a SYSTERM listing to be produced. Of course, you must also make sure that a DD statement describing the data set is included.

Each diagnosed statement in the assembly listing printed in the SYSTERM listing immediately followed by the messages that are issued for the statement. 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:

Line No.(s) (option NUM)	Statement No. (option STMT)	Source record(s) (Columns 1-72 of the source statement lines)

If a statement contains continuation lines, it will occupy several lines in the listing, each identified by a line number (if option NUMBER is used). If a statement in error is discovered during the expansion of a macro, or of any inner macro called by an outer macro, the first line of the outer macro instruction is listed before the flagged statement. If a statement is flagged during variable symbol substitution in open code, the first line of the model statement is listed as well as the generated statement.

Figures 20 and 21 illustrate the content and format of SYSTERM output. Figure 20 shows the source statement section of a SYSPRINT listing, and Figure 21 shows the SYSTERM listing produced during the same assembly. The example illustrates the rules given above. Options TERMINAL, NUMBER, and STMT were in effect during the assembly.

The SYSTERM listing starts with the statement ASSEMBLER DONE. At the end of the listing the following diagnostic information is given:

- NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY = nn
  (Indicates the total number of source statements in error)
- HIGHEST SEVERITY CODE WAS nn
  (Indicates the maximum severity code encountered)
- OPTIONS FOR THIS ASSEMBLY
  (Indicates the options in effect for this assembly)

```
                                                                                                                  PAGE    2

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                ASM 0100 14.59 01/03/72

                                      1          MACRO
                                      2          GENF  &P,&L
                                      3          LCLA  &K
                                      4 .LOOP    ANOP
                                      5 &K       SETA  &K+1
                                      6 &P&L(&K) DC    F'&L(&K)'
                                      7          AIF   (&K LT N'&L).LOOP
                                      8 .DONE    MEND
                                      9          GBLC  &Q
000000                               10 SAMPL2   CSECT
                                     11          SAVE  (14,12)          ALL REGS ARE SAVED IN SUPERVISOR SAVEAREA
000000                               12+         DS    0H                                                          00660000
000000 90EC D00C      0000C          13+         STM   14,12,12(13)                       SAVE REGISTERS           01180000
000004 05C0                          14          BALR  R12,0
                            00006    15          USING *,R12            SET UP BASE REGISTER
                                     16 &Q       SETC  'B'
000006 0000 0000      00000          17          L     R2,END           END OF AREA
        *** ERROR ***
00000A 0000 0000      00000          18          LA    R3,A             THIS IS A                                  *
                                                                        DUMMY COMMENT                              *
                                                                        TO SHOW                                    *
        *** ERROR ***
                                                                        A STATEMENT CONTAINING                     *
                                                                        TOO MANY CONTINUATION CARDS
00000E 5840 C022      00028          19          L     R4,F0            ZERO CONSTANT FOR RESETTING AREA
000012 5043 0000      00000          20 LOOP     ST    R4,0(R3)
000016 4130 3004      00004          21          LA    R3,4(,R3)        RESET AREA A
00001A 1923                          22          CR    R2,R3
00001C 4770 C00C      00012          23          BNE   LOOP
                                     24          AIF   ('A' EQ '&Q').GO
                                     25          SR    &Q,&Q            OPEN CODE MODEL STATEMENT                  *
                                                                        WITH CONTINUATION CARD
000020 0000                          26+         SR    B,B              OPEN CODE MODEL STATEMENT                  *
                                       +                                WITH CONTINUATION CARD
        *** ERROR ***
                                     27 .GO      RETURN (14,12)         EXIT FROM RTN
000022 98EC D00C      0000C          28+         LM    14,12,12(13)                       RESTORE THE REGISTERS    00260000
000026 07FE                          29+         BR    14                                 RETURN                   00800000
                                     30 *
                                     31 * CONSTANTS AND AREA ARE DELETED ON PURPOSE
                                     32 *
                                     33          GENF  F,0              GENERATION OF CONSTANTS
000028 00000000                      34+F0       DC    F'0'
                                     35          GENF  1,234            EXAMPLE OF MORE THAN ONE CARD              *
                                                                        IN A MACRO INSTRUCTION
00002C 000000EA                      36+1234     DC    F'234'
        *** ERROR ***
                            00002    37 R2       EQU   2
                            00003    38 R3       EQU   3
                            00004    39 R4       EQU   4
                            0000C    40 R12      EQU   12
                                     41          END
```

Figure 20. SYSPRINT listing of the source statements used to show SYSTERM output

```
ASSEMBLER (XF) DONE
               17          L     R2,END          END OF AREA
IFO188 END IS AN UNDEFINED SYMBOL
               18          LA    R3,A            THIS IS A                       *
                                                 DUMMY COMMENT                   *
                                                 TO SHOW                         *
IFO188 A IS AN UNDEFINED SYMBOL
IFO069 EXCESSIVE CONTINUATION CARDS, TWO ALLOWED
                                                 A STATEMENT CONTAINING          *
                                                 TOO MANY CONTINUATION CARDS
               25          SR    &Q,&Q           OPEN CODE MODEL STATEMENT       *
                                                 WITH CONTINUATION CARD
               26+         SR    B,B             OPEN CODE MODEL STATEMENT       *
                 +                               WITH CONTINUATION CARD
IFO188 B IS AN UNDEFINED SYMBOL
IFO188 B IS AN UNDEFINED SYMBOL
               35          GENF  1,234           EXAMPLE OF MORE THAN ONE CARD   *
               36+1234     DC    F'234'
IFO125 INVALID NAME- ILLEGAL EMBEDDED CHARACTER OR NON-ALPHABETIC FIRST CHARACTER
NUMBER OF STATEMENTS FLAGGED IN THIS ASSEMBLY =     4
HIGHEST SEVERITY WAS    8
OPTIONS FOR THIS ASSEMBLY
  ALIGN, ALOGIC, BUFSIZE(STD), NODECK, ESD, FLAG(0), LINECOUNT(55), LIST, NOMCALL
  NOMLOGIC, NUMBER, NOOBJECT, NORENT, RLD, STMT, NOLIBMAC, TERMINAL, NOTEST, XREF
  SYSPARM()
```

Figure 21. SYSTERM listing produced for the source statements shown in Figure 20.

Appendix G. Assembler Diagnostic Error Messages

This appendix lists all the diagnostic messages issued by the VS Assembler. The messages are listed sequentially by statement number.

HOW TO USE THIS SECTION

Once you have found an error message in the diagnostics section of your listing that you are not sure you understand fully, look up the entry for the message in this appendix. The entry for the message will give you the following items:

- The message number and the text of the message.
- Explanation of the message.
- Assembler action in response to the message.
- Programmer response to correct the error.
- Operator response to correct the error (only for certain messages).
- Severity code assigned to the message.

The following paragraphs describe the messages as they appear in your listing and explain in detail the various items of each entry in this appendix.

The Message Itself

In the diagnostics section of your assembler listing you will find the following items for each message:

- The number of the statement in error.
- The message identification number.
- The text of the message.

STATEMENT NUMBER: For certain messages the statement number given is always 0, either because the assembler cannot identify the number of the statement in which the error occurs when it finds the error, or because the error cannot be associated with a specific statement. For some of these messages, the text of the message identifies the macro in which the error is found.

For errors found during the editing of a library macro, the statement number given is that of the last numbered statement in the source module, unless the LIBMAC and MLOGIC assembler options are in effect, as described below under "Explanation".

MESSAGE NUMBER: The message identification number is a unique number consisting of the letters IFO followed by a three digit number.

TEXT: The text of the message is not always printed out in full in the diagnostics section of the listing. However, the corresponding text in this appendix is always fully printed out.

Certain messages include information in the message text to help you localize the error within the statement. In the message text as it appears in this section, 'nn' denotes a number and 'xxxxxxxx' a character string. The number identifies a column in the operand of the statement in error that is close to the column where the error is found.

The character string may represent a symbol or the word MACRO. It is limited to eight characters, so if the string containing the error is longer, it is truncated.

Explanation

This item gives the probable cause or causes of the error message. An error message is issued at the point where the assembler can no longer make sense of the text, not necessarily at the point where the real error occurred. For example, if you want to code the instruction LR 3,5, and leave out the R in the operation code, the assembler will treat the instruction as a storage-to-register instruction, and give an error message for the second operand (unless NOALIGN is specified).

If errors occur during the editing or expansion of a library macro and the assembler options specified cause the logic of the macro expansions not to be printed, error messages for the library macro will be logged against the last numbered statement in the program. However, if you use the LIBMAC and MLOGIC assembler options, errors in library macros will be logged against the statements in error. See the section "Assembler Options" for a discussion of these options.

Assembler Action

This item tells you how the assembler reacts to the error. A machine instruction usually causes zeros to be generated in its place in the object module if a major error occurs anywhere in that instruction. An assembler instruction is usually printed out but not processed ("processed as a comment"). Some machine and assembler instructions, however, are partially processed or processed with a default value. In some cases the assembler terminates the whole assembly.

Programmer Response

This item tells you how to correct the statement in error. It is assumed that you will detect certain errors when an error message draws your attention to the statement. Thus, the programmer response for each message does not tell you to check for keypunching errors or to check the use of the flagged statement.

Operator Response

This item tells the operator how to correct certain errors. The operator response is only given for messages that are printed on the operator's console. The operator will not change your source deck. He may, however, do such things as change partition or region size, or correct certain job control errors.

Severity Code

The severity code indicates the seriousness of the error. The severity codes used by the VS Assembler and their meanings are shown in the following table.

Severity Code	Explanation
4	Minor error; successful program execution is probable
8	Significant error; unsuccessful program execution is possible
12	Serious error; unsuccessful program execution is probable
16	Critical error; normal execution is impossible
20	Critical error; further assembly impossible, assembly terminated

The severity 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 greater than 8.

RECURRING ERRORS

If an error message recurs after the error situation has been corrected and there seems to be nothing wrong with the statement, there may be an error in the assembler. If you suspect that this is the case, make sure the program is correct and reassemble if necessary. If the problem still persists, do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- If a COPY statement was used, execute the IEBPTPCH utility to obtain a copy of the partitioned data set member specified in the COPY statement.
- Make sure that MSGLEVEL=(1,1) was specified in the JOB statement.

IFO000 UNDEFINED ERROR CODE IFOxxx

Explanation: An error code has been generated by the assembler for which no message has been defined. This is caused by a logical error in the assembler.

Assembler Action: Assembly continues.

Programmer Response: Perform the actions described under "Recurring Errors" above before calling IBM.

Severity Code: 16

IFO001 SYSTEM VARIABLE SYMBOL xxxxxxxx USED AS SYMBOLIC PARAMETER IN MACRO PROTOTYPE

Explanation: A variable symbol used as a symbolic parameter on a macro prototype statement has the same characters as a system variable symbol. The system variable symbols are:

&SYSECT	&SYSPARM
&SYSLIST	&SYSTIME
&SYSNDX	&SYSDATE

Assembler Action: Editing of the macro definition is terminated. All statements in the macro definition are processed as comments.

Programmer Response: Redefine the parameter with a variable symbol other than &SYSPARM, &SYSDATE, &SYSTIME, &SYSLIST, &SYSECT, or &SYSNDX.

Severity Code: 8

IFO002 SYMBOLIC PARAMETER xxxxxxxx IS DUPLICATED IN SAME MACRO PROTOTYPE

Explanation: Two identical symbolic parameters have been specified in the same macro prototype statement.

Assembler Action: Editing of the macro definition is terminated. All statements in the macro definition are processed as comments.

Programmer Response: Redefine one of the symbolic parameters with a variable symbol that is unique to that particular macro definition.

Severity Code: 8

IFO003 SYSTEM VARIABLE SYMBOL xxxxxxxx USED IN OPERAND OF GLOBAL OR LOCAL DECLARATION

Explanation: A system variable symbol has been used in the operand of a global or local declaration. The system variable symbols are:

&SYSECT	&SYSPARM

&SYSLIST &SYSTIME
&SYSNDX &SYSDATE

Assembler Action: The declaration conflicting with the system variable symbol is ignored. All subsequent references to the variable symbol in error are treated as references to the system variable symbol.

Programmer Response: Redefine the variable symbol using character combinations other than those listed above in the explanation.

Severity Code: 8

IFO004 GLOBAL OR LOCAL VARIABLE xxxxxxxx DUPLICATES A SYMBOLIC PARAMETER IN SAME MACRO DEFINITION

Explanation: A variable symbol that appears in the operand field of a global or local declaration is identical to a symbolic parameter defined on the macro prototype earlier in the macro definition.

Assembler Action: The declaration conflicting with the symbolic parameter is ignored. All subsequent references to it are treated as references to the symbolic parameter that it duplicates.

Programmer Response: Redefine the global or local variable with a variable symbol that is unique to the macro definition.

Severity Code: 8

IFO005 GLOBAL OR LOCAL VARIABLE SYMBOL xxxxxxxx DUPLICATES PREVIOUS DECLARATION

Explanation: A global or local variable symbol was declared twice in the same macro definition or in open code.

Assembler Action: The second declaration of the variable symbol is ignored. All subsequent references to it are treated as references to the first declaration.

Programmer Response: If the second declaration is LCLx, redeclare it using a variable symbol unique to the macro definition or to open code. If the second declaration is GBLx, redeclare it as for LCLx, but be sure that all declarations of that global variable elsewhere in the program are identical.

Severity Code: 8

IFO006 UNDEFINED VARIABLE SYMBOL xxxxxxxx

Explanation: A variable symbol has been referenced in this statement that is not a system variable symbol; has not been defined within the macro definition as a symbolic parameter, a local variable, or a global variable; or has not been defined in open code as a local or global variable.

Assembler Action: The statement is processed as a comment, unless the error has occurred in a macro instruction parameter. If the macro instruction parameter contains an undefined variable symbol, the parameter is assigned the value of a null string.

Programmer Response: Define the variable symbol as a symbolic parameter, a local variable, or a global variable; or, if desired, reference a previously-defined variable symbol of the appropriate type. This message may be issued if an ampersand erroneously appears as the first character of an ordinary symbol, and thus creates an unintended variable symbol.

