GC33-4024-1
File No. S370-21(DOS/VS)

Systems

Guide to the DOS/VS Assembler

IBM

Second Edition (September 1973)

This is a major revision of, and obsoletes GC33-4024-0. This edition incorporates minor technical and editorial changes. Changes to the text and to illustrations are indicated by a vertical line to the left of the change.

This edition applies to version 5 of the Disk Operating System, DOS/VS, and to all subsequent versions and releases until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822, 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.

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

Forms are provided at the back of this publication for reader's comments. If the forms have been removed, comments may be addressed to IBM Nordic Laboratory, Programming Publications, Box 962, S-181 09 Lidingo 9, Sweden. Comments become the property of IBM.

This Manual...

...shows how to write job control language (JCL) statements needed to assemble, link-edit, and execute a program written in the DOS/VS assembler language.

...shows how to maintain the macro and copy libraries.

...shows how to de-edit and update macros.

...shows how to interpret the listings produced by the DOS/VS Assembler.

...explains the files used by the DOS/VS Assembler.

...explains the object-deck output.

...explains all messages issued by the assembler and de-editor programs.

Audience For This Manual

- assembler language programmers
- system programmers responsible for JCL and the maintenance of the DOS/VS system files
- FEs and CEs or system programmers who want to de-edit and update macros, and who want information on how the assembler libraries are maintained.

Level of Knowledge Required For This Manual

- a basic understanding of the DOS/VS operating system as described in Introduction to DOS/VS, Order No. GC33-5370.
- a good understanding of the DOS/VS assembler language as described in OS/VS and DOS/VS Assembler Language, Order No. GC33-4010.

Related Manuals

DOS/VS System Control Statements, Order No. GC33-5376.
IBM System/360 and 370 Bibliography, Order No. GA22-6822.
IBM System/370 Advanced Function Bibliography, Order No. GC20-1763.

Summary of Contents

The following descriptions summarize the contents of the major sections of this manual.

SECTION TITLE	DESCRIPTION
INTRODUCTION	Shows the purpose of the assembler and introduces the reader to the basic concepts covered in this manual.
HOW TO WRITE JOB CONTROL LANGUAGE	Shows how to prepare the basic job control statements for assembling, link editing, and executing a program written in the assembler language.
MAINTAINING THE MACRO AND COPY LIBRARIES	Shows how to maintain macro and copy libraries, that is, how to add, delete, or update statements in macro definitions. It also shows how to convert old macros to edited format.
DE-EDITING AND UPDATING MACROS: ESERV PROGRAM	Shows when and how to use the de-editor program and how to combine the function of de-editing with that of updating.
INTERPRETING THE ASSEMBLER LISTING	Shows how to interpret the six parts of the assembler listing produced by the assembler program.
STORAGE REQUIREMENTS	Gives the minimum main storage requirements and explains how to estimate auxiliary storage requirements.
CONFIGURATION SPECIFICATIONS	Lists the required and optional hardware for use with the DOS/VS Assembler.
FILES USED BY THE ASSEMBLER	Explains what each file contains and what it is used for.
OBJECT DECK OUTPUT	Gives format and contents of the cards that make up the output deck produced by the assembler.
DIAGNOSTIC AND ERROR MESSAGES	Explains each message produced by the assembler and by the de-editor (ESERV) program.
GLOSSARY	Contains definitions of all terms specific to the assembler and not included in the IBM Data Processing Glossary and OS/VS and DOS/VS Assembler Language, Order No. GC33-4010.

Contents

Figures

Introduction

This section describes the purpose of the DOS/VS Assembler, its relationship to the operating system, and its input and output. It also explains the concept of edited macros, a feature of the DOS/VS Assembler.

Purpose of the Assembler

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

Relationship of the Assembler to the Disk Operating System

The assembler is supplied with the DOS/VS control program package. In the same way as the linkage editor, it is executed under control of the DOS/VS 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 DOS/VS.

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 can consist of an object module, edited macros and program listing. (The concept of edited macros in the DOS/VS Assembler is explained below.) The object module can either be punched or included in a file residing on a direct-access device or a magnetic tape. From that file the object module can be read into the computer and processed by the linkage editor. The format of the object module is described in the section "Object Deck Output".

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 "Interpreting the Assembler Listing".

Compatibility

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

Concept of Edited Macros

Edited macros are source macros that have been partially processed by the assembler and stored in a new macro sublibrary within the source statement library. These definitions, being edited, can be assembled more quickly, thus reducing total processing time.

You can keep macro definitions in source format on the copy library or in edited format on the macro library. Quite often, you will want to change a macro definition; to add or delete a statement from it, for instance. The coding you use to perform this maintenance work varies according to which library you have used. The reason for this variation in coding stems from the manner in which the assembler program handles macros prior to placing them on the copy or macro library. See the section "Maintaining the Macro and Copy Libraries" for a discussion of the circumstances under which you might choose either the macro or the copy library, or both, to contain your macro definitions.

Before placing a macro definition on the macro library, the assembler first partially processes it. This processing is called editing. In the past, this time-consuming editing function was performed each time you called a macro definition into your program. Now, under DOS/VS, it is performed only once. Should, however, you want to add to or delete from or somehow change a macro definition, you cannot use the edited version of the macro definition that you have on the macro library. Instead, you must change a copy of the non-edited (that is, source) macro definition. If you do not have a non-edited copy of the macro definition, then you can convert the edited macro back to its source format. This conversion process is called de-editing and is described in the section "De-Editing and Updating Macros: ESERV Program".

No editing step is involved when the assembler puts a macro definition on the copy library. That is, a macro definition remains in source format. You can use this copy of the macro definition when you wish to do maintenance work on it. You can then edit it and place it on the macro library, and also place the non-edited version of it back on the copy library. Thus at any one time you have a source and edited version of the macro definition on the copy and macro library, respectively.

Edited Macros and DOS/360 Users

All users who wish to use old macro libraries created by previous DOS assemblers must convert these libraries of unedited macros to edited format. An example of one method of this process is shown in the subsection "How to Convert an Old Macro Library".

How to Write Job Control Language Statements

Purpose of This Section

This section shows by examples how to prepare job control language (JCL) statements to assemble, link-edit, and execute a program written in assembler language.

Foldout: Use the foldout at the end of this section as you read. You will find on it a summary of the rules governing the writing of JCL and linkage editor control statements, along with brief definitions of the JCL and linkage editor statements used in the examples. For a full coverage of JCL, consult the publication DOS/VS System Control Statements.

Assembly

The following example shows the job control statements you need in order to assemble a source module and produce an object module.