Severity Code: 8

IFO007 USAGE OF xxxxxxxx IS INCONSISTENT WITH ITS DECLARATION

Explanation: A global or local variable symbol was defined as dimensioned but was used without a subscript, or a global or local variable symbol was defined as undimensioned but was used with a subscript.

Assembler Action: Editing of the statement that contains the inconsistent usage is terminated, and the statement is processed as a comment.

Programmer Response: Make the usage of the SET symbol consistent with its global or local declaration, or make the declaration of the SET symbol consistent with its usage.

Severity Code: 8

IFO008 CIRCULAR OPSYN STATEMENTS

Explanation: The assignment of a synonym in the operand field of an OPSYN statement to the established mnemonic in the name field results in the mnemonic being its own synonym. For example:

```
ADD     OPSYN A
PLUS          OPSYN ADD
XYZ     OPSYN PLUS
ADD     OPSYN XYZ
```

The final OPSYN statement in the above sequence is flagged.

Assembler Action: The flagged OPSYN statement is processed as a comment.

Programmer Response: Remove any OPSYN statement that results in a circular definition, or alter such an OPSYN statement by respecifying the synonym or the mnemonic.

Severity Code: 8

IFO009 EDIT DICTIONARY SPACE EXHAUSTED

Explanation: The work space available is not sufficient to contain the dictionaries that are required to edit the macro definition or open code.

Assembler Action: If a macro definition is being edited, the remaining statements up to the MEND statement are processed as comments, and editing resumes. If open code is being edited, the remaining statements up to the end-of-file are processed as comments.

Programmer Response: Increase the size of the region or partition that is allocated to assembly, or allocate more dictionary space via the BUFSIZE assembler option. See Appendix E of this manual.

Severity Code: 12

IFO010 SOURCE MACRO xxxxxxxx HAS BEEN PREVIOUSLY DEFINED

Explanation: The mnemonic in the macro instruction prototype of a source macro duplicates a mnemonic already defined as a source macro.

Assembler Action: All statements in this macro definition are processed as comments. All subsequent references to the mnemonic are treated as references to the first definition associated with that op code.

Programmer Response: Provide a unique mnemonic op code for the flagged macro prototype.

Severity Code: 8

IFO012 ICTL OR OPSYN STATEMENT APPEARS TOO LATE IN THE PROGRAM

Explanation:

- The ICTL statement does not precede all other statements in the source module; or
- The OPSYN statement does not appear before source macro definitions and open code statements. The only statements that can precede an OPSYN statement are: ICTL, ISEQ, TITLE, PRINT, EJECT, SPACE, OPSYN, COPY (unless the member copied contains any other than the statements listed here), and comments statements.

Assembler Action: The ICTL or OPSYN statement is processed as a comment.

Programmer Response: Place the ICTL or OPSYN statement at the beginning of your program as described in the explanation above.

Severity Code: 8

IFO013 OPSYN NAME FIELD NOT ORDINARY SYMBOL, OR OPSYN OPERAND FIELD NOT ORDINARY SYMBOL OR BLANK

Explanation: The name or operand field of an OPSYN instruction contains more than 8 alphanumeric characters or does not begin with an alphabetic character.

Assembler Action: The OPSYN statement is processed as a comment.

Programmer Response: Correct the invalid name field or operand field.

Severity Code: 8

IFO014 INVALID OPCODE IN OPSYN OPERAND OR NAME FIELD

Explanation:

- The name field of an OPSYN instruction with a blank operand field does not specify a machine instruction operation code, an extended machine instruction operation code, or an assembler operation code; or
- The operand field of an OPSYN instruction does not specify a machine instruction operation code, an extended machine instruction operation code, or an assembler operation code.

Assembler Action: The OPSYN statement is treated a comment.

Programmer Response: Make sure that the name field contains a valid operation code, or supply a valid operation code in the operand.

Severity Code: 8

IFO016 ILLEGAL OR INVALID NAME FIELD

Explanation: One of the following errors was detected.

- No name was found where one is required.
- A name was supplied where none is allowed.
- An invalid character was found in the name field.

Assembler Action: The statement is processed as a comment, unless the error has occurred in the name field of a macro instruction. If the macro name field parameter contains an error, the parameter is assigned the value of a null string.

Programmer Response: Supply a name if one is required, omit the name if one is not allowed, or correct the invalid character.

Severity Code: 8

IF0017 .* COMMENTS STATEMENT IS ILLEGAL OUTSIDE MACRO DEFINITION

Explanation: An internal macro comments statement (.*) appears outside macro definitions (in open code).

Assembler Action: The statement is printed.

Programmer Response: Remove the .* comments statement. If you want a comment, put an * in the begin column and follow it by the comment.

Severity Code: 4

IF0018 MORE THAN 5 ERRORS IN THIS STATEMENT, ERROR ANALYS OF THE STATEMENT IS TERMINATED

Explanation: The maximum number of error messages issued during editing to each statement is 5. The sixth error causes this message.

Assembler Action: Error analysis for this statement is terminated.

Programmer Response: Correct the indicated errors and reassemble. Any additional errors on this statement will be detected in the next assembly.

Severity Code: 4

IF0019 INVALID OPERAND IN ICTL OR ISEQ STATEMENT

Explanation:

(1) The value of one or more operands in an ICTL statement is incorrect. The begin column must be within columns 1 to 40; the end column must be within columns 41 to 80 and at least 5 columns away from the begin column; and the continue column must be within columns 2 to 40.

(2) One of the following errors has occurred in an ISEQ statement:

- The operand has an illegal range; the operand value cannot fall between the begin and end columns, and the second operand must not be less than the first.
- The operand field is invalid. The operand field must contain two valid decimal self-defining terms, separated by a comma or be blank.

Assembler Action: If a program contains an ICTL error, the whole program is processed as comments. If one of the ISEQ errors has occurred, no sequence checking is performed.

Programmer Response: Supply valid operand(s).

Severity Code: 8

IFO021 INVALID TERM IN OPERAND

Explanation: An invalid term has been used in an expression of the operand.

Assembler Action: The statement is processed as a comment.

Programmer Response: Make sure the operand is a character relation, an arithmetic relation, a logical relation, a SETx symbol, a symbolic parameter, or a decimal self-defining term.

Severity Code: 8

IFO022 ICTL STATEMENT IS ILLEGAL IN COPY CODE

Explanation: An ICTL statement appears in code that is inserted in the program by a COPY instruction.

Assembler Action: The ICTL statement is processed as a comment.

Programmer Response: Make sure the ICTL instruction is not in code inserted by the COPY instruction. If used, the ICTL instruction must always be the first instruction in your source module.

Severity Code: 8

IFO023 ILLEGAL MACRO, MEND, OR MEXIT STATEMENT - MAY APPEAR ONLY WITHIN MACRO DEFINITIONS

Explanation: MACRO, MEND, or MEXIT statements are not allowed in open code. They can be used only in macro definitions. This message will be issued if an instruction other than ICTL, ISEQ, OPSYN, TITLE, PRINT, EJECT, SPACE, or COPY appears before any macro definitions in your program. Of course, any such COPY instruction cannot copy any other statements than ISEQ, OPSYN, TITLE, PRINT, EJECT, or SPACE. This message will also be issued, if an undefined operation code appears before your macro definitions.

Assembler Action: The illegal MACRO, MEND, or MEXIT statement is processed as a comment.

Programmer Response: Remove the statement from open code on place it within a macro definition. Make sure that all your macro definitions are placed at the beginning, before open code.

Severity Code: 8

IFO024 UNPAIRED PARENS, OR BLANK FOUND INSIDE PAIRED PARENS

Explanation:

- Unpaired parentheses appear in the operand field; or
- A blank appears inside paired parentheses in the operand field of a macro instruction. This may be an error in sublist structure; or

- A blank appears inside parentheses of an arithmetic expression; or
- A term is missing in a logical expression.

Assembler Action: The operand in error is ignored.

Programmer Response: If unpaired parentheses appear, be sure that there is a right parenthesis for every left parenthesis. Remove illegal blanks inside paired parentheses.

Severity Code: 8

IFO025 STATEMENT OUT OF SEQUENCE

Explanation: The input sequence checking specified by the ISEQ instruction has determined that the flagged statement id out of sequence.

Assembler Action: The statement is flagged and assembled, however, the sequence number of the following statements will be checked relative to this statement and not relative to the sequence of previous statements.

Programmer Response: Put the statement in the proper sequence.

Severity Code: 4

IFO026 CHARACTERS APPEAR BETWEEN THE BEGIN AND CONTINUE COLUMNS ON CONTINUATION CARD

Explanation: On a continuation card, the begin column and all columns between the begin column and the continue column (usually column 16) must be blank.

Assembler Action: Characters that appear between the begin column and the continue column are ignored.

Programmer Response: Determine whether the operand started in the wrong continue column or whether the preceding card contained an erroneous continue punch in column 72.

Severity Code: 4

IFO027 ICTL, ISEQ, MACRO, OR OPSYN STATEMENT APPEARS IN MACRO DEFINITION

Explanation: One of the specified operations is used within a macro definition, which is illegal.

Assembler Action: The illegal operation is ignored and the statement is processed as a comment.

Programmer Response: Remove all ICTL, ISEQ, MACRO, and OPSYN statements from within macro definitions. Make sure your ICTL and OPSYN instructions precede your macro definitions, and that each macro definition ends with a MEND statement.

Severity Code: 8

IFC028 ILLEGAL PROTOTYPE KEYWORD PARAMETER DEFAULT VALUE

Explanation: A variable symbol is used as the default value of a keyword parameter.

Assembler Action: The statement is ignored.

Programmer Response: Supply a valid default value for the keyword parameter.

Severity Code: 8

IFC029 xxxxxxxx IS AN ILLEGAL OPERAND IN A GLOBAL OR LOCAL DECLARATION

Explanation: In a global (GBLx) or local (LCLx) SET symbol declaration, the indicated operand does not consist of one or more variable symbols that are separated by commas and terminated with a blank.

Assembler Action: The attempted global or local SET symbol declaration is processed as a comment. Recovery is made in certain circumstances and some valid variable symbols in the declaration are recognized and defined correctly.

Programmer Response: Supply the operand with valid variable symbols and delimiters. Check all global and local declarations.

Severity Code: 8

IFC030 DECLARED DIMENSION OF xxxxxxxx IS ILLEGAL

Explanation: The declared dimension, which appears in the error message, must be a nonzero, unsigned decimal integer, not greater than 32,767, and enclosed in parentheses.

Assembler Action: If the declared dimension was a decimal self-defining term greater than 32,767, a default dimension of 32,767 is assigned to the variable symbol. In all other cases, the variable symbol declaration is ignored.

Programmer Response: Supply a valid dimension.

Severity Code: 8

IFC031 SET STATEMENT NAME NOT A VARIABLE SYMBOL, OR SET STATEMENT NAME INCONSISTENT WITH DECLARED TYPE

Explanation: (1) The name field of a SET statement does not consist of an ampersand followed by from 1 to 7 alphameric characters, the first of which is alphabetic.

(2) The symbol does not match its previously declared type. For instance, the symbol might have been previously defined as LCLA, but the flagged statement may have tried to assign a SETC character string to it.

(3) A system variable symbol appears in the name field of a SETx instruction. The system variable symbols are &SYSECT, &SYSLIST, &SYSNDX, &SYSPARM, &SYSDATE, and &SYSTIME.

Assembler Action: The flagged statement is processed as a comment.

Programmer Response: Assign a valid variable symbol to the name field of the SET statement (the symbol must be previously defined as a global or local variable), or be sure that the usage of the symbol corresponds to its previously declared type.

Severity Code: 8

IFO032 xxxxxxxx APPEARS IMPROPERLY IN THE OPERAND OF THIS STATEMENT

Explanation: The specified operand part is invalid.

Assembler Action: The statement is processed as a comment.

Programmer Response: Check the syntax required for the operand field of this statement, and supply a valid operand.

Severity Code: 8

IFO033 xxxxxxxx IS AN INVALID LOGICAL OPERATOR

Explanation: The specified character string was found where a logical operator (AND or OR) was expected.

Assembler Action: The statement is processed as a comment.

Programmer Response: Use either AND or OR, as appropriate, for the logical operator.

Severity Code: 8

IFO035 QUOTES NOT PAIRED, OR ILLEGAL TERMINATION OF QUOTE STRING

Explanation: The quotes in the operand field of this statement are unpaired, or the string is illegally terminated.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply any missing quotes.

Severity Code: 8

IFC036 ATTRIBUTE REFERENCE FOR xxxxxxxx IS INVALID

Explanation: The flagged statement has attempted to reference a symbol that is not a valid ordinary or variable symbol. The attributes referenced were one or more of the following: type (T'), length (L'), scaling (S'), integer (I'), count (K'), and number (N').

Assembler Action: The attribute referenced is ignored, and/or the statement is ignored, and/or default values for type, length, and scaling attributes are supplied.

Programmer Response: Determine if a clerical error was made in coding either the reference or the definition of the symbol that appears in the message text; or supply a valid ordinary or variable symbol where necessary.

Severity Code: 8

IFO037 xxxxxxxx IS AN ILLEGAL SUBSCRIPT

Explanation: The subscript that appears in the message text either is not enclosed by paired parentheses, or is an illegal subscript.

Assembler Action: The statement that contains the illegal subscript is processed as a comment.

Programmer Response: Be sure the parentheses are paired, and that a valid subscript appears inside them.

Severity Code: 8

IFO038 xxxxxxxx IS AN INVALID SELF-DEFINING TERM

Explanation: The characters specified in the message are invalid in the operand field of a binary (type B), character (type C), decimal, or hexadecimal (type X) self-defining term.

Assembler Action: The statement that contains the invalid self-defining term is processed as a comment.

Programmer Response: Make sure that the characters used for a self-defining term are consistent with the type of term.

Severity Code: 8

IFO039 xxxxxxxx IS AN INVALID VARIABLE SYMBOL

Explanation: The specified symbol does not consist of an ampersand followed by from 1 to 7 alphameric characters, the first of which is alphabetic.

Assembler Action: The statement that contains the invalid variable symbol is processed as a comment. If the statement is a macro prototype statement, all statements in the macro definition are treated as comments.

Programmer Response: Supply a valid variable symbol, or check

that a single ampersand is not used where a double ampersand is needed.

Severity Code: 8

IFO042 PARAMETER IN MACRO PROTOTYPE OR MACRO INSTRUCTION EXCEEDS 255 CHARACTERS

Explanation: A parameter value that appears in the operand field of either a macro prototype or a macro instruction exceeds 255 characters in length.

Assembler Action: The first 255 characters of the parameter are deleted. The remaining characters are used as the parameter value.

Programmer Response: Limit the parameter to 255 characters or separate it into two or more parameters.

Severity Code: 8

IFO043 MACRO INSTRUCTION PROTOTYPE STATEMENT HAS INVALID OP CODE

Explanation:

- The operation code of a macro prototype statement is previously defined as the operation code of a machine, assembler, or macro instruction; or
- The operation code of a macro prototype statement is not a valid ordinary symbol; that is, it does not consist of a letter, followed by 0 to 7 letters or digits or both.

Assembler Action: The entire macro definition is processed as comments.

Programmer Response: Supply a valid ordinary symbol that does not conflict with any machine, assembler, or macro instruction operation code.

Severity Code: 8

IFO046 STATEMENT COMPLEXITY EXCEEDED

Explanation: The expression evaluation work area has overflowed because the expression is too complex. The complexity of an expression is determined by the number of nested operators and levels of parentheses. Up to 35 operators and levels of parentheses are allowed. For logical expressions, this total allows 18 unary and binary operators, and 17 levels of parentheses. For arithmetic expressions in conditional assembly, the total allows 24 unary and binary operators, and 11 levels of parentheses.

Assembler Action: The statement is processed as a comment.

Programmer Response: Simplify the expression to the limits described in the explanation.

Severity Code: 8

IFC047 UNEXPECTED END OF FILE ON SYSTEM INPUT (SYSIN)

Explanation:

- A continuation record was expected when an end-of-file occurred on SYSIN (the source program ended); or
- End-of-file immediately follows a REPRO statement; or
- End-of-file occurs before an END card has been read.

Assembler Action: An END statement is generated and assembly continues.

Programmer Response: Determine if any statements were omitted from the source program.

Severity Code: 4

IFO048 ICTL STATEMENT HAS NO OPERAND

Explanation: The ICTL statement requires an operand, but none is present.

Assembler Action: The entire source module is processed as comments.

Programmer Response: Supply from 1 to 3 decimal self-defining terms to indicate respectively the begin, end, and continue columns. If the ICTL statement is omitted, columns 1, 71, and 16, respectively, are the default values.

Severity Code: 8

IFO049 COPY STATEMENT OPERAND NOT A VALID ORDINARY SYMBOL

Explanation: The operand of a COPY statement is not a symbol of 1 to 8 alphameric characters, the first of which is alphabetic.

Assembler Action: The COPY request is processed as a comment.

Programmer Response: Supply a valid ordinary symbol in the operand field.

Severity Code: 8

IFO050 COPY STATEMENT DOES NOT HAVE AN OPERAND

Explanation: No operand found on this COPY statement.

Assembler Action: The statement is processed as a comment.

Programmer Response: Place the name of a member to be copied in the operand field, or remove the COPY statement.

Severity Code: 8

IFC051 UNEXPECTED END OF DATA ON SYSTEM LIBRARY (SYSLIB)

Explanation: An end-of-file occurred on the input from a system library before a MEND statement terminating a macro definition was encountered.

Assembler Action: The missing MEND statement is generated.

Programmer Response: Determine if the MEND statement was omitted from the library macro, or if the library contains an otherwise incomplete macro definition, or if a macro call has been made to a non-macro definition.

Severity Code: 4

IFC052 UNARY OPERATOR NOT A PLUS OR MINUS SIGN

Explanation: An operator other than a plus or minus sign appears as a unary operator. Except for unary operators, which are limited to plus and minus signs, only one operator can appear between two terms.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply the missing term or a correct operator.

Severity Code: 8

IFC053 CP CODE NOT FOUND ON FIRST OR ONLY CARD

Explanation: The complete statement name (if one is used) and the operation code, each followed by a blank, do not appear before the continuation indicator column on the first card of a continued statement.

Assembler Action: The entire statement is processed as a comment.

Programmer Response: Make sure that both the name and operation code of the statement appear on the first card. Check for syntactic errors.

Severity Code: 8

IFC054 INVALID OPERATION CODE

Explanation:

- The operation code specified is not a valid ordinary symbol; or
- A variable symbol in the operation field is invalid; or
- The resulting operation code after substitution with or without concatenation is not a valid ordinary symbol.

Assembler Action: The statement is processed as a comment.

Programmer Response: Make sure that ordinary or variable symbols used in the operation field are valid. If you use variable symbols with or without concatenation, make sure the resulting symbol is a valid ordinary symbol.

Severity Code: 8

IFO055 MEND STATEMENT GENERATED

Explanation: An end-of-file occurred on the input from the system input device (SYSIN) or the system library (SYSLIB) before a MEND statement terminating a macro definition was encountered.

Assembler Action: A MEND statement is generated.

Programmer Response: Supply a MEND statement to terminate the macro definition.

Severity Code: 8

IFO057 DUPLICATION FACTOR xxxxxxxx IN SETC EXPRESSION NOT TERMINATED BY A RIGHT PARENTHESIS

Explanation: A SETC operand begins with a left parenthesis, but a comma, a period, or a blank appears before the closing right parenthesis.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply a right parenthesis.

Severity Code: 8

IFO058 NO ENDING QUOTE ON SETC EXPRESSION

Explanation: The character expression in the operand field of a SETC statement must be enclosed in quotes. The statement ends before a delimiting quote.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply any missing quotes.

Severity Code: 8

IFO059 INVALID TERM IN LOGICAL EXPRESSION

Explanation: One of the terms in the logical expression is invalid in the context.

Assembler Action: The statement is processed as a comment.

Programmer Response: Make sure that the terms in the logical expression are valid.

Severity Code: 8

IFO060 END STATEMENT GENERATED

Explanation: One of two errors occurred.

(1) End-of-file occurred on the system input device (SYSIN) before an END card was read.

(2) The ACTR limit was exceeded in open code.

Assembler Action: An END statement is generated.

Programmer Response:

(1) Supply a valid END statement; or

(2) Either correct the conditional assembly loop in open code so that the ACTR limit is not exceeded, or set the ACTR limit in open code to a higher value.

Severity Code: 4

IFC061 COPY NEST GREATER THAN FIVE

Explanation: The maximum limit of five nested levels of COPY statements is exceeded.

Assembler Action: COPY processing terminates.

Programmer Response: Eliminate excessive levels of COPY statements.

Severity Code: 8

IFO062 REQUIRED OPERAND FIELD MISSING

Explanation: This statement requires an operand in the operand field and none is present.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply the missing operand.

Severity Code: 8

IFO064 INTERLUDE DICTIONARY SPACE EXHAUSTED

Explanation: The work space available is not sufficient to contain the dictionaries required to build either

(1) The skeleton dictionary for a macro definition or all of open code, or

(2) The ordinary symbol attribute reference dictionary.

This message is always logged against statement number 0.

Assembler Action: If a macro is being processed, building of the skeleton dictionary for that macro definition is terminated and the macro will not be expanded. If open code is being processed, the building of the open code skeleton dictionary is terminated and the program is processed as comments. If space for the ordinary symbol attribute reference dictionary is exhausted, the building of it is abandoned.

Programmer Response: Within the partition, increase the size of the region that is allocated to assembly, or allocate more of the partition to dictionary space via the BUFSIZE assembler option (see Appendix E).

Severity Code: 12

IFO065 EXPRESSION 2 OF EQU SYMBOL xxxxxxxx NOT IN RANGE 0-65535

Explanation: The value of the expression specified in the second operand of the EQU instruction where this symbol is defined is not in the range 0-65535.

This message is always logged against statement number 0.

Assembler Action: The length attribute of the symbol is set to 1

Programmer Response: Make sure the value of the second operand of the EQU instruction is in the range 0-65535, or delete the second operand.

Severity Code: 8

IFO066 EXPRESSION 3 OF EQU SYMBOL xxxxxxxx NOT IN RANGE 0-255

Explanation: The value of the expression specified in the third operand of the EQU instruction where this symbol is defined is not in the range 0-255.

This message is always logged against statement number 0.

Assembler Action: The type attribute of the symbol is set to U.

Programmer Response: Make sure the value of the third operand of the EQU instruction is in the range 0-255, or delete the third operand.

Severity Code: 8

IF0067 DECLARED DIMENSION FOR GLCBAL VARIAELE xxxxxxxx IN xxxxxxxx xxxxxxxx IS INCONSISTENT

Explanatien: The declared dimension of a global variable defined in a macro definition or in open code is not consistent with the declared dimension of the same glcbal variable in anether macre definition or in open code.

This message is always logged against statement number 0. The message text identifies the macro (or open code) where the error is found.

Assembler Actien: All references to the glcbal variable in the macre definitien or in open code where the inccnsistency was detected result in a null (zero) value.

Programmer Response: Be sure that all definitions of a given global variable have the same declared dimension.

Severity Code: 4

IFO068 COPY MEMBER xxxxxxxx NCT FCUND IN LIBRARY

Explanation: The COPY member shcwn in the message text was not found in the library.

Assembler Actien: The CCPY statement is processed as a comment.

Prcgrammer Response: Determine whether the library member name is misspelled or whether an incorrect member name was referenced. Make sure the proper macre library is assigned in your JCL statements.

Severity Code: 8

IFO069 TOO MANY CONTINUATION CARDS, TWC ALLOWED

Explanation: Only two continuatien cards are allowed for each statement, except for macro definition prototype and macro call statements.

Assembler Actien: Excess continuatien cards are prccessed as comments.

Programmer Response: Restructure the statement so that it can be contained on a total of three cards. Extensive remarks may be recorded as comment statements by ccding an asterisk in cclumn 1 and eliminating the continuation indicators.

Severity Code: 4

IFO070 SUBSTRING NOTATICN IS NCT DELIMITED BY COMMA OR RIGHT PARENTHESIS

Explanation: Two SETA expressions used in substring nctation are not separated by a comma or enclosed in parentheses.

Assembler Actien: The statement is processed as a comment.

Programmer Response: Supply the missing delimiter, or check for other syntax errors that make this appear as substring notation.

Severity Code: 8

IFO073 AGO OR AIF OPERAND NOT A SEQUENCE SYMBOL

Explanation: The symbol in the operand field of an AIF or AGO statement is not a period (.) followed by from 1 to 7 alphameric characters, the first of which is alphabetic.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply a valid sequence symbol.

Severity Code: 8

IFO074 SEQUENCE SYMBOL xxxxxxxx IS MULTIPLY DEFINED IN xxxxxxxx xxxxxxxx

Explanation: The sequence symbol in the name field has been used in the name field of a previous statement within the same macro definition or open code.

This message is always logged against statement number 0. The message text identifies the macro (or open code) where the error is found.

Assembler Action: All definitions of the sequence symbol after the first one are ignored. All references to the sequence symbol are treated as references to the first definition.

Programmer Response: Provide unique sequence symbols for the macro definition or open code.

Severity Code: 4

IFO076 SEQUENCE SYMBOL xxxxxxxx IS UNDEFINED IN xxxxxxxx xxxxxxxx

Explanation: A sequence symbol appears in the operand of an AIF or AGO statement, but does not appear in the name field of another statement in the same macro definition or open code.

This message is always logged against statement number 0. The message text identifies the macro (or open code) where the error is found.

Assembler Action: All statements which reference the undefined sequence symbol are processed as comments.

Programmer Response: Define the sequence symbol at the appropriate point, or reference a sequence symbol that is already defined.

Severity Code: 4

IFO078 UNDEFINED OP CODE

Explanation: The mnemonic operation code of this statement does not correspond to any of the following:
- a machine instruction operation code
- an extended machine instruction operation code
- an assembler instruction operation code
- a macro instruction operation code
- an operation code that has been defined by an OPSYN instruction.

This message is also issued for operation codes that have been deleted by OPSYN instructions.

Assembler Action: The statement is treated as a comment. If the statement appears before open code, all statements following it are considered to belong to open code. This means that any macro definitions following the error are treated as errors.

Programmer Response: Either make sure you use a valid mnemonic operation code, or make sure that the proper OPSYN instructions are included in your program.

Severity Code: 8

IFO080 ATTRIBUTE REFERENCE TO UNDEFINED SYMBOL

Explanation: The symbol specified in a length (L'), scaling (S'), or integer (I') attribute reference is either an undefined symbol or a symbolic parameter (or a &SYSLIST specification) representing an undefined symbol.

Assembler Action:

- The length attribute, if specified, is set to 1.
- The integer or scaling attribute, if specified, is set to 0.

Programmer Response: Make sure the symbol is defined.

Severity Code: 4

IFO081 DECLARED TYPE FOR GLOBAL VARIABLE xxxxxxxx IN xxxxxxxx xxxxxxxx IS INCONSISTENT

Explanation: The type (GBLA, GBLB, or GBLC) of a global variable declared in a macro definition or in open code is not consistent with the type of the same global variable declared in another macro definition or in open code.

This message is always logged against statement number 0. The message text identifies the macro (or open code) where the error is found.

Assembler Action: All references to the global variable in the macro definition or in open code where the inconsistency was detected result in a null (zero) value.

Programmer Response: Make all declarations of the same global variable consistent.

Severity Code: 4

IFO085 MACRO HEADER MISSING, MACRO NOT EXPANDABLE

Explanation: The first statement of a library macro definition was not a MACRO statement, and the search for the macro definition is terminated.

Assembler Action: The macro call is processed as a comment.

Programmer Response: Be sure that the library macro definition begins with a MACRO statement.

Severity Code: 8

IFO087 INVALID MACRO DEFINITION PROTOTYPE, MACRO NOT EXPANDABLE

Explanation: A comment statement appears immediately after a macro header (MACRO statement).

Assembler Action: All the statements of the macro definition are processed as comments.

Programmer Response: Make sure that the statement immediately following the macro header is a macro prototype statement. No comments or any other statements are permitted between the macro header and the prototype of a macro definition.