// JOB ANYNAME	Initiates the job ANYNAME.
// OPTION DECK,NOLINK	Causes the object deck to be punched on SYSPCH and not to be copied on SYSLNK. Default values determined when the system was generated are used for the other options.
// EXEC ASSEMBLY	Causes the assembler to be loaded from the core image library into main storage, and to start executing.
source module	The assembler source module.
/*	Delimits the input to the assembler.
/&	Delimits the job.

There are no // ASSGN statements in this example; therefore, the assembler uses the standard assignments for the source statement library, SYSIPT, and SYSPCH that were set up during system generation. Figure 1 on the next page shows the data flow of this example.

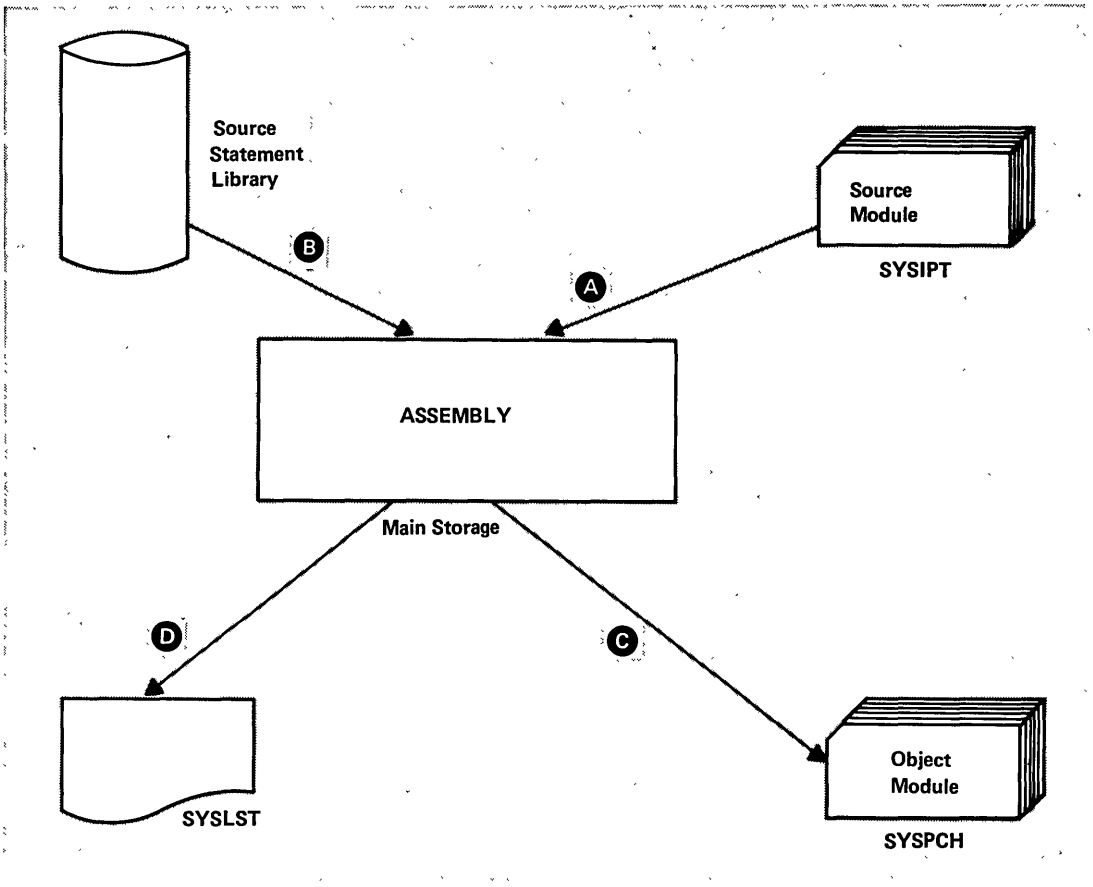


Figure 1. Data Flow When Assembling

Explanation:

A. The assembler reads the source module into main storage for processing in the assembly step.

B. It also reads macro definitions and copy code sequences, if any, from the source statement library, and

C. Punches the object deck on SYSPCH.

D. Prints the list on SYSLST.

Assembly and Link Editing

EXAMPLE 1

This example expands the previous example by adding to it a linkage editor step.

The job first assembles the source module and then link-edits the object module produced in the assembly step together with another previously assembled object module and catalogs the linkage editor output in the core image library under the name EXAMPLE.

// JOB TWOMODS	Initiates the job TWOMODS
// OPTION CATAL	Causes the assembler to store the object module on SYSLNK and, later, causes the linkage editor to catalog the phase it produces in the core image library. Default values determined when the system was generated are used for the other options.
PHASE EXAMPLE,S	Provides the linkage editor with the name of the phase (EXAMPLE) and the main storage address (S) where it is loaded (immediately after the supervisor).
// EXEC ASSEMBLY	Causes the assembler to be loaded into main storage and to start executing.
source module	The assembler source module.
/*	Delimits the input to the assembler.
INCLUDE	Causes the object deck that follows in the input stream to be transferred to SYSLNK and included in the linkage editor input.
object module	The object module to be included.
// EXEC LNKEDT	Causes the linkage editor to be loaded into main storage and to start execution.
/&	Delimits the job.

There are no // ASSGN statements in this example; therefore, the assembler uses the standard assignments set up during system generation.

Figure 2 shows the data flow of this example.