Severity Code: 8

IFO088 LIBRARY MACRO PROTOTYPE DOES NOT MATCH MEMBER NAME, MACRO NOT EXPANDABLE.

Explanation: The mnemonic operation code in the macro prototype in a library macro definition does not match the entry in the macro library.

Assembler Action: The macro instruction is processed as a comment.

Programmer Response: Enter the macro definition in the library under the same name as the mnemonic op code that appears on the macro prototype.

Severity Code: 8

IFO089 GENERATION-TIME DICTIONARY SPACE EXHAUSTED

Explanation: The workspace available is not sufficient to contain the dictionaries required to expand the macro, to extend a SETC variable, or to contain the basic global dictionaries.

Assembler Action: If the global dictionary workspace is insufficient, the text is processed as comments. If there is insufficient space to extend the SETC variable, expansion of the macro that contains the variable is terminated. If the space for macro definition dictionaries is insufficient, calls

to those macros are not expanded.

Programmer Response: Within the partition, increase the size of the region that is allocated to assembly, or allocate more of the partition to dictionary space via the BUFSIZE assembler option (see Appendix E).

Severity Code: 12

IFO090 UNDEFINED SEQUENCE SYMBOL ENCOUNTERED DURING CONDITIONAL ASSEMBLY

Explanation: A sequence symbol referenced in the operand field of this statement is undefined in the macro definition or open code. This statement has been encountered during conditional assembly.

Assembler Action: The statement is processed as a comment.

Programmer Response: Define the sequence symbol at an appropriate point, or reference a sequence symbol that is already defined.

Severity Code: 8

IFO091 KEYWORD PARAMETER xxxxxxxx IS DUPLICATED ON SAME MACRO CALL

Explanation: A keyword parameter has appeared more than once on the same macro instruction.

Assembler Action: The last value assigned to the parameter is used, the other value(s) are ignored.

Programmer Response: Define only one value for each parameter.

Severity Code: 8

IFO092 KEYWORD PARAMETER xxxxxxxx UNDEFINED IN MACRO DEFINITION

Explanation: A keyword parameter has been used in the macro instruction that is not a keyword parameter in the macro prototype, or an equal sign not surrounded by quotes is found in a positional parameter.

Assembler Action: The extra keyword parameter in the macro instruction is ignored.

Programmer Response:

(1) Delete the keyword parameter and its value from the macro instruction; or

(2) make the keyword parameter in the macro call correspond to one of the keyword parameters in the macro prototype; or

(3) define the keyword parameter in the operand field of the macro prototype; or

(4) if you want to include an equal sign in a positional parameter, enclose the parameter within single quotes.

Severity Code: 8

IFO100 DICTIONARY SPACE EXHAUSTED, NO SKELETON DICTIONARY BUILT

Explanation:

- If the message is given for a macro definition or for open code: no available space is left to build the skeleton dictionary after space has been used for the definition of global symbols, sequence symbols, or referenced ordinary symbols.
- If the message is given for a macro instruction: dictionary space was exhausted during the editing of a library macro.

Assembler Action: The macro is not considered defined, and any calls to it are processed as comments. If the error occurs in open code, the entire assembly is processed as comments.

Programmer Response: Within the partition, increase the size of the region that is allocated to assembly, or allocate more of the partition to dictionary space via the BUFSIZE assembler option (see Appendix E).

Severity Code: 8

IFO101 GENERATED OP CODE INVALID OR UNDEFINED

Explanation: The operation code created by substitution is not a valid ordinary symbol or is not a valid machine, assembler, or macro instruction, or defined by an OPSYN instruction.

Assembler Action: The generated statement is treated as a comment.

Programmer Response: Be sure that substitution results in a valid ordinary symbol that consists of from 1 to 8 alphameric characters, the first of which is alphabetic, and that the resulting symbol is a defined operation code.

Severity Code: 8

IFO102 GENERATED OP CODE IS BLANK

Explanation: The op code created by substitution contains no characters, or from 1 to 8 blank characters.

Assembler Action: The generated statement is processed as a comment.

Programmer Response: Be sure that substitution results in a valid ordinary symbol that consists of from 1 to 8 alphameric characters, the first of which is alphabetic.

Severity Code: 8

IFO104 MORE THAN ONE TITLE STATEMENT NAMED

Explanation: This is at least the second TITLE statement that contains something other than a sequence symbol or blanks in the name field.

Assembler Action: The name field is ignored.

Programmer Response: Be sure that the name fields of all but one TITLE statement contain only sequence symbols or blanks.

Severity Code: 4

IFO105 GENERATED FIELD EXCEEDS 255 CHARACTERS

Explanation: As a result of substitution, a character string that is longer than 255 characters has been generated.

Assembler Action: The first 255 characters are used.

Programmer Response: Limit the generation of any character string to 255 characters, minus the number of non-substituted characters. (Limit substitution in the name and operation fields to 8 characters, in the operand field to 255 characters.)

Severity Code: 8

IFO107 CHARACTER STRING USED AS AN ARITHMETIC TERM EXCEEDS 10 CHARACTERS

Explanation: A character string used in a SETA expression or in an arithmetic relation in a SETB expression is longer than 10 characters. Ten is the maximum number of characters permitted in a decimal self-defining term.

Assembler Action: The character string is replaced by an arithmetic value of zero.

Programmer Response: Be sure that all character strings used as described in the explanation are from 1 to 10 decimal digits with a value in a range of 0 to 2,147,483,647. Also be sure that the values of all variables that contribute to the generation of the character string are valid for their type.

Severity Code: 8

IFO108 CHARACTER STRING USED AS AN ARITHMETIC TERM CONTAINS NON-DECIMAL CHARACTERS

Explanation: A character string used in a SETA expression or in an arithmetic relation in a SETB expression contains characters other than 0 through 9.

Assembler Action: The character string is replaced by an arithmetic value of zero.

Programmer Response: Be sure that all character strings used in

a SETA expression or as an arithmetic relation in a SETB expression contain from 1 to 10 decimal digits with a value in the range of 0 to 2,147,483,647. Also be sure that the values of all variables that contribute to the generation of the character string are valid for their type.

Severity Code: 8

IFO109 CHARACTER STRING USED AS ARITHMETIC TERM IS A NULL STRING

Explanation: A character string used in a SETA expression or in an arithmetic relation in a SETB expression is zero characters in length.

Assembler Action: The character string is replaced by an arithmetic value of zero.

Programmer Response: Be sure that all character strings used in an arithmetic context are from 1 to 10 decimal digits with a value in a range of 0 to 2,147,483,647. Also make sure that the values of all variables that contribute to the generation of the character string are valid.

Severity Code: 8

IFO110 ARITHMETIC OVERFLOW IN INTERMEDIATE RESULT OF SETA EXPRESSION

Explanation: During the evaluation of a SETA expression, an intermediate value was produced that was outside the range of -231 to 231-1.

Assembler Action: The intermediate result is replaced by an arithmetic value of zero.

Programmer Response: Be sure that the values of all variables that contribute to the intermediate result are valid. No expression should ever attempt a value outside the range of -231 to 231-1. Overflow may be avoided if you adjust the sequence of expression evaluation, or if you separate components of the expression and evaluate them individually (perhaps by additional SET statements) before combining them.

Severity Code: 8

IFO111 SUBSCRIPT EXPRESSION HAS A ZERO OR NEGATIVE VALUE

Explanation: A term or a SETA expression used as the subscript on a dimensioned global or local variable symbol results in a zero or negative value.

Assembler Action: Any such reference to the dimensioned variable results in a null (zero) value.

Programmer Response: Be sure that the values of all the variables that contribute to the subscript are valid. Expressions that are used as subscripts must have a value in the range of 1 through the declared dimension of the global or local variable. A zero subscript is allowed only on the system variable &SYSLIST.

Severity Code: 8

IFO112 SUBSCRIPT EXPRESSION EXCEEDS MAXIMUM DIMENSION

Explanation: A term or a SETA expression used as the subscript on a dimensioned global or local variable results in a value greater than the declared dimension of the variable.

Assembler Action: Any such reference results in a null (zero) value.

Programmer Response: Be sure that all terms and variables that contribute to the subscript have valid values. Be sure that a term or a SETA expression used as a subscript has a value in the range of 1 through the declared dimension of the global or local variable.

Severity Code: 8

IFO113 ILLEGAL REFERENCE MADE TO A PARAMETER THAT IS A SUBLIST

Explanation: A reference has been made in a SETA or SETB expression (i.e., in an arithmetic context) to a parameter that is a sublist.

Assembler Action: The reference to the parameter results in an arithmetic value of zero.

Programmer Response: Check to see that the proper parameter is being referenced. Be sure that an appropriate value is assigned to a parameter that is referenced in a SETA or SETB expression. Check for a missing subscript.

Severity Code: 8

IFO114 NEGATIVE DUPLICATION FACTOR IN CHARACTER STRING

Explanation: A term or a SETA expression that is used as the duplication factor in a SETC operand results in a negative value.

Assembler Action: The duplication factor is set to an arithmetic value of zero.

Programmer Response: Be sure that any term or expression used as a duplication factor has a positive value, and that the values of all variables that contribute to the duplication factor are valid.

Severity Code: 8

IF0115 FIRST EXPRESSION IN SUBSTRING NOTATION HAS ZERO OR NEGATIVE VALUE

Explanation: A term or SETA expression that is used to specify the starting character for a substring operation has a zero or negative value.

Assembler Action: The assembler assigns the value of null to the substring.

Programmer Response: A term, a SETA expression, or a combination of variables used to produce the first expression in a substring notation must result in a positive, nonzero value, not exceeding the length of the character string.

Severity Code: 8

IF0116 SECOND EXPRESSION IN SUBSTRING NOTATION HAS NEGATIVE VALUE

Explanation: A term or SETA expression that is used to specify the number of characters affected by a substring operation has a negative value.

Assembler Action: The value of the second expression of the substring notation is set to 0, that is, the assembler assigns a value of null to the substring.

Programmer Response: A term, a SETA expression, or a combination of variables used to produce the second expression in a substring notation must result in a non-negative value.

Severity Code: 4

IF0117 FIRST EXPRESSION IN SUBSTRING NOTATION EXCEEDS THE LENGTH OF THE STRING

Explanation: A term or SETA expression that specifies the starting character for a substring operation specifies a character beyond the end of the string.

Assembler Action: The assembler assigns the value of null to the substring.

Programmer Response: Make sure the term, SETA expression, or combination of variables used to produce the first expression in a substring notation results in a value in the range of 1 through the length of the character string.

Severity Code: 8

IFO118 ACTR LIMIT HAS BEEN EXCEEDED

Explanation: The number of AIF and AGO branches within the text segment exceeds the value specified in the ACTR instruction or the conditional assembly loop counter default value.

Assembler Action: If a macro is being expanded, the expansion is terminated. If open code is processed, all remaining statements are processed as comments.

Programmer Response: Correct the conditional assembly loop that caused the ACTR limit to be exceeded, or set the ACTR value to a higher number.

Severity Code: 8

IFO119 ILLEGAL TYPE ATTRIBUTE REFERENCE

Explanation: A type attribute reference is made to a symbol defined by an EQU instruction with an invalid third operand.

Assembler Action: The type attribute value is set to U.

Programmer Response: Correct the third operand on the EQU instruction. It must be a self-defining term in the range 0-255.

Severity Code: 4

IFO120 ILLEGAL LENGTH ATTRIBUTE REFERENCE

Explanation:

- A length attribute reference specifies a SETx symbol; or
- A length attribute reference specifies a symbolic parameter (or a &SYSLIST representation) that does not represent an ordinary symbol; or
- The ordinary symbol referenced by a length or integer attribute reference is defined by an EQU instruction, and the value of the second operand of that instruction is not in the range 0-65535; or
- The ordinary symbol referenced by a length or integer attribute reference is defined in a DC or DS instruction, and the instruction contains a length modifier that is not a self-defining term.

Assembler Action: The length attribute is set to 1.

Programmer Response: Review the use of the length attribute and recode.

Severity Code: 4

IFO123 ILLEGAL SCALE ATTRIBUTE REFERENCE

Explanation:

- A scaling attribute reference specifies a SETx symbol; or
- A scaling attribute reference specifies a symbolic parameter (or a &SYSLIST representation) that does not represent an ordinary symbol; or
- A scaling attribute reference is made to an ordinary symbol whose type attribute is not H, F, G, E, D, L, K, P, or Z; or
- The ordinary symbol referenced by a scaling or integer attribute reference is defined in a DC or DS instruction containing a scaling modifier that is not a self-defining term.

Assembler Action: The scale attribute is set to 0.

Programmer Response: Review the use of the scale attribute and recode.

Severity Code: 4

IFO124 ILLEGAL INTEGER ATTRIBUTE REFERENCE

Explanation:

- An integer attribute reference specifies a SETx symbol; or
- An integer attribute reference specifies a symbolic parameter (or a &SYSLIST representation) that does not represent an ordinary symbol; or
- An integer attribute reference is made to an ordinary symbol whose type attribute is not H, F, G, E, D, L, K, P, or Z.

Assembler Action: The integer attribute is set to 0.

Programmer Response: Review the use of the integer attribute and recode.

Severity Code: 4

IFO125 INVALID NAME - ILLEGAL EMBEDDED CHARACTER OR NON-ALPHABETIC FIRST CHARACTER

Explanation:

- The symbol generated in the name field does not begin with an alphabetic character or it contains a special character or an embedded blank after substitution; or
- for the TITLE instruction: the name field contains a special character.

Assembler Action: The name field is ignored.

Programmer Response: Be sure that the symbol generated in the name field conforms to the rules for forming valid ordinary symbols, or is a valid TITLE name field entry. Also check to

make sure that the values of all variables that contribute to the generation of the symbol in the name field are valid.

Severity Code: 8

IFO126 MORE THAN 5 ERRORS IN THIS STATEMENT, PROCESSING OF THE STATEMENT IS TERMINATED

Explanation: Six or more errors were detected in processing this statement. The maximum number of error messages issued by the processor to each statement is five.

Assembler Action: The sixth error causes this message to be issued, and messages are not issued for any further errors in this statement.

Programmer Response: Correct the indicated errors and check carefully for errors beyond the point indicated by the fifth error message. Assemble again. Any additional errors will be located in the next assembly.

Severity Code: 8

IFO127 VALUE OF CHARACTER STRING USED IN ARITHMETIC CONTEXT EXCEEDS 2,147,483,647

Explanation: A character string used in a SETA expression or in an arithmetic relation in a SETB expression exceeds a value of 2,147,483,647, which is the maximum value allowed for a decimal self-defining term.

Assembler Action: The character string is replaced by an arithmetic value of zero.

Programmer Response: Be sure that all character strings used in an arithmetic context are from 1 to 10 decimal digits and have a value in the range of 0 to 2,147,483,647. Be sure that the values of all variables that contribute to the generation of the character string are valid.

Severity Code: 8

IFO128 GENERATED OP CODE EXCEEDS 8 CHARACTERS

Explanation: The syntax for mnemonic operation codes must follow the same rules as ordinary symbols; that is, they must be from 1 to 8 alphanumeric characters long and the first character must be alphabetic.

Assembler Action: The statement that contains the illegal op code is processed as a comment. Only the first 8 characters of the generated op code appear in the printed statement.

Programmer Response: Be sure that the values of all variables that contribute to the generation of the op code are valid, and be sure that no attempt is made to generate an op code of more than 8 characters.

Severity Code: 8

IFO129 GENERATED SYMBOL IN NAME FIELD EXCEEDS 8 CHARACTERS

Explanation: A generated symbol that appears in the name field exceeds 8 characters. It should be from 1 to 8 alphanumeric characters in length, and the first character should be alphabetic.

Assembler Action: The name field is ignored. Only the first eight characters of the generated symbol appear in the printed statement.

Programmer Response: Be sure that the values of all variables that contribute to the generation of the symbol in the name field are valid. Be sure that no attempt is made to generate a symbol of more than 8 characters.

Severity Code: 8

IFO130 FIRST SUBSCRIPT OF &SYSLIST REFERENCE IS NEGATIVE

Explanation: A term or an arithmetic (SETA) expression that is used as the first subscript of a &SYSLIST reference has resulted in a negative value.

Assembler Action: The parameter reference is treated as a reference to an omitted operand.

Programmer Response: Be sure that the values of all variables that contribute to the generation of the first subscript are valid.

Severity Code: 8

IFO131 INCONSISTENT GLOBAL VARIABLE DECLARATION| SETx INSTRUCTION IGNORED

Explanation: Global variable declaration inconsistent with a previous definition of the variable in another macro definition or in open code.

Assembler Action: The value of the global variable remains the same and the SETx instruction is ignored.

Programmer Response: Correct all inconsistencies between global variable declarations regarding dimension and type.

Severity Code: 8

IFO132 REFERENCE TO INCONSISTENTLY DECLARED GLOBAL VARIABLE RESULTS IN ZERO VALUE

Explanation: An attempt to obtain a value from a global variable has been ignored because the declaration of the global variable was inconsistent with a previous declaration of the same variable in another macro definition or in open code. Either the dimension or the type does not agree.

Assembler Action: The reference to the global variable is replaced by a null or zero value.

Programmer Response: Correct all inconsistencies among declarations of the same global variable.

Severity Code: 8

IFO133 NO WORK SPACE FOR OPEN CODE SKELETON DICTIONARY

Explanation: The allotted dictionary work space is insufficient to build the skeleton dictionary for open code. Since the generation process requires the open code dictionary, generation is not attempted.

Assembler Action: The entire assembly is processed as comments.

Programmer Response: Within the partition, increase the size of the region that is allocated to assembly, or allocate more of the partition to dictionary space via the BUFSIZE assembler option (see Appendix E).

Severity Code: 12

IFO157 DC OPERAND VALUE TOO LONG

Explanation: The object code generated from an operand in a DC instruction is too long. The maximum object code length of a DC operand is 16,777,215 bytes.

Assembler Action: The specified value is ignored.

Programmer Response: Make the constant shorter, or break it up into two constants.

Severity Code: 8

IFO158 NAME OF STATEMENT IN DSECT USED IN RELOCATABLE ADDRESS CONSTANT

Explanation: A non-paired relocatable term used in an A-type or Y-type address constant is defined in a dummy section.

Assembler Action: The constant is ignored.

Programmer Response:

- Make sure the relocatable term is not defined in a dummy section; or
- Make sure the term defined in the dummy section is paired with another term (with the opposite sign) from the same dummy section.

Severity Code: 8

IFC159 RELOCATABLE EXPRESSION AS EXPLICIT DISPLACEMENT IN S-TYPE CONSTANT

Explanation: The displacement used in an explicit S-type address constant specification is a relocatable expression.

Assembler Action: The value of the operand is set to zero and no entry is made in the relocation dictionary.

Programmer Response: Make sure the displacement is specified as an absolute expression, or specify an implicit address.

Severity Code: 8

IFO161 INVALID LITERAL NEAR OPERAND COLUMN nn

Explanation: An invalidly constructed literal appears near the specified operand column.

Assembler Action: The value of any reference to the invalid literal is set to 0.

Programmer Response: A literal should be constructed like a DC or DS constant with the following exceptions:

- The literal is preceded by a equal sign.
- The duplication factor must not be 0.

Severity Code: 8

IFO162 VALUE ERROR - SHOULD BE BETWEEN 0 AND 9 NEAR OPERAND COLUMN nn

Explanation: A value is negative or is not in the range of 0 to 9, which is required by this instruction.

Assembler Action: Zeros are generated in place of the machine instruction in the object module.

Programmer Response: Be sure the operand field has a positive value in the range of 0 to 9.

Severity Code: 8

IFO163 MISSING OR INVALID SYMBOL IN NAME FIELD

Explanation: One of two errors has occurred:

- A symbol is missing in the name field where one is required.
- The symbol in the name field is invalid.

Assembler Action: The statement is processed as a comment.

Programmer Response: Supply a valid name.

Severity Code: 4

IFO164 INVALID OR ILLEGAL START STATEMENT

Explanation: The START statement did not start the first control section in the assembly, or the operand on the START statement was not an absolute value.

Assembler Action: The START statement is treated as a CSECT statement.

Programmer Response: Be sure that the START statement has an absolute operand and that it begins the first control section in the assembly.

Severity Code: 4

IFO165 NULL PUNCH OPERAND OR PUNCH OPERAND EXCEEDS 80 CHARACTERS

Explanation: The operand of a PUNCH instruction either specifies only a null string surrounded by quotes, or is more than 80 characters long.

Assembler Action: The PUNCH statement is processed as a comment.

Programmer Response: Be sure that the operand of a PUNCH statement consists of from 1 to 80 characters surrounded by quotes.

Severity Code: 4

IFO167 SYMBOL FILE OUT OF STEP

Explanation: References to the symbol file (an internal data file) out of step because of an error in the source code. This message is always accompanied by other error messages, not necessarily for the same statement.

Assembler Action: Assembly continues, but results subsequent to the point of error may not be valid.

Programmer Response: This message will always be accompanied by user errors. Correct them and reassemble the program.

If the message is issued even though the source code is error-free, do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- If a COPY statement was used, execute the IEBPTPCH utility to obtain a copy of the partitioned data set member specified in the COPY statement.
- Make sure that MSGLEVEL=(1,1) was specified in the JCB statement.

Severity Code: 16

IFO168 AN ARITHMETIC EXPRESSION NOT USED IN CONDITIONAL ASSEMBLY CONTAINS MORE THAN 20 TERMS

Explanation: An arithmetic expression used in a macro definition or in open code, but not in a conditional assembly statement, contains more than 19 unary and binary operators and 6 levels of parentheses. The maximum number of terms this combination allows is 20.

Assembler Action: The value of the expression is set to 0.

Programmer Response: Be sure that this arithmetic expression does not contain more than 19 operators (unary and binary) and 6 levels of parentheses. If greater complexity is necessary, use EQU statements to evaluate intermediate results.

Severity Code: 8

IFO169 INVALID SELF-DEFINING TERM NEAR OPERAND COLUMN nn

Explanation: A self-defining term was invalidly specified.

Assembler Action: The value of the term is set to zero.

Programmer Response: Check the syntax and correct the error.

Severity Code: 8

IFO170 TWO ADJACENT BINARY OPERATORS, OR BINARY OPERATOR EXPECTED BUT NOT FOUND NEAR OPERAND COLUMN nn

Explanation: One of two errors has occurred.

(1) Two binary operators appear consecutively near the column specified in the message text. This applies only to "*" (multiply) and "/" (divide).

(2) A binary operator was expected near the column specified in the message text, but none was found. A single binary operator must occur between all terms of an expression.

Assembler Action: The expression that contains the absent or illegal operator is set to zero.

Programmer Response:

(1) Eliminate one of the binary operators.

(2) Provide a binary operator.

Severity Code: 8

IFO171 TITLE STATEMENT OPERAND EXCEEDS 100 CHARACTERS

Explanation: The operand of a TITLE instruction contains more than 100 characters.

Assembler Action: The character string in the operand is truncated to 100 characters.

Programmer Response: Be sure that the length of the character string in the operand of a TITLE statement does not exceed 100 characters.

Severity Code: 4

IFO172 VALUE OF ORG OPERAND IS LESS THAN THE CONTROL SECTION STARTING ADDRESS

Explanation: The operand of an ORG statement results in a value less than the starting address of the control section.

Assembler Action: The ORG statement is processed as a comment and has no effect on the value of the location counter.

Programmer Response: Be sure that the operand of the ORG statement is a positive relocatable expression, greater than the starting address of the control section, or blank.

Severity Code: 8

IFO173 ONE OR MORE SYMBOLS IN AN ORG OPERAND DO NOT BELONG TO THE CURRENT CSECT, DSECT, OR COM

Explanation: One or more of the symbols used in the operand of an ORG statement are not defined in the current control section (dummy, common or ordinary).

Assembler Action: The ORG statement is processed as a comment and the value of the location counter remains unchanged.

Programmer Response: Be sure that all symbols used in the operand field of an ORG statement belong to (are defined by appearing in the name field of a statement within) the current control section.

Severity Code: 8

IFO174 ORG OPERAND IS ABSOLUTE, MUST BE RELOCATABLE

Explanation: An absolute term or expression used in the operand of an ORG statement must be a relocatable term, a relocatable expression, or a blank.

Assembler Action: The ORG instruction is processed as a comment and the value of the location counter remains unchanged.

Programmer Response: Be sure that the operand of an ORG statement is a relocatable term, a relocatable expression, or a blank. An ORG to an absolute address is not possible because the assembler assumes that all location references are relocatable. A common error is an ORG to 0. Since the start of the program is not absolute machine location 0 but relocatable 0, replace the 0 with a symbol or expression that

makes reference to the labeled program start.

Severity Code: 8

IFC175 OPERAND SHOULD BEGIN WITH A QUOTE

Explanation: A quote was expected to begin a character string in the operand field, but was not found.

Assembler Action: The invalid character string is ignored.

Programmer Response: Supply the missing leading quote in the character string of the operand.

Severity Code: 8

IFC176 UNPAIRED AMPERSAND NEAR OPERAND COLUMN nn

Explanation: A single ampersand followed by a blank was found in a quoted character string. If an ampersand is desired as a character in a quoted character string, two ampersands must be coded. Ampersands must be either paired or part of a valid variable symbol.

Assembler Action: The character string that contains the illegal ampersand is ignored.

Programmer Response: Determine whether the ampersand is desired as a character in a quoted character string or whether the ampersand is intended as the beginning of a valid variable symbol, and correct the error.

Severity Code: 8

IFC177 MISSING OPERAND

Explanation: This statement requires an operand, but none is found.

Assembler Action: The statement which lacks the operand is processed as a comment.

Programmer Response: Supply a valid operand.

Severity Code: 12

IFC178 SYNTAX ERROR NEAR OPERAND COLUMN nn

Explanation: A syntax error has occurred in the operand of this statement.

Assembler Action: The statement which contains the invalid operand is processed as a comment.

Programmer Response: Correct the syntax of the operand. There are a large number of syntactic errors that can produce this

diagnostic. All of them require careful checking of the syntax of the specific type of statement being processed. The error is logged at the point where the syntax becomes ambiguous or unrecognizable, not necessarily at the point where the actual error occurs.

Severity Code: 8

IFO179 OPERAND SUBFIELD NEAR OPERAND COLUMN nn MUST BE ABSOLUTE

Explanation: All terms and expressions used in the operand field of this statement must result in an absolute value.

Assembler Action: The operand is processed as a comment.

Programmer Response: Be sure that each term or expression used in the operand field of this statement has an absolute value. No relocatable expressions are allowed.

Severity Code: 8

IFO180 OPERAND 2 OF CNOP MUST BE EITHER 4 OR 8

Explanation: The second operand of a CNOP statement must be either 4 or 8.

Assembler Action: The CNOP statement is processed as a comment and no alignment is performed.

Programmer Response: Be sure that the second operand of a CNOP statement is either a 4 or an 8.

Severity Code: 12

IFO181 OPERAND 1 OF CNOP MUST BE 0, 2, 4, OR 6

Explanation: The first operand of a CNOP statement must be 0, 2, 4, or 6.

Assembler Action: The CNOP statement is ignored and no alignment is performed.

Programmer Response: Be sure that the first operand of a CNOP statement is a 0, 2, 4, or 6.

Severity Code: 12

IFO182 OPERAND 1 OF CNOP IS NOT LESS THAN OPERAND 2

Explanation: The value of the first operand of a CNOP statement must be less than the value of the second operand.

Assembler Action: The CNOP statement is processed as a comment and no alignment is performed.

Programmer Response: Check the validity of each operand of the CNOP statement to be sure that the value of the second operand is greater than the value of the first operand.

Severity Code: 12

IFC183 MNCTE/CCW OPERANC EXCEECS 255

Explanaticn: The value of an operand used as an MNCTE severity ccde or as the first operand in a channel ccmmand wcrd (CCW) exceeds 255.

Assembler Action: The MNOTE is prccessed as a ccmment. Space is allccated fcr the CCW, but the value fcr the flagged cperand is set tc 0.

Programmer Response: Check the validity cf the cperand.

Severity Code: 12

IFC184 INVALID RANGE ON CCW NEAR OPERAND COLUMN nn, 65535 IS MAXIMUM VALUE

Explanaticn: The value of the fourth operand of a channel command word has exceeded X'FFFF' (65535).