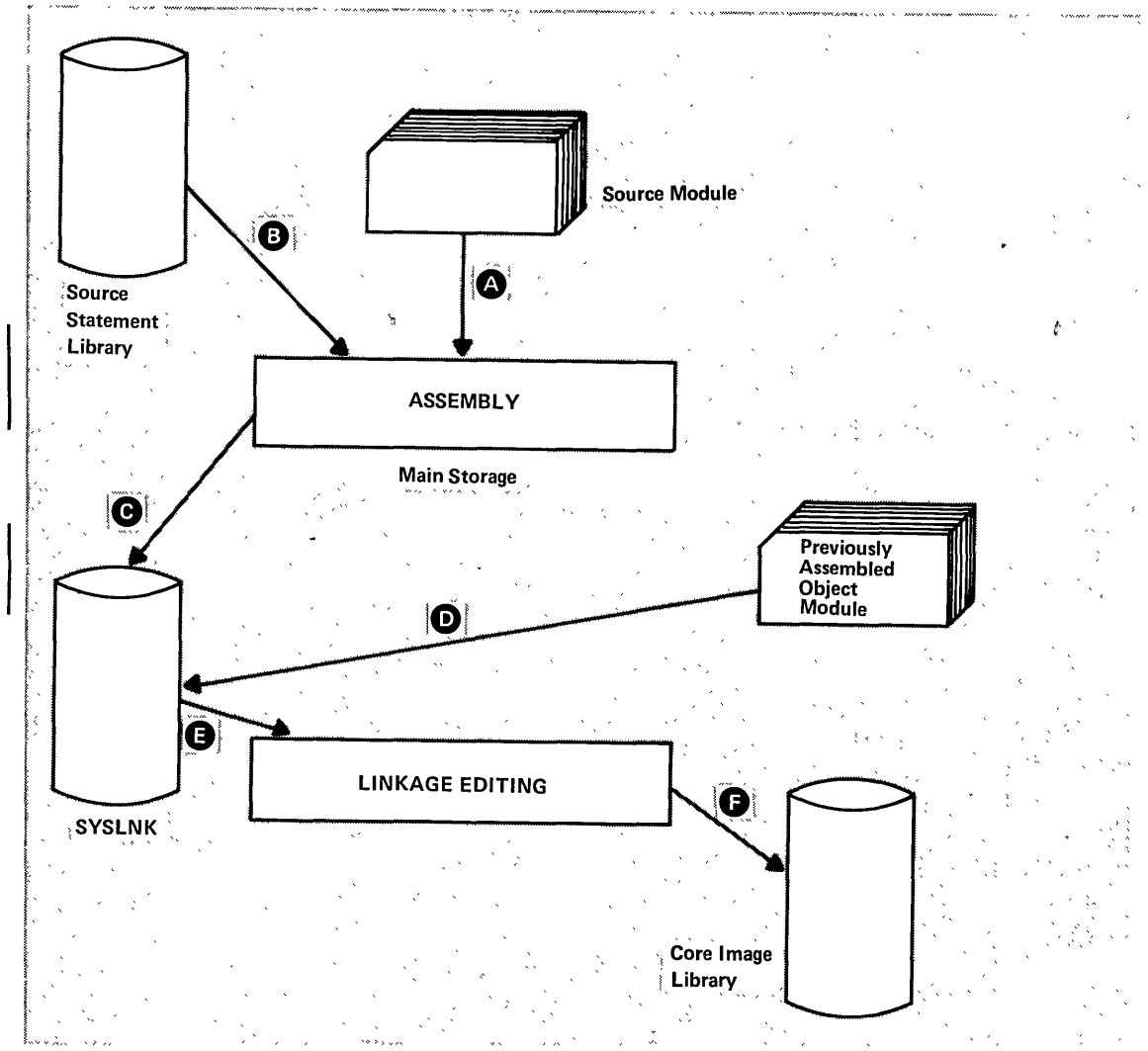


Figure 2. Data Flow When Assembling and Link Editing

Explanation:

A. The assembler reads the source module into main storage for processing in the assembly step.

B. It also reads macro definitions and copy code sequences, if any, from the source statement library, and

C. Copies the output from the assembler -- the object module -- on SYSLNK.

D. Job control transfers the object deck in the input to SYSLNK for processing by the linkage editor.

E. Linkage editor reads the object modules from SYSLNK into main storage for processing.

F. The phase produced by the linkage editor is cataloged into the core image library.

EXAMPLE 2

The following example illustrates another method for setting up a job for assembly and link editing. In addition, the program is executed in the same job.

In the example, three source modules (A) and one object module (B) have been loaded on an unlabeled 9-track magnetic tape in a previous job; an additional object module is on the relocatable library.

The contents of the tape are as follows:

```
ALPHA       CSECT
              .
              .
              .
            END
/*
BRAVO       CSECT
              .          } (A)
              .
              .
            END BRAVO
/*
CHARLIE     CSECT
              .
              .
              .
            END
/*
     (object module)      (B)

/*
     (data for problem
      program)
/*
/&
```

The coding is as follows (see next page):

```
// JOB FIVEMODS
// ASSGN SYSIPT,X'182'
// OPTION LINK,LIST,NODECK
   PHASE EXAMPLE,S
// EXEC ASSEMBLY
// EXEC ASSEMBLY
// EXEC ASSEMBLY
   INCLUDE
   INCLUDE MOD24
   ENTRY
// EXEC LNKEDT
// EXEC
/&
```

Assign to SYSIPT the tape mounted on tape drive 182.

The object modules are to be link-edited (LINK) directly. The output will be on SYSLNK (LINK), the source module listing will be on SYSLST (LIST), and the assembler will not put the object modules on SYSPCH (NODECK).

The phase name (EXAMPLE) and the load address (S) of the phase.

The three source modules are assembled.

The object module on SYSIPT is to be included in the linkage editor input.

The object module (MOD24) on the relocatable library is to be included in the linkage editor input.

As the operand of this ENTRY statement is blank, the linkage editor searches the modules in the input and picks the first primary entry point specified in an assembly END statement. In this example, the statement labeled BRAVO is chosen as the entry point of the whole phase.

The program is link-edited.

The program is executed. The blank operand causes the program which has just been link-edited to be executed.

Execution

The following statements show how to execute the program that was included in core image library in Example 1.

```
// JOB EXECEXAM          Initializes the job.

// EXEC EXAMPLE          Causes problem program (EXAMPLE) to be
                         loaded into main storage and to start
                         executing.

   .
   data                  Data for problem program, if any.
   .

/*                       Delimits data.

/&                       Delimits the job.
```

Assembler Options and JCL Summary

NOTE: The information given below is intended only as an aid to memory. A full discussion of job control can be found in the publication System Control Statements.

JOB CONTROL STATEMENTS

The following rules apply when filling out control statements:

1. Two slashes (//) identify the statement as a control statement. They must be in columns 1 and 2. At least one blank must follow the second slash. The end-of-job statement contains /& in columns 1 and 2. The end-of-data statement contains /* in columns 1 and 2. The comments statements contain an * in column 1, and a blank in column 2.

2. Operation. This describes the operation to be performed. It can be up to eight characters long. At least one blank follows the last character.

3. Operand. Can be blank or contain three or more entries separated by commas.

// JOB	Indicates the beginning of the control information for the job. Obligatory first card.
// ASSGN	Assigns physical I/O device addresses to data files.
// DLBL	Contains information the job control program needs to write and check label information on a direct access device. Must be followed by one or more EXTENT cards.
// EXTENT	Defines each area (extent) of a direct access file.
// TLBL	As above, but for tape devices.
// OPTION	Specifies one or more JCL options. For details of options see inner page of this foldout.
// EXEC	Indicates the end of job and linkage-editor controls cards for a job step, and gives control to a processing program.

LINKAGE EDITOR CONTROL STATEMENTS

The rules listed above for JCL statements are similar to those for Linkage Editor control statements except that the latter do not have slashes (//) preceding them: these statements begin after column 1, which must be a blank.