Assembler Action: Space is allccated fcr the CCW, but the value of the flagged operand is set to 0.

Programmer Response: Check the validity cf the fcurth cperand cf the channel ccmmand word.

Severity Ccde: 12

IFC185 ELANK EXPECTEC AS A CELIMITER NEAR OPERAND CCLUMN nn

Explanation: A blank was expected as a delimiter but ncne was found. Subsequent characters have nc syntactic meaning, and the statement is ambiguous.

Assembler Action: The statement that ccntains the invalid delimiter is processed as a comment.

Prcgrammer Response: Supply a blank delimiter.

Severity Ccde: 8

IFC186 INVALIC SYMEOL NEAR OPERAND COLUMN nn OF ENTRY, EXTRN, OR WXTRN

Explanation: An improperly constructed symbcl was found in the operand field of an ENTRY, EXTRN, cr WXTRN statement.

Assembler Acticn: The statement that contains the invalid symbol is processed as a comment.

Programmer Response: Be sure that the symbol in the operand field of EXTRN, WXTRN, or ENTRY statements contain from 1 to 8 alphameric characters, the first of which is alphabetic.

Severity Code: 8

IFO187 SYMBOL LONGER THAN 8 CHARACTERS NEAR OPERAND COLUMN NN

Explanation: A symbol that is more than 8 characters in length has appeared in the operand field of this statement.

Assembler Action: The invalid symbol in the operand field is replaced by a zero.

Programmer Response: Be sure that symbols do not exceed 8 characters in length. A missing or misplaced delimiter or operator may cause a symbol to appear longer than intended.

Severity Code: 8

IFO188 xxxxxxxx IS AN UNDEFINED SYMBOL

Explanation: The symbol that appears in the message text has not appeared in the name field of another statement, or as an operand of an EXTRN or WXTRN statement.

Assembler Action: Reference to the undefined symbol results in a zero value.

Programmer Response: Define the symbol in the program.

Severity Code: 8

IFO189 INVALID ENTRY OPERAND, LINKAGE CANNOT BE PERFORMED

Explanation: The symbol in the operand field of an ENTRY statement is invalid because it is either undefined or improperly defined.

Assembler Action: The invalid symbol in the operand field is processed as a comment, and no linkage is provided if another program references it.

Programmer Response: Define the symbol at an appropriate place in this program, or correct it. A valid symbol consists of from 1 to 8 alphameric characters, the first of which is blank.

Severity Code: 8

IFO190 OPERAND OF PUSH STATEMENT IS NOT USING OR PRINT NEAR OPERAND COLUMN nn

Explanation: The only symbols allowed in the operand field of a

PUSH or POP statement are PRINT and USING, in any order, separated by commas.

Assembler Action: The PUSH instruction is processed as a comment.

Programmer Response: Be sure the operand of the PUSH statement is either PRINT or USING or both.

Severity Code: 4

IFO191 PUSH LEVELS EXCEED 4 NEAR OPERAND COLUMN nn

Explanation: More than 4 levels of PUSH and POP statements were attempted for either PRINT or USING.

Assembler Action: The PUSH instruction is processed as a comment.

Programmer Response: Rework the program logic to require no more than 4 levels of PUSH and POP for USING and 4 for PRINT.

Severity Code: 8

IFO192 OPERAND OF POP STATEMENT IS NOT USING OR PRINT NEAR OPERAND COLUMN nn

Explanation: The only symbols allowed in the operand of a PUSH or POP statement are USING and PRINT, in any order, separated by commas.

Assembler Action: The POP instruction is processed as a comment.

Programmer Response: Be sure the operand of the POP statement is either PRINT or USING or both.

Severity Code: 4

IFO193 POP REQUEST NOT BALANCED BY PREVIOUS PUSH

Explanation: No PUSH request was issued prior to this POP request, or more POP statements have been issued than PUSH statements. A POP statement restores the USING or PRINT status saved by the most recent PUSH statement, on a one for one basis.

Assembler Action: The POP instruction is processed as a comment.

Programmer Response: Check for errors in balancing PUSH and POP statements, or rework the program logic to request balanced PUSH and POP statements. Repetition of a given operand (i.e., USING or PRINT) on a single PUSH or POP statement is treated as multiple statements, and could cause unbalanced PUSH and POP statements.

Severity Code: 8

IFO194 INVALID OPTION IN PRINT STATEMENT NEAR OPERAND COLUMN nn

Explanation: An option appears in the operand field of a PRINT statement that is not one of the following: ON, OFF, GEN, NOGEN, DATA, and NODATA.

Assembler Action: The invalid operand is ignored.

Programmer Response: Be sure that only the options listed in the explanation above appear in the operand field of a PRINT statement.

Severity Code: 4

IFO195 INVALID USING OR DROP STATEMENT NEAR OPERAND COLUMN nn

Explanation: One of three errors has occurred:

(1) register 0 is specified for other than the second operand of a USING statement, or

(2) a register number outside the range of 0 to 15 has been used, or

(3) a DROP statement has been issued for a register that was never assigned for use by a USING statement.

Assembler Action: The invalid register specification is set to zero.

Programmer Response: The second and following operands of a USING or DROP instruction must be decimal terms 0 to 15. Register 0 may only be specified as the second operand of a USING statement.

Severity Code: 12

IFO196 xxxxxxxx HAS BEEN PREVIOUSLY DEFINED

Explanation: The specified symbol has previously appeared in the name field of a statement or in the operand field of an EXTRN or WXTRN instruction.

Assembler Action: All references to the symbol are interpreted as references to the first definition of the symbol.

Programmer Response: A given symbol must be defined only once. Determine which occurrence of the symbol you want to use, and change all others.

Severity Code: 8

IFO197 *** MNOTE ***

Explanation: An MNOTE statement has been encountered during the generation of a macro or in open code. The text of the MNOTE message appears in-line in the listing at the point where it is

encountered. (Refer to CS/VE Assembler Language for a description of the MNOTE instruction.)

Assembler Action: None.

Programmer Response: Investigate the reason for the MNOTE. Errors flagged by MNOTE will often cause unsuccessful execution of the program, depending upon the severity code.

Severity Code: An MNOTE is assigned a severity code of 0 to 255 by the writer of the MNOTE statement.

IFO198 INVALID TYPE DECLARED ON DC/DS/DXD CONSTANT NEAR OPERAND COLUMN nn

Explanation: Operand subfield 2 is not a valid type for a DC, DS, or DXD statement. Valid types are the following: A, B, C, D, E, F, H, L, P, Q, S, V, X, Y, and Z.

Assembler Action: The statement that contains the invalid type declaration is processed as a comment.

Programmer Response: Supply a valid type in operand subfield 2.

Severity Code: 8

IFO199 INVALID LENGTH MODIFIER NEAR OPERAND COLUMN nn

Explanation: The length modifier in operand subfield 3 of this statement is invalid. The length attribute of a symbol is not allowed as a term in the length modifier expression for the first operand of the DC, DS, or DXD statement in which the symbol is defined. For example, SYM DC CL(L'SYM)'AA' is invalid.

Assembler Action: The statement that contains the invalid length modifier is processed as a comment.

Programmer Response: Supply a valid length modifier, or eliminate the explicit length modifier.

Severity Code: 8

IFO200 INVALID SCALE MODIFIER NEAR OPERAND COLUMN nn

Explanation: The scale modifier in operand subfield 3 of a DC, DS, or DXD statement is invalid. The scale modifier should be either a decimal value or an absolute expression enclosed in parentheses.

Assembler Action: The statement that contains the invalid scale modifier is processed as a comment.

Programmer Response: Supply a valid scale modifier for the type of constant used.

Severity Code: 8

IFO201 ILLEGAL OR INVALID EXPONENT MODIFIER IN DC/DS/DXD CONSTANT NEAR OPERAND COLUMN nn

Explanation: An exponent modifier used in a DC, DS, or DXD constant is not a decimal self-defining term, an absolute expression enclosed in parentheses, or produces a value outside the range allowed for that constant type.

Assembler Action: The invalid or illegal operand is ignored.

Programmer Response: Be sure that the exponent modifier used conforms to the rules for exponent modifiers for each type of DC, DS, or DXD constant.

Severity Code: 8

IFO202 ARITHMETIC PRECISION OF FLOATING-POINT CONSTANT LOST NEAR OPERAND COLUMN nn

Explanation: Low order digits were lost during the construction of an L-, D-, or E-type constant, because the designated field was too small to contain the whole constant.

Assembler Action: The value of the constant is set to zero.

Programmer Response: Check the length, scale, and exponent modifier of the flagged constant.

Severity Code: 8

IFO203 L-, D-, E-, F-, H-, OR Y-TYPE CONSTANT TRUNCATED, HIGH ORDER DIGITS LOST NEAR OPERAND COLUMN nn

Explanation: The high order digits of an L-, D-, E-, F-, H-, or Y-type constant were lost because the designated field was too small to contain the whole constant.

Assembler Action: Processing continues using the truncated constant.

Programmer Response: Modify the explicit or implicit length of the constant, so that the value may be contained within the area designated for it.

Severity Code: 4

IFO204 RELOCATABLE EXPRESSION NOT ALLOWED IN A- OR Y-TYPE ADDRESS CONSTANT WITH BIT LENGTH SPECIFICATION

Explanation: A relocatable expression in used to specify a constant for which bit length specification is used. This is not allowed.

Assembler Action: The value of the operand is set to 0 and no entry for this constant is made in the relocation dictionary.

Programmer Response: Convert the operand to an absolute expression, or use a length of 3 or 4 bytes for A-type or 2 bytes for Y-type constants.

Severity Code: 8

IFO205 RELOCATABLE Y-TYPE CONSTANT, VALUE TRUNCATED TO RIGHTMOST 2 BYTES

Explanation: A relocatable Y-type constant has been declared. This is a warning only. All relocatable Y-type constants are diagnosed in this manner because the assembler must provide an entry in the relocation dictionary for each one. If the actual address is contained within the rightmost two bytes and the coding is otherwise correct, when the program is loaded and relocation is considered the constant will be resolved. If the address cannot be contained in the rightmost two bytes, it is likely that further relocatability errors will result.

Assembler Action: The value of the constant is truncated to the rightmost two bytes.

Programmer Response: Be sure that the value of the Y-type constant will not exceed two bytes when the program has been loaded and the relocation factor has been considered.

Severity Code: 4

IFO206 DUPLICATION FACTOR ERROR

Explanation: The duplication factor in a DC, DS, or DXD statement is negative.

Assembler Action: No storage is reserved for the operand, but alignment is performed as required by the type of constant used.

Programmer Response: Supply a non-negative duplication factor.

Severity Code: 8

IFO207 OPERAND OF Q-TYPE CONSTANT DOES NOT NAME A DSECT OR DXD

Explanation: The symbol in the operand field of a Q-type constant must have been previously defined as the name of a DSECT or DXD section.

Assembler Action: The value of the constant is set to 0.

Programmer Response: Define the symbol as the name of a DSECT or DXD section. The symbol must be defined before being used

in the constant.

Severity Code: 8

IFO208 DISPLACEMENT GREATER THAN X'FFF'

Explanation: The displacement of this statement or the address referenced by this statement is greater than X'FFF' (decimal 4095). The displacement field in the machine instruction must contain a value of from 0 to 4095.

Assembler Action: The base and displacement fields of the machine instruction are set to 0.

Programmer Response: Correct the displacement term or expression or provide another base register with a USING statement.

Severity Code: 8

IFO209 ADDRESSABILITY ERROR - BASE AND DISPLACEMENT CANNOT BE RESOLVED AND ARE SET TO 0

Explanation: The assembler cannot resolve the address of this statement or the address referenced by this statement for one of the following reasons:

- Current USING registers produce a displacement of less than 0 or greater than 4095.
- No USING registers are available.

Assembler Action: The base and displacement fields of the machine instruction are set to 0.

Programmer Response: Make sure you have correctly set up base registers with the USING instruction. Be sure the referenced address can be specified by the value in a USING register plus a displacement in the range of 0 through 4095.

Severity Code: 8

IFO210 TOO FEW OPERANDS

Explanation: More operands are required for this statement, but they were not found.

Assembler Action: The value of any missing operand is set to 0.

Programmer Response: Supply the necessary operands. Refer to Principles of Operation for details on the operands required for this instruction.

Severity Code: 12

IFO211 TOO MANY OPERANDS

Explanation:

- More than 255 operands in a DC, DS, or DXD instruction; or
- Too many operands in a machine instruction.

Assembler Action: The extra operands are ignored.

Programmer Response: Delete the extra operands. Refer to Principles of Operation for details on operands required for individual machine instructions.

Severity Code: 12

IFO212 PREMATURE END OF OPERAND NEAR OPERAND COLUMN nn

Explanation: A term or an expression used as an operand is incomplete.

Assembler Action: The value of the operand is set to 0.

Programmer Response: Supply the characters necessary to terminate the operand.

Severity Code: 8

IFO213 COMPLEXLY RELOCATABLE EXPRESSION NEAR OPERAND COLUMN nn

Explanation: The indicated operand contains a complexly relocatable expression. The expression should be absolute or simply relocatable.

Assembler Action: The value of the complexly relocatable expression is set to 0.

Programmer Response: Be sure that only absolute and simply relocatable expressions are used in the operand field of this statement.

Severity Code: 8

IFO214 ILLEGAL USE OF LITERAL NEAR OPERAND COLUMN nn

Explanation: A literal is used in an assembler instruction, in another literal, or in a field of a machine instruction where it is not allowed.

Assembler Action: The value of the operand where the literal is used is set to 0.

Programmer Response: Use a valid relocatable term or expression in place of the literal. If applicable, replace the literal with the name of a DC statement which defines the same constant as the literal.

Severity Code: 12

IFO215 ILLEGAL DELIMITER, RIGHT PARENTHESIS EXPECTED NEAR OPERAND COLUMN nn

Explanation: A right parenthesis was expected as a delimiter, but none was found.

Assembler Action: The value of the operand that is lacking a right parenthesis is set to 0.

Programmer Response: Supply a right parenthesis.