PHASE	Indicates the beginning of a phase. The linkage editor can link-edit several phases in one step, in which case a PHASE card must precede the input for each phase.
INCLUDE	Indicates that a module or some control sections (CSECT) from a module located in the relocatable library or in SYSIPT are to be included in the linkage editor input.
ENTRY	Specifies the entry point of the first phase produced in a linkage editor step.

Figure 3. Assembler Options and JCL Summary

OPTION	DESCRIPTION
LIST*	Causes the assembler to write the source module listing on the output device assigned to SYSLST.
NOLIST	Suppresses the LIST option and overrides the ESD, RLD and XREF options.
LINK	Causes the program being assembled to be link-edited in the same job. This option (or the CATAL option - see below) must be used when assembling, link-editing, and executing in the same job. LINK causes the linkage editor to write its output (phases) temporarily on the core image library. Use of this option decreases processing time. If CATAL is specified, the LINK option is automatically set.
NOLINK	Suppresses the LINK option.
CATAL	Causes the program being assembled to be link-edited in the job and stored permanently in the core image library. LINK - see above - performs the same function except that it stores the object module temporarily on the core image library. Thus, CATAL or LINK must be used when assembling, linkage editing, and executing in the same job. Use of the CATAL option decreases processing time.
DECK*	Causes the assembler to place the object modules on the output device assigned to SYSPCH. Used when a back-up copy of the object code is needed.
NODECK	Suppresses the DECK option.
EDECK	Causes the source macros in the program to be punched in edited format on the output device assigned to SYSPCH. This option is used to punch the macros for later cataloging into the macro library.
NOEDECK	Suppresses the EDECK option.
XREF*	Causes a cross-reference list to be printed on the output device assigned to SYSLST. See section "Interpreting the Assembler Listing" for a sample of the cross-reference list produced by this option. The option is used primarily as a debugging tool.
NOXREF	Suppresses the XREF option.
ALIGN	Causes all data to be 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. Use of this option decreases the execution time of the phase that results from assembling and link-editing.
NOALIGN	The assembler does not align data areas other than those specified in CCW instructions, nor does it skip bytes to align constants on proper boundaries. Alignment violations in machine instructions are not diagnosed.
SYSPARM= string	Specifies the value for assembler system variable symbol, &SYSPARM.

* Unless otherwise specified, the LIST, DECK and XREF options are default options of the DOS/VS system. Some of the other options listed above may have been made standard features at system generation: you are advised therefore, to familiarize yourself with your system's standard assignments.

Maintaining the Macro and Copy Libraries

Introduction

The assembler uses two sublibraries of the source statement library; the macro library (sublibrary E) and the copy library (sublibrary A).

What Is the Macro Library?

The macro library contains IBM-supplied system macro definitions and user-written macro definitions in an edited (partially processed) format. This library contains only edited macro definitions.

The assembler edits the macro definition you have included in a source module and will produce an edited macro definition which it puts on tape or disk when you specify the EDECK option in the OPTION statement. This output may later be cataloged into the macro library.

What Is the Copy Library?

The copy library contains sequences of source code (books) which you can insert into the source module by writing one or more COPY statements. These books can contain any kind of source code, including source (un-edited) macro definitions.

Which Library to Use for Macro Definitions?

- Often-used macro definitions go, in edited format, on the macro library because you save assembly time by having them there.
- Keep a backup copy of these macro definitions, in source (un-edited) format, on the copy library, or in card format. You will need this source copy when you wish to update a macro definition you have on the macro library. That is, you cannot update an edited macro. If you do not have a backup source copy you must first de-edit the macro definition on the macro library. See the section "De-Editing and Updating Macros: ESERV Program".
- While debugging, keep your macro definition on the copy library. When free of errors, place it on the macro library.

Figure 4 summarizes this discussion and acts as a table of contents for this section.