Severity Code: 8

IFO216 ILLEGAL OPERAND FORMAT NEAR OPERAND COLUMN nn

Explanation: The operand of this statement is illegally constructed.

Assembler Action: The value of the operand is set to 0.

Programmer Response: Refer to Principles of Operation for details on the operand structure of this statement, and supply a valid operand.

Severity Code: 12

IFO217 RELOCATABILITY ERROR NEAR OPERAND COLUMN nn

Explanation: One of the following fields contains a relocatable value. All values in these fields must be absolute.

- Immediate field in an SI instruction
- Mask field
- Register specification
- Length modifier

Assembler Action: If any of the above fields contains a relocatable value, the value of the field is set to 0.

Programmer Response: Be sure that the field contains an absolute value.

Severity Code: 12

IFO218 INVALID REGISTER SPECIFICATION - EVEN-NUMBERED REGISTER REQUIRED

Explanation: An odd-numbered register was specified in a context that requires an even-numbered register.

Assembler Action: The invalid operand is set to 0.

Programmer Response: Specify an available even-numbered register. Refer to the Principles of Operation for details on the register requirements of this instruction.

Severity Code: 12

IFO219 REGISTER OR IMMEDIATE FIELD OVERFLOW NEAR OPERAND COLUMN nn

Explanation:

- The value of the immediate field used in an SI instruction is greater than 255; or
- A register number was specified that was greater than 15.

Assembler Action: The value of the field where the overflow occurred is set to 0.

Programmer Response: Be sure the value of an immediate field does not exceed 255 and that no register number greater than 15 is specified.

Severity Code: 8

IFO220 ALIGNMENT ERROR NEAR OPERAND COLUMN nn

Explanation: The operand of this instruction refers to a main storage location that is not on the boundary required by the instruction.

Assembler Action: The faulty alignment is unchanged.

Programmer Response: Align the main storage location referenced in the operand field. Refer to the Principles of Operation for details on the boundary requirements of this instruction. For machines that do not require data to be aligned to certain boundaries, specify NOALIGN as an assembly option and no error will occur.

Severity Code: 4

IFO221 ILLEGAL INDEX REGISTER OR LENGTH MODIFIER NEAR OPERAND COLUMN nn

Explanation: An index register or a length field was specified for a machine instruction where none is expected.

Assembler Action: The invalid specification is ignored.

Programmer Response: Correct the index register or length field specification.

Severity Code: 12

IFO222 INVALID INDEX REGISTER SPECIFIED NEAR OPERAND COLUMN nn

Explanation: A register number not in the range 0 - 15 has been specified as an index register.

Assembler Action: A default value of 0 (to indicate that no indexing is used) replaces the invalid index register specification in the machine instruction.

Programmer Response: Specify an available register in the range of 0 to 15 as an index register.

Severity Code: 12

IFO223 RELOCATABLE INDEX REGISTER SPECIFIED NEAR OPERAND COLUMN nn

Explanation: A relocatable value has been specified as an index register.

Assembler Action: A default value of 0 (to indicate that no indexing is used) replaces the invalid index register specification in the machine instruction.

Programmer Response: Specify an absolute value in the range of 0 to 15 as an index register.

Severity Code: 12

IFO224 LENGTH ERROR NEAR OPERAND COLUMN nn

Explanation:

- The length modifier of a constant is illegal or invalid for the type of constant; or
- A constant of type C, X, B, Z, or P is too long; or
- A relocatable address constant has an illegal length.

Assembler Action: The operand in error and any following operands of the DC, DS, or DXD statement are processed as comments. An address constant with an illegal length is truncated.

Programmer Response: Supply a valid length modifier or decrease the length of the operand.

Severity Code: 8

IFO225 RELOCATABLE LENGTH FIELD IN MACHINE INSTRUCTION NEAR OPERAND COLUMN nn

Explanation: The length field of this machine instruction is specified as relocatable; an absolute term or expression is required.

Assembler Action: The length field in error is assembled to 0.

Programmer Response: Use an absolute term or expression to specify the length field.

Severity Code: 4

IFO226 BASE REGISTER OF MACHINE INSTRUCTION NOT ABSOLUTE NEAR OPERAND COLUMN nn

Explanation: An explicit base register has been specified as a relocatable value; an absolute term or expression is required.

Assembler Action: The operand in error (base and displacement) is assembled to 0.

Programmer Response: Use an absolute term or expression to specify the base register.

Severity Code: 12

IFO228 RELOCATABLE DISPLACEMENT IN MACHINE INSTRUCTION NEAR OPERAND COLUMN nn

Explanation: In a machine instruction that has an explicit base register specification, the specification for the displacement field is relocatable. As this would imply a second base register, the combination is invalid.

Assembler Action: The displacement field of the machine instruction is assembled to 0.

Programmer Response: Either specify the displacement as an absolute term or expression, or delete the explicit base register.

Severity Code: 8

IFO229 POSSIBLE REENTERABILITY ERROR NEAR OPERAND COLUMN nn

Explanation: This machine instruction could store data into a control section or common area that is not dynamically acquired. This message is produced only when the RENT assembler option is specified in the PARM field of the EXEC statement.

Assembler Action: The statement is assembled as written.

Programmer Response: If you want reentrant code, correct the instruction so that it references a DSECT or other dynamically acquired space. Otherwise you can suppress reentrant checking by specifying the NORENT assembler option.

Note: Absence of this message does not guarantee reentrant code, as the assembler has no control over addresses actually loaded into base and index registers at program execution time.

Severity Code: 4

IFO230 BASE REGISTER NUMBER GREATER THAN 15 NEAR OPERAND COLUMN nn

Explanation: An explicit base register in a machine instruction or S-type address constant is greater than 15.

Assembler Action: The base register field of the machine

instruction is assembled to 0.

Programmer Response: Specify the base register in the range of 0 to 15.

Severity Code: 12

IFO231 SYMBOL NOT PREVIOUSLY DEFINED - xxxxxxxx

Explanation: A symbol in this statement is used in a way that requires previous definition, but it has not been previously defined. For example, a symbol in a duplication factor expression or modifier expression of a DC statement must be previously defined.

Assembler Action: The value of the symbol or the expression that contains it is set to 0.

Programmer Response: Define the symbol earlier in the program. Add a defining statement if it does not exist, or place the existing defining statement ahead of the statement that references it.

Severity Code: 8

IFO233 MORE THAN 6 LEVELS OF PARENTHESES NEAR OPERAND COLUMN NN

Explanation: An expression in this statement contains more than six nested levels of parentheses.

Assembler Action: The value of the expression is set to 0.

Programmer Response: Rewrite the expression to reduce the number of levels of parentheses, or use a preliminary statement (such as an EQU) to partially evaluate the expression.

Severity Code: 8

IFO234 PREMATURE END OF EXPRESSION NEAR OPERAND COLUMN nn

Explanation: An expression in this statement ended prematurely due to one of the following errors:

- Unpaired parenthesis
- Illegal character
- Illegal operator
- Operator not followed by a term

Assembler Action: The value of the expression is set to 0.

Programmer Response: Check the expression for omitted or mispunched characters or terms.

Severity Code: 8

IFO235 ARITHMETIC OVERFLOW NEAR OPERAND COLUMN nn

Explanation: The intermediate value of a term or an expression is not in the range -231 through 231-1.

Assembler Action: The value of the expression is set to 0.

Programmer Response: Rewrite the expression or term. The assembler computes all values using fixed-point full-word arithmetic. Or, perform arithmetic operations in a different sequence to avoid overflow.

Severity Code: 8

IFO236 ILLEGAL CHARACTER IN EXPRESSION NEAR OPERAND COLUMN nn

Explanation: Syntax error. A character in an expression has no syntactic meaning in the context used; the assembler cannot determine if it is a symbol, an operator, or a delimiter.

Assembler Action: The value of the expression is set to 0.

Programmer Response: Check the expression for unpaired parentheses, invalid delimiter, invalid operator, or a character (possibly unprintable) that is not recognized by the assembler. The 51 characters recognized by the assembler are:

Letters: A through Z and $ # @
Digits: 0 through 9
Special Characters: + - , = . * ( ) ' / &
Blank

Severity Code: 8

IFO237 CIRCULAR DEFINITION

Explanation: The value of the first expression in the operand field of an EQU statement is dependent upon the value of the symbol being defined in the name field.

Assembler Action: The value of the expression defaults to the current location counter value.

Programmer Response: Remove circularity in the definition.

Severity Code: 8

IFO238 ILLEGAL AMPERSAND IN SELF-DEFINING TERM NEAR OPERAND COLUMN nn

Explanation: An ampersand in a self-defining term is unpaired and/or not part of a quoted character string.

Assembler Action: The value of the expression containing the self-defining term is set to 0.

Programmer Response: Check that all ampersands in the term are paired and part of a quoted character string. (The only valid

use of a single ampersand is as the first character of a variable symbol.) Note that ampersands produced by substitution must also be paired.

Severity Code: 8

IFO239 INVALID FLOATING POINT CHARACTERISTIC

Explanation: A converted floating-point constant is too large or too small for the field assigned to it. The allowable range is 7.2x1075 to 5.3x10-77.

Assembler Action: The floating-point constant is assembled to 0.

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 Code: 8

IFO240 CHARACTER STRING OR SELFDEFINING TERM TERMINATED BEFORE ENDING QUOTE FOUND

Explanation: The assembler has found what appears to be a quoted character string or a self-defining term, but the closing quote is missing, or an illegal character is found before the closing quote.

Assembler Action: The term or expression is ignored.

Programmer Response: Supply the missing quote or check for other syntax errors.

Severity Code: 8

IFO241 SECOND OPERAND OF CCW NOT BETWEEN 0 and X'FFFFFF'

Explanation: The second operand of a CCW instruction, which specifies the data address, is outside the range of 0 to X'FFFFFF'.

Assembler Action: The low-order three bytes of the operand are used.

Programmer Response: Supply a correct term or expression for the second operand.

Severity Code: 8

IFO242 SPACE OPERAND NOT A SINGLE POSITIVE DECIMAL SELFDEFINING TERM

Explanation: The operand of a SPACE instruction is not a zero or positive decimal self-defining term.

Assembler Action: The SPACE statement is processed as a comment.

Programmer Response: Use a single decimal self-defining term with a zero or positive value.

Severity Code: 4.

IFO243 FIRST CCW OPERAND IS NEGATIVE

Explanation: The first operand (command code) of a CCW instruction is negative. The value of the operand must be in the range 0-255.

Assembler Action: The CCW is processed as a comment.

Programmer Response: Supply an operand with a value in the range of 0-255.

Severity Code: 8

IFO244 BITS 38 AND 39 OF CCW OPERAND NOT ZERO

Explanation: The bits specified as bits 38 and 39 of a CCW instruction are not zero.

Assembler Action: The bits are set as specified.

Programmer Response: Correct the third operand of the CCW instruction.

Severity Code: 8

IFO246 LOCATION COUNTER OVERFLOW

Explanation: The location counter is greater than X'FFFFFF' (224-1), the largest address that can be contained in 3 bytes.

Assembler 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 of the error is a high ORG statement value or a high START statement value. Correct the value or split up the control section.

Severity Code: 8

IF0254 ILLEGAL FORMAT OF SECOND OPERAND OF END STATEMENT

Explanation: Second operand of END instruction is inconsistent with the format required.

Assembler Action: Second operand ignored.

Programmer Response: Correct the operand.

Severity Code: 8

IF0255 FIXED OR FLOATING POINT EXPRESSION ERROR NEAR OPERAND COLUMN nn

Explanation: An error occurred during conversion of a decimal number into a fixed-point or floating-point number.

Assembler Action: The number is assembled as zeros.

Programmer Response: Check the scale and exponent modifier of the number for validity.

Severity Code: 4

IF0256 SYSGO DD CARD MISSING -- NOOBJECT OPTION USED

Explanation: A DD statement for the SYSGO data set is not included in the JCL for this assembly. The SYSGO data set normally receives the object module output of the assembler when it is to be used as input to the linkage editor or loader, executed in the same job.

Assembler Action: The program is assembled using the NOOBJECT option. No output is written on SYSGO. If the DECK option is specified, the object module will be written on the device specified in the SYSPUNCH DD statement.

Programmer Response: Optional. If the assembly is error free and the object module has been produced on SYSPUNCH, you can execute it without reassembling. Otherwise, reassemble the program and include a SYSGO DD statement in the JCL or use a cataloged procedure that includes it. (See the section "The Assembler Cataloged Procedures" in this manual.)

Severity Code: 16

IF0257 SYSPUNCH DD CARD MISSING -- NODECK OPTION USED

Explanation: A DD statement for the SYSPUNCH data set is not included in the JCL for this assembly. The SYSPUNCH data set is normally used when the object module of the assembly is directed to the card punch.

Assembler Action: The program is assembled using the NODECK option. No deck is punched on SYSPUNCH. If the OBJECT option has been specified, the object module will be written on the device specified in the SYSGO DD statement.

Programmer Response: Optional. The object module can be link edited and executed from SYSGO instead of SYSPUNCH by adjusting JCL. Otherwise, if you want a punch data set, reassemble the

program with a SYSPUNCH DD statement.

Severity Code: 16

IFO258 INVALID ASSEMBLER OPTION ON EXEC CARD -- OPTION IGNORED

Explanation: One or more of the assembler options specified in the PARM field of the EXEC statement are invalid. The error may be caused by use of the wrong option, a misspelled option, or syntax errors in coding the options.

Assembler Action: Invalid options are ignored. The assembly is performed using the valid options.

Programmer Response: Check the spelling of the options, the length of the option list (100 characters maximum), and the syntax of the option list. The options must be separated by commas, and parentheses in the option list (including SYSPARM) must be paired. Two quotes or ampersands are needed to represent a single quote or ampersand in a SYSPARM character string. The section "Assembler Options" in this manual describes the assembler options and how to code them.

Severity Code: 16

IFO260 ASSEMBLY TERMINATED -- DD CARD MISSING FOR SYSxxx

Explanation: This assembler job step cannot be executed because a DD statement is missing for one of the following assembler data sets: SYSUT1, SYSUT2, SYSUT3, or SYSIN. The missing DD statement is indicated in the message text.