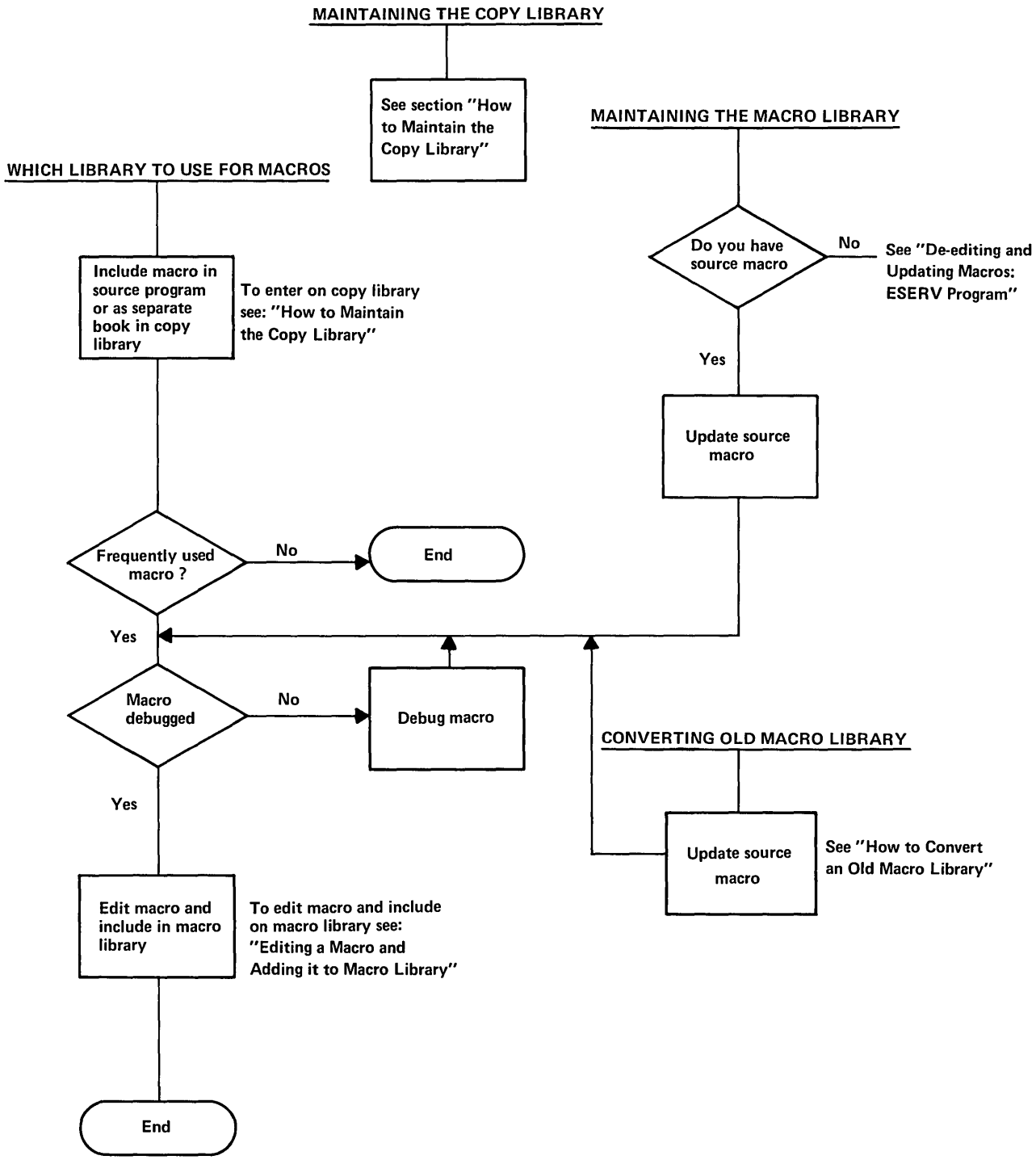


Figure 4. Choosing a Library for Macro Definitions

How to Maintain the Macro Library

EDITING A MACRO AND ADDING IT TO THE MACRO LIBRARY

The assembler reads in and assembles the source macro definition(s) from either the copy library or from a source deck or tape assigned to SYSIPT. If you specified EDECK in the OPTION statement the assembler will direct the one or more resulting edited macro definitions to the output device assigned to SYSPCH. Each edited macro will be preceded by an assembler-supplied CATALS statement in which the level of the assembler is printed immediately after the macro name.

Then from the resulting card deck, tape file, or disk file, you can place the edited macro definitions into the macro library using the MAINT program.

EXAMPLE 1.

Editing a macro and, using tape as intermediate storage, adding it to macro library.

// JOB	
// ASSGN SYSPCH,X'180'	Assigns SYSPCH (output device for edited macros) to tape, at unit address 180.
// OPTION EDECK,NODECK	The EDECK option has a macro deck punched; the NODECK option suppresses the punching of the object module.
// EXEC ASSEMBLY	
(Macro definition to be edited)	The macro to be edited and placed on library.
/*	
// CLOSE SYSPCH,X'00D'	Writes tape mark, rewinds tape, and unloads the reel.
// ASSGN SYSIPT,X'180'	Tape with macro edited deck used as input.
// EXEC MAINT	Puts edited macro on macro library.
/&	

EXAMPLE 2.

Editing a macro and, using disk as intermediate storage, adding it to the macro library.

```
// JOB PUNCH                          See the publication System
// DLBL IJSYSPH,'PCHFILE',0           Control Statements for full
                                      description of these statements.

// EXTENT SYSPCH,111111,,,1300,500    Assigns SYSPCH (output device
// ASSGN SYSPCH,X'391'                for edited macros) to disk at
                                      unit address 391.

// OPTION EDECK,NODECK
// EXEC ASSEMBLY
   (macro definition to
   be edited)
   END
/*
CLOSE SYSPCH,X'00D'
// DLBL IJSYSIN,'PCHFILE'
// EXTENT SYSIPT
ASSGN SYSIPT,X'391'
// EXEC MAINT
/&
CLOSE SYSIPT,X'00C'
```

Deleting Macro from Macro Library

```
// JOB
// EXEC MAINT                         MAINT program used to delete.
   DELETS E.MAC1                      Deletes MAC1 (name of macro) from
                                      the macro library (E).
/*
/&
```

Updating Macro Definitions That Are on Macro Library

You update a library macro definition by making changes in the source macro definition (on cards, tape, etc.) or from the copy library, as discussed below. If you do not have the macro definition in source format, you must use the ESERV program to de-edit your macro before updating (discussed in the section "De-Editing and Updating Macros: ESERV Program").

UPDATING MACRO DEFINITIONS ON MACRO LIBRARY: FROM A SOURCE DECK

Update source deck by hand for this macro definition, then replace the old macro with this updated source macro to the macro library using the coding shown in the section "Editing a Macro and Adding It to the Macro Library".

UPDATING MACRO DEFINITIONS ON MACRO LIBRARY: FROM COPY LIBRARY

Update the source macro on the copy library. Then, assemble it and use the EDECK option. Finally, replace the old macro with this updated macro on the macro library using the MAINT program, as follows:

```
// JOB UPDATMAC
// EXEC MAINT                    Use MAINT to update source macro.
   UPDATE A.MAC1                 The macro to be updated is MAC1 and
                                 is on the copy library (A).
                                 A ")" in the first column identifies
                                 these statements as update control
                                 statements.
) DEL 0011                       Deletes statements 0011
) END
/*
// ASSGN SYSPCH,X'180'           Assign tape 180 as intermediate
                                 medium.
// OPTION EDECK,NODECK           Edited macro punched on SYSPCH
                                 (EDECK).
// EXEC ASSEMBLY
       COPY MAC1                 Bring in MAC1 from copy library.
       END
/*
// CLOSE SYSPCH,X'00D'           Writes tape mark, rewinds tape, and
                                 unloads the reel.
// ASSGN SYSIPT,X'180'           Tape 180, with macro edited deck,
                                 assigned as input.
// EXEC MAINT                    Place edited macro on macro library.
/&
```

How to Convert an Old Macro Library

All macro definitions on the macro library must be in edited format. To convert a library of non-edited macros created by previous DOS assemblies:

Edit source copies of macro definitions that are on the library in a regular assembly run, using the EDECK option (defined in Figure 3) on the OPTION statement, to produce an edited macro deck. These edited decks are then placed on the macro library using the MAINT program.

The source macros can be either in the copy library or in source deck format. The following code shows one method for converting an old macro library to a library of edited macros.

```
// JOB EPUNCH                          Name of job is EPUNCH.

// ASSGN SYSPCH,X'180'                 Assigns this tape for edited
                                       output.

// DLBL IJSYSSL,'USERS PRV MACROS'     File name of user's private
                                       macro library is IJSYSSL.

// EXTENT SYSSLB,xxxxxx                Its volume serial number is from
                                       one to six characters in length.

// ASSGN SYSSLB,X'192'                 SYSSLB must be assigned if source
                                       code is held in private source
                                       statement library.
// OPTION EDECK,NODECK                 Only the edited macro deck will
                                       be punched.
// EXEC ASSEMBLY
    COPY MAC1                          Bring in all macros that you wish
    COPY MAC2                          to include in the new macro
      .                                library.
      .
      .
    END
/*
// CLOSE SYSPCH,X'00D'                 Writes tape mark, rewinds tape, and
                                       unloads the reel.
/&
// JOB CATALOG                         This job is named CATALOG.
// ASSGN SYSSLB,X'193'                 And will be cataloged here
// DLBL IJSYSSL,'USERS PRV EDMACS'     Under this name
// EXTENT SYSSLB,'xxxxxx'              With this serial number.

// ASSGN SYSIPT,X'180'                 Assign tape containing edited
                                       output as input.

// EXEC MAINT                          Place edited macros on macro
                                       library.
/&
```

Note: If a macro in the macro library contains copy code and the copy code is updated, then the macro has to be re-edited to get the new version of the copy code.

How to Maintain the Copy Library

The copy library is maintained in the same way as any source code library in current DOS/VS using the MAINT program, as follows:

ADDING MACRO DEFINITIONS

```
// JOB COPY
// EXEC MAINT                    MAINT used to update.
   CATALS A.MAC1                 Catalogs macro MAC1 in copy library (A)
   MACRO
   MAC1                          This macro is cataloged.

   MEND
/*
/&
```

DELETING MACRO DEFINITIONS

```
// JOB DEL
// EXEC MAINT                    MAINT used to update.
   DELETS A.MAC1                 Deletes macro MAC1 from copy library (A)
/*
/&
```

UPDATING A BOOK

```
// JOB UPDATE
// EXEC MAINT                    MAINT used to update.
   UPDATE A.MAC1                 Updates macro MAC1 on copy library (A).
) REP 0011                       Replace statement 0011 with
    CLC FIELD1,BLANKS            This statement and
) ADD 0060                       Add after statement 0060
PUNCH MVI SWITCH,SW1             These statements.
      B   CHKOPND
) DEL 0601,0632                  Delete statements 0601 through 0632.
) END
/*
/&
```

Another way to update a book in the copy library is to have it punched update it manually, and then catalog it again in the copy library:

```
// JOB PUNCH                     Name of job is PUNCH
// EXEC SSERV                    Initiates execution of SSERV
                                 (see publication System Control
                                 Statements for details of the
                                 SSERV program).

 PUNCH A.MAC1                    Causes the book MAC1 to be punched
                                 out on cards from the copy library
                                 (A).
```

Once the book has been updated you can use the coding shown under "Adding Macro Definitions" above, to re-catalog into the copy library.

How to Use Edited and Un-Edited Macro Definitions

1. If the macro is edited, you just write your macro instruction:

```
MAC PARM1,PARM2
```

2. If the macro is not edited, that is, it is on the copy library, you must bring the definition into your source module at the beginning of the module in which it is called.

```
COPY MAC        Must be ahead of all other statements
    .           in the assembly.
    .
    .
MAC PARM1,PARM2
```

De-Editing and Updating Macros: ESERV Program

Introduction

Before you can put a macro definition on the macro library the assembler has to edit it and, using the EDECK option, produce an edited macro deck. The edited deck is placed on the macro library using the MAINT program. An edited macro definition cannot be directly updated; instead, the source macro definition, either in card format or on the copy library, is updated. After the changed macro definition has been tested, debugged, and edited, it can be placed on the macro library.

If your source macro definition is not available, the de-editor program, ESERV, can be used to de-edit the edited macro definition to source format. Several macros can be handled at one time. The de-editor program also combines the function of de-editing with that of updating the source macro definition.

Input to the ESERV Program

- ESERV control statements on SYSIPT.
- Edited macro definitions in the macro library.

Output from the ESERV Program

- The selected macro definitions in source format (and updated) on the device assigned to SYSPCH, and/or the device assigned to SYSLST. To allow immediate editing of the updated macro an END card can be generated at the end of the update run. (See "Using ESERV to De-Edit and Update a Macro Definition".)
- Update information (see following diagram for details).
- Error Diagnostics (see "Diagnostic and Error Messages", beginning at message IPK301).

The following diagram graphically illustrates the input and output of the ESERV program. Included in the diagram is a list of the control statements for the ESERV program, and some explanatory notes on the update information.

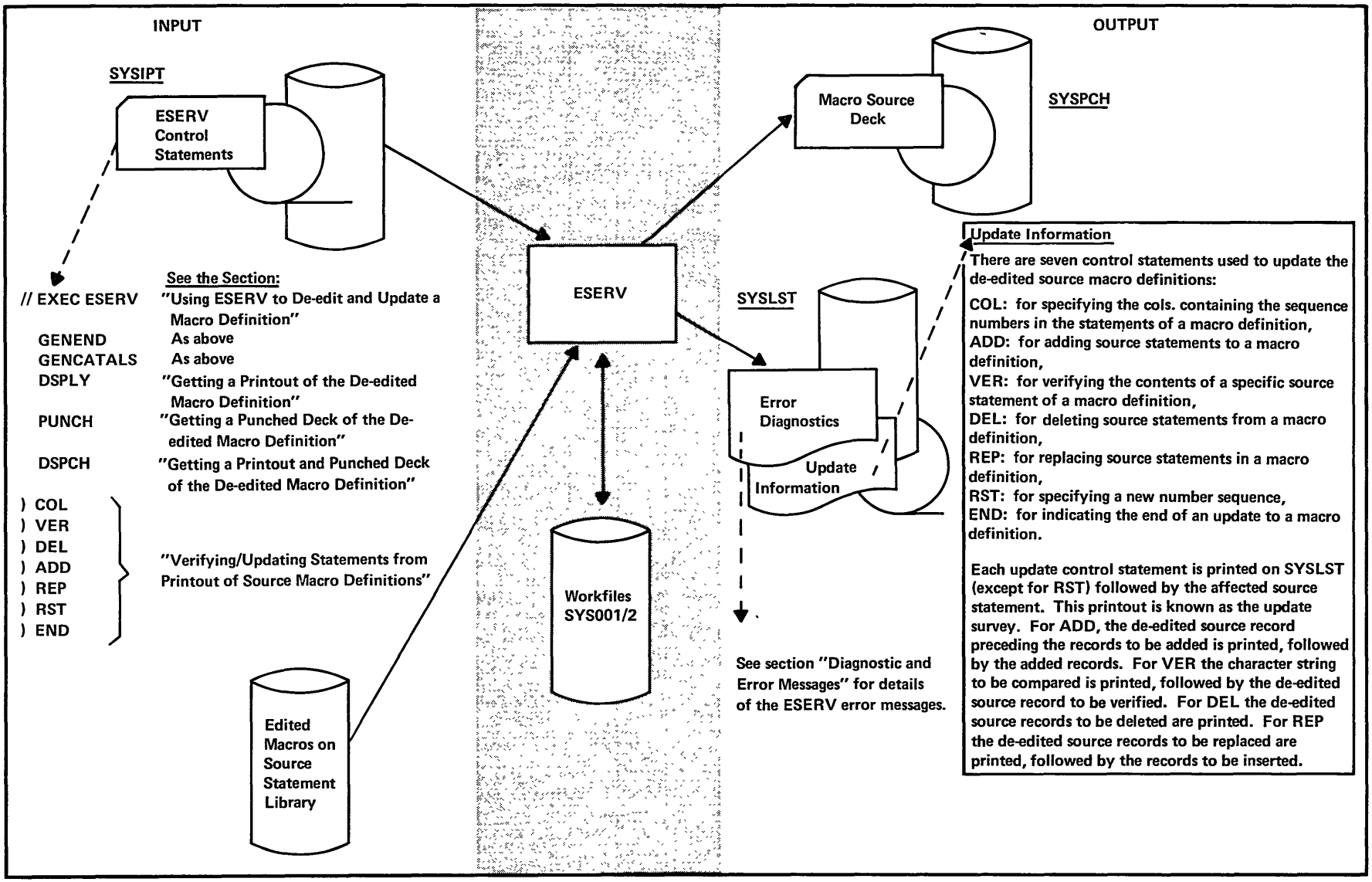


Figure 5. Input to, and Output from the ESERV Program

To request the ESERV program, use the following EXEC control statement:

```
// EXEC ESERV
```

To allow you immediately to use the output from the ESERV program as assembler input, you need an assembler END statement, plus a /* statement. You generate these by using the GENEND statement, the format of which is:

```
Column 2—▼
         GENEND
```

GENEND, when present, must be the first statement after the // EXEC ESERV statement.

To allow you immediately to use the output from the ESERV program as SYSIPT for cataloging into the macro library you need an appropriate CATALS statement before each macro in the run, and a /* statement after the last macro. You generate these by using the GENCATAL statement, the format of which is:

```
Column 2—▼
         GENCATALS
```

GENCATALS, when present, must be the first statement after the // EXEC ESERV statement. If neither GENEND or GENCATALS is used, GENCATALS is assumed.

Before you can update your source macro definition, you need

(a) a printout of the source macro definition, or
(b) a source deck of the source macro definiton.

The following paragraphs show how you can get a printout of the source macro definition, or a source deck of the source macro definition, or both.

GETTING A PRINTOUT OF THE DE-EDITED MACRO DEFINITION

DSPLY - Display

The DSPLY (display) control statement produces a printout of the de-edited macro on the device assigned to SYSLST. DSPLY has the following format:

```
Column 2—▼

          DSPLY sublib1.mac1,sublib2.mac2,...
```

If the qualifier sublib is omitted the macro library (E) is assumed by default. The mac in the operand field represents the name of the macro definition in the sublibrary. If more than one macro definition is to be displayed the entries must be separated by commas.

If update of the de-edited macro definition is desired the appropriate control statements (see "Verifying/Updating Statements from Printout of Source Macro Definition") follow this statement. However, only the last macro can be updated.

GETTING A PUNCHED DECK OF THE DE-EDITED MACRO DEFINITION

Punch - PUNCH

The PUNCH function produces a de-edited deck on the device assigned to SYSPCH. The formats and their governing conditions are the same as for the DSPLY function defined above.

GETTING A PRINTOUT AND PUNCHED DECK OF THE DE-EDITED MACRO DEFINITION

Display and Punch - DSPCH

The display-and-punch function combines the separate operations of the display function and the punch function. The formats and their governing conditions are the same as for the DSPLY function defined above.

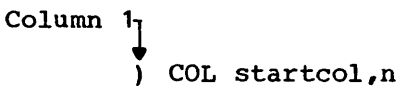
Verifying/Updating Statements from Printout of Source Macro Definition

You can use the control statements COL, VER, ADD, DEL, REP, RST, and END with the ESERV program to verify and/or update properly identified statements in an edited macro definition. Statements are identified by the sequence number in the identification field. Statements without sequence numbers are identified by their position relative to a previous statement that has a sequence number.

The verifying and updating is performed on the last macro specification in the preceding DSPLY, PUNCH, or DSPCH statement.

COL Statement

The COL statement is used to specify the columns containing the sequence numbers in the statements of a macro definition. If present, this statement must be the first control statement following the DSPLY, PUNCH or DSPCH statement. The COL statement has the following format:



startcol is a decimal number within the range 73-80, which identifies the start column of the sequence number. n is a decimal number within the range 1-8, specifying the number of columns used by the sequence number.

If the COL statement is omitted, startcol receives a default value of 73 and n a value of 6.

VER Statement

The VER statement is used when the contents of a specified source statement of a macro definition is to be verified. The format is:

seqno+rel is used to identify the source statement which is to be compared with the statement following the VER statement.
seqno represents the sequence number of a source statement. It is a decimal number of 1 to n digits in length, where n is the length of the sequence field as specified in the COL statement.
rel is a decimal number of 1-4 digits in length. If omitted, rel receives a default value of 0.
l is a decimal number within the range 1-80. When omitted, l gets a default value of 72. Only the first l characters are used in the comparison. If the strings do not match, an error message is given.

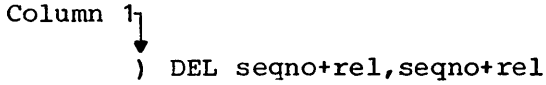
ADD Statement

The ADD statement is used to add statements to a source macro definition. The format is:

seqno+rel is used to identify the source statement after which the new statements following the ADD statement are to be inserted.
seqno represents the sequence number of a source statement. It is a decimal number of 1 to n digits in length, where n is the length of the sequence field as specified in the COL statement.
rel is a decimal number of 1-4 digits in length. If omitted, rel receives a default value of 0.

DEL Statement

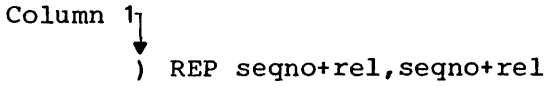
The DEL statement is used to delete statements from a macro source definition. The format is:



seqno+rel,seqno+rel are used to identify the first and last source statements of the section to be deleted. If second operand is omitted, only the source statement identified by the first operand is deleted.
seqno represents the sequence number of the source statement. It is a decimal number of 1 to n digits in length where n is the length of the sequence field as specified in the COL statement.
rel is a decimal number of 1-4 digits in length. If omitted, rel gets a default value of 0.

REP Statement

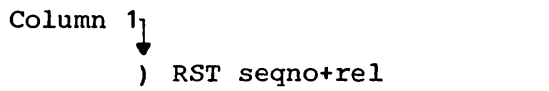
The REP statement is used to replace statements in a source macro definition. The format is:



seqno+rel,seqno+rel are used to identify the first and last source statements of the section which are to be replaced by the statements following the REP statements. If the second operand is omitted, only the source statement identified by the first operand is replaced.
seqno represents the sequence number of a source statement. It is a decimal number of 1-n digits, where n is the length of the sequence field as specified in the COL statement.
rel is a decimal number of 1-4 digits in length. If omitted, rel receives a default value of 0.

RST Statement

In macro definitions containing copy code, the sequence numbers are usually not in ascending order throughout the whole macro definition. The RST (restart) statement is used to specify that a new sequence number series starts in the next macro statement. The RST statement has the following format:



seqno+rel is used to identify the source statement after which the new series starts. seqno represents the last sequence number in the old series. It is a decimal number of 1-n digits in length, where n is the length of the sequence field as specified in the COL statement.
rel is a decimal number of 1-4 digits in length. If omitted, rel receives a default value of 0.

Note: If an ADD, DEL, or REP operation has to be performed on the last macro statement in the series, the first statement in the new series has to be referenced in the RST statement. seqno must still be the last sequence number in the old series.

The following illustration shows the use of the RST statement:

Assume that you have the following macro:

```
       MACRO
       MAC1
         .                    10
         .                    20
         .                    30
         .                    40
       COPY                   50
         .                    60
         .                    70
         .                    80
       MEND
```

and assume that you have the following DSPLY output from the above macro:

```
       Statement          Sequence No.

       MAC1
         .                    10
         .                    20
         .                    30
         .                    40
         .                     1 )
         .                     2 |
         .                     3 } These statements were
         .                     4 } inserted by the COPY
         .                     5 } statement
         .                     6 )
         .                    60
         .                    70
         .                    80
```

If you wanted to delete statement 20 and statement 3, your coding would be:

```
       ) DEL 20
       ) RST 40
       ) DEL 3
```

If you wanted to delete statement 40 and statement 3 (see Note under "RST Statement", above), your coding would be:

```
       ) DEL 40
       ) RST 40+1
       ) DEL 3
```

END Statement

The END statement is used to indicate the end of an update to a macro definition. The format is:

```
   Column 1
          |
          v
          ) END
```

This statement is required for all updating; otherwise an error message is given.

All update control statements must have a right parenthesis in column 1. At least one blank must separate the right parenthesis from the operation field. At least one blank must separate the operation field from the operand field.

The following rules apply to the relationship between operands.

1. Any seqno+rel must specify a source statement after the one addressed by the previous seqno+rel.

2. Any seqno must be greater than or equal to the last seqno in the previous control statement, or the seqno of the first operand in the same control statement.

   An exception to rule 1 is:

   Two consecutive seqno+rel may be equal, (a) as the two operands of a DEL or REP statement, (b) if a verified source statement is referenced in the next control statement which must be an ADD, DEL, REP, or RST statement.

   An exception to rule 2 is:

   The first seqno in the control statement following an RST statement is independent of the seqno in the RST card.

ERRORS DETECTED DURING UPDATE - ACTION TAKEN

If an error is detected during updating, a message is printed on the update survey. The requested update action will not be performed. If possible, updating will continue with the next update control card. Otherwise, a termination message is given and only the remaining update control cards are printed on the survey. De-editing of the macro will always be completed. The job will be cancelled at the end of the ESERV run; that is, remaining jobsteps to edit the macro and catalog the edited macro definition again will not be executed.

Updating will continue with the next update control card for all errors except when:

1. The COL statement has invalid operands.
2. COL statement is not the first update control statement.
3. The macro is completely de-edited without all update control statements being completely processed.
4. An RST statement has an invalid operand.

Examples of De-Editing With and Without Updating a Macro Definition

The following two examples show the two different features of the ESERV program: that of de-editing without updating an edited macro definition, and that of de-editing and updating an edited macro definition.

SAMPLE CODING FOR DE-EDITING WITHOUT UPDATING A MACRO DEFINITION

```
// JOB NOUPDATE          Name of job is NOUPDATE.
// EXEC ESERV            Causes ESERV to de-edit the macro
                         specified in the following PUNCH
                         statement.

 PUNCH E.MAC1,MAC2       Causes the macros MAC1 and MAC2 to
                         be punched out from the macro library
                         (E).
/*
/&
```

You could use the above coding to produce a de-edited source macro for possible future updates.

SAMPLE CODING FOR DE-EDITING AND UPDATING A MACRO DEFINITION

The procedure in the following example produces a de-edited, updated macro definition in source format, and edits and places the updated macro definition in the macro library, using the MAINT program.

```
// JOB UPDATE

// EXEC ESERV                 Causes ESERV to de-edit the macro
                              specified in the following DSPCH state-
                              ment.

 GENEND                       Causes an END and /* statement to be
                              generated.  These are necessary to allow
                              output from ESERV to be used immediately
                              as input to assembler program.

 DSPCH E.MAC1                 Causes the macro definition MAC1 to be
                              punched and printed from the macro
                              library (E).

) COL 77,4                    For explanation see: "Verifying/
) VER 72+1,5                  Updating Statements from Printout
                              of Source Macro Definition".
.PP9
) ADD 72+1

  AIF (&PCH NE 1400).D4
) DEL 102,103
) REP 245
JOYCE CLC 0(4,REGG),BLANKS
) END
/*
// PAUSE                      Check list, move deck to reader.
// OPTION EDECK,NODECK        Causes the assembler to produce an edited
// EXEC ASSEMBLY              deck (EDECK): no object module
                              will be produced (NODECK).
    (deck produced by ESERV
     goes here)
/*
// PAUSE                      Move SYSPCH deck to reader.
// EXEC MAINT                 Causes MAINT to put edited macro
                              definition on macro library.
    (deck produced by
     assembler goes here)
/&
```

Note: If desired, tapes or disk extents may be used for input, output, or intermediate work files. A DOS/VS cataloged procedure could be written for installations using disk extents.

Differences Between De-Edited Macro Definitions and Source Macro Definitions

- Remarks are lost in the statements which are not generated, for example, prototype and declaration statements, AIF, AGO, etc.).
- Identification sequence field is lost in:

    a) ANOP statements
    b) Continuation cards to AIF, SET, and inner macro instruction
    c) Prototype and Declaration statements

- Identification-sequence field is changed for the MACRO statement. Zeros are generated in columns 73-80 by the de-editor.
- All self-defining terms in conditional assembly expressions will be de-edited as decimal; thus, a character self-defining term in a SETA expression might be de-edited into, for example, 16382457 (8 positions). This might give a too long source statement. If this happens, a message will be punched and no END statement generated.
- .X comments are lost.
- COPY statements are lost; instead the copied code is inserted.
- Local declarations and positional parameters in prototype statements will be packed as efficiently as possible; for example,

```
LCLA  &A1
LCLA  &A2
```

  will be de-edited as:

```
LCLA &A1,&A2
```

- If two or more ANOPs occur adjacent to each other, only the one with the shortest symbolic name will be kept, and