Assembler Action: The assembly is terminated before any statements are assembled. No assembler listing is produced, so this message is printed on the system output unit following the job control language statements for the assembly job step and on the operator's console.

Programmer Response: Supply the missing DD statement and reassemble the program. The cataloged procedures supplied by IBM contain all the required DD statements. They are described in the section "The Assembler Cataloged Procedures" in this manual.

If the problem recurs, do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- If a COPY statement was used, execute the IEBPTPCH utility to obtain a copy of the partitioned data set member specified in the COPY statement.
- Make sure that MSGLEVEL=(1,1) was specified in the JCB statement.

Operator Response: If possible, supply the missing DD statement in the JCL statements for the assembly and run the job again.

Severity Code: 20

IF0261 ASSEMBLY TERMINATED -- PERM I/O ERROR jobname, stepname, unit address, device type, ddname, operation attempted, error description

Explanation: A permanent I/O error occurred on the assembler data set indicated in the message text. This message, produced by a SYNADAF macro instruction, also contains more detailed information about the cause of the error and where it occurred.

Assembler Action: The assembly is terminated. Depending on where the error occurred, the assembly listing up to the point of the I/O error may be produced. If the listing is produced, this message appears on it. If the listing is not produced, this message appears on the operator's console and on the system output unit following the job control language statements for the assembler job step.

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 the DD statement for the data set with the largest blocksize (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).

In any case, reassemble the program: it may assemble correctly. If the problem recurs, do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- If a COPY statement was used, execute the IEBPTPCH utility to obtain a copy of the partitioned data set member specified in the COPY statement.
- Make sure that MSGLEVEL=(1,1) was specified in the JCB statement.

Operator Response: If the I/O error is on SYSUT1, SYSUT2, or SYSUT3, allocate the data set to a different volume and rerun the job. If the I/O error is on tape, check the tape for errors.

Severity Code: 20

IF0262 ASSEMBLY TERMINATED -- INSUFFICIENT MAIN STORAGE

Explanation: The main storage allocated to the assembler is not enough for assembler tables, working storage, and/or utility file buffers.

Assembler Action: The assembly is terminated.

Programmer Response: Increase the size of the region or partition allocated to the assembler. Reassemble the program. If the problem recurs, do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- Make sure that MSGLEVEL=(1,1) was specified in the JCB statement.

Operator Response:

- Increase the size of the region allocated on the JOB card or on the EXEC card for the assembler job step and rerun the job; or
- Run the job in a larger partition.

Severity Code: 20

IFO263 ASSEMBLY TERMINATED -- PROGRAM LOGIC ERROR

Explanation: The assembly has been abnormally terminated because of a logic error within the assembler.

Assembler Action: Abnormal termination. No assembler listing is produced; the assembler prints this message on the system output device following the JCL statements for the assembler job step.

Programmer Response: Do the following before calling IBM:

- Have your source program, macro definitions, and associated listings available.
- If a COPY statement was used, execute the IEBPTPCH utility program to obtain a copy of the partitioned data set member specified in the operand field of the COPY statement.
- Make sure that MSGLEVEL=(1,1) was specified on the JOB statement.

Severity Code: 20

IFO264 TOO MANY ESD ENTRIES

Explanation: More than 399 entries have been made in the external symbol dictionary. Entries in the external symbol dictionary are made for the following: control sections, dummy sections, external references (EXTRN and WXTRN), ENTRY symbols, and external dummy sections.

Assembler Action: Entries over the 399 limit are not added to the dictionary and linkage is not provided for them.

Programmer Response: Subdivide your program and reassemble each section individually. Be sure that there are not more than 399 ESD entries in each assembly.

Severity Code: 16

IFO265 SYMBOL RESOLUTION DATA AREA HAS BEEN EXHAUSTED

Explanation:

- Too many literals have been encountered since a LTORG statement was encountered, and the assembler has filled available work space with literals; or
- The assembler has filled available work space with ESD entries.

Assembler Action: No assembly is performed.

Programmer Response:

- Insert more LTORG statements in the source deck or allocate more working storage to the assembler; or
- If there are more than 399 ESD entries in your source module, segment it into several modules.

Severity Code: 16

IFO266 LAST ASSEMBLER PHASE LOADED WAS xxxxxxxx

Explanation: This message is issued by the abort routine when the assembly is abnormally terminated.

Assembler Action: Abnormal termination.

Programmer Response: Correct problems indicated by other error messages and reassemble.

Severity Code: 4

IFO267 SYSPRINT DD CARD MISSING -- NOLIST OPTION USED

Explanation: The LIST option is specified, but the DD statement for the SYSPRINT data set is not included in the JCL for this assembly. The SYSPRINT data set holds the object module output of the assembly normally directed to the printer.

Assembler Action: The program is assembled using the NOLIST option. The message is printed on the system output device following the JCL statements for the assembler job step and on the operator's console.

Programmer Response: If you want a listing, reassemble the program with a SYSPRINT DD statement. Otherwise, do not specify the LIST option.

Operator Response: Supply, if possible, a SYSPRINT DD card for the assembler job step and rerun the job.

Severity Code: 16

IFC268 SYSTERM DD CARD MISSING - NOTERMINAL OPTION USED

Explanation: The TERMINAL option is specified, but the DD statement for the SYSTERM data set is not included in the JCL statements for this assembly. The SYSTERM data set contains diagnostic information output of the assembly, normally directed to a remote terminal.

Programmer Response: If you want a SYSTERM listing, reassemble the program with a SYSTERM DD statement. Otherwise, do not specify the TERMINAL option.

Operator Response: Supply, if possible, a SYSTERM DD card for the assembly step and rerun the job.

Severity Code: 16

IFO269 SYSLIB DD CARD MISSING

Explanation:

- A COPY instruction appears in the assembly, but no SYSLIB DD statement is included in the JCL statements; or
- An operation code that is not a machine, assembler, or source macro instruction operation code appears in the assembly, but no SYSLIB DD statement is included in the JCL statements. The assembler assumed the operation code to be a library macro operation code.

Assembler Action:

- The COPY instruction is ignored; or
- The operation code is treated as an undefined operation code.

Programmer Response: Supply the missing DD statement or correct the invalid operation code.

Severity Code: 16

Index

Indexes to systems reference library manuals are consolidated in the publication OS/VS Master Index, Order Number GC28-0602. For additional information on any subject listed below, refer to other publications listed for the same subject in the Master Index.

A

adding macro definitions to a library 50
ALGN option 22
ALIGN option 23
alignment of instructions and data (see ALIGN option)
ALOGIC option 18
ASMFC
  description 24-25
  example of use 13-14,32,33
ASMFCG
  description 30-31
  example of use 14-15
ASMFCL
  description 26-27
  example of use 33
ASMFCLG
  description 28-29
  example of use 32,33
assembler
  dynamic invocation of 70,22
  name of 16
  purpose 7
assembler cataloged procedures 23-33
assembler data sets 72-74
Assembler F, compatibility 8
assembler language 7
assembler listing 34-46
  cross reference 42-44
  diagnostics 45-46
  external symbol dictionary 36-37
  literal cross reference 44
  relocation dictionary 41
  source and machine language statements 38-40
  statistics 45-46
  symbol cross reference 42-43
assembler options
  description 15-23
  how to specify 16-17
assembler sample program 57-64
assembler speed and capacity 73
assembler storage requirements 73-74
assembler work space 22,73
assembly, JCL for (see ASMFC)
assembly and execution, JCL for (see ASMFCG; ASMFCLG)
assembly and link editing, JCL for (see ASMFCL)
assembly, link editing and execution, JCL for (see ASMFCLG)
assembly and loader-execution, JCL for (see ASMFCG)
ATTACH macro 70

B

base register, establishing 47
BLKSIZE for assembler data sets 73
blocking and buffering information 73
buffer size
  of SYSIN, SYSLIB, SYSPRINT, SYSGO, and SYSPUNCH 73
  of SYSUT1, SYSUT2, and SYSUT3 73
  (see also BUFSIZE option)
BUFNO for assembler data sets 74
BUFSIZE option 22,73

C

CALL macro 70
cataloged procedures
  description 12
  assembler 23-33
  (see also ASMFC, ASMFCG, ASMFCL, ASMFCLG)
  overriding parts of 32-33
changing parts of cataloged procedures 32-33
COBOL (see problem-oriented languages)
compatibility with Assembler F 8
COND parameter 80
conditional assembly statements in listing (see ALOGIC option, MLOGIC option)
conventions for linking 49
COPY instruction 50
cross reference listing 42-44

D

data sets, assembler 72-74
  SYSGO 73,74
  SYSIN 72,74
  SYSLIB 72,74
  SYSPRINT 71,72,74
  SYSPUNCH 73,74
  SYSTERM 73,74
  SYSUT1, SYSUT2, SYSUT3 73,74
data set characteristics, assembler 74
DD statements, overriding 32-33
DECK option 20
default values for assembler options 15
diagnostic messages
  explanations 78-141
  in listing 45-46
  special data set for (see SYSTERM listing)
  suppressing (see FLAG option)
  on terminal (see SYSTERM listing)
diagnostics 45-46
DOS option 23

dynamic invocation of assembler 70,22
dynamic invocation of IBM-supplied program 49

E

effective address 40
END card, object module 67
END instruction to specify entry point 48
entry point 48
error messages (see diagnostic messages)
ESD (see external symbol dictionary)
ESD card 65
ESD option 18
ESDID (external symbol dictionary identification number) 37,42
example of assembler language program 57-64
examples of cataloged procedures 13-15,32-34
EXEC statement, overriding parts of 32
execution of user program 8
external symbol dictionary 36-37

F

FLAG option 18
FORTRAN (see problem-oriented languages)

H

High-level language (see problem-oriented languages)

I

IEBUPDTE utility program 50
inner macro instructions, listing of (see MCALL option; MLOGIC option)
input to the assembler 7

J

JCL (job control language) 10
job 10
job control language 10
job control statements for assembler jobs 13-33
job step 10

L

LIBMAC option 19,78
library macro definitions
  adding to library 50
  errors in 45-46
    (see also LIBMAC option)
  listing of (see LIBMAC option)
library maintenance, macro 50
LINECNT option 23
LINECOUNT option 18
LINK macro 70
linkage conventions 47
linkage editor
  choosing entry point 48
  examples 32,33
  purpose 8
linkage registers 47
linking with modules produced by other language translators 48
linking with IBM-supplied programs 49
LIST option 18
listing control options 18-19
literal cross reference listing 44
literals in listing 38,44
load module 8
load module modification - entry point restatement 48
LOAD option 23
loader
  example of use 14-15
  purpose 8
location counter in listing 40
LRECL for assembler data sets 74

M

machine language code in listing 40
macro definitions, library (see library macro definitions)
macro instructions in listing, inner (see MCALL option; MLOGIC option)
macro library 50
  (see also SYSLIB)
MCALL option 18
messages (see diagnostic messages, statistics)
message identifier number 46,78
message text 46,78
MLOGIC option 18,79
MNOTE message 18

N

Name of assembler 16
NOALIGN option 22
NOALOGIC option 18
NODECK option 20
NOESD option 18
NOLIBMAC option 19
NOLIST option 18
NOMCALL option 18
NOMLOGIC option 18
NONUMBER option 21
NOOBJECT option 20
NORENT option 22
NORLD option 19
NOSTMT option 21,75
NOTERMINAL option 21,75
NOTEST option 20
NOXREF option 19
NUM option 21
NUMBER option 21,75

O

OBJ option 20
object code in listing 40
object deck output 65-69
  END 67
  ESD 65
  RLD 66-67
  SYM 67-69
  TXT 66

object module
  definition 7
  records of 65-69
object module linkage 48
OBJECT option 20
options, assembler (see assembler options)
options in listing 46
output control options 20
output from assembler 7
overriding parts of cataloged
 procedures 32-33

P

page size, assembler listing (see LINECOUNT
 option)
PARM field
  (see also assembler options)
  acquiring information in 49
  coding rules 16-17
  overriding in cataloged
   procedures 16,32
performance, influencing (see assembler
 speed and capacity)
PL/I (see problem-oriented languages)
PRINT instruction 38,40
problem-oriented languages, linking
 with 48
procedures, cataloged (see cataloged
 procedures)
program design 47
program listing (see assembler listing)

R

RECFM for assembler data sets 74
recurring errors 80
reentrability check 22
relocatable constants 41
relocation dictionary 41
RENT option 22
return code 80
RLD (see relocation dictionary)
RLD card 66-67
RLD option 19

S

Sample program 57-64
Save area 47
Saving registers 47
Severity code 80,46
source and machine language statements in
 listing 38-40
source module 7
source statement in listing 38-40
statement number 40,46,78
statistics listing 45-46
step 10
storage requirements, virtual 73
STMT option 21,75
SYM card 67-69
symbol cross reference listing 42-43
SYS1.MACLIB 50,72
SYSGO data set 73,74
SYSIN data set 72,74
SYSLIB data set 72,74
SYSPRINT data set 71,72,74
SYSPUNCH data set 73,74
SYSTERM data set 73,74
SYSTERM listing 75-77
SYSTERM options 21
SYSUT1, SYSUT2, SYSUT3 data sets 72,74

T

TERM option 21
TERMINAL option 21,75
terminal output (see SYSTERM listing)
TEST option 20
Time Sharing Option 75
TSO (see Time Sharing Option)
TXT card 66

U

use of assembler cataloged
 procedures 13-15,32-33
utility data sets 72,74
  (see also BUFSIZE option)

V

virtual storage requirements 73

W

work space, assembler 22,73

X

XCTL macro 70
XREF option 19

OS/VS Assembler Programmer's Guide (File No. S370-21 (OS/VS)) Printed in U.S.A. GC33-4021-1

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS Assembler
Programmer's Guide

Order No. GC33-4021-1

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such request, please contact your IBM representative or the IBM Branch Office serving your locality.*

CUT ALONG DOTTED LINE

Reply requested:
Yes ☐
No ☐

Name: _____
Job Title: _____
Address: _____
_____ Zip _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department 813 L
1133 Westchester Avenue
White Plains, New York 10604

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

CUT OR FOLD ALONG LINE

OS/VS Assembler Programmer's Guide (File No. S370-21 (OS/VS)) Printed in U.S.A. GC33-4021-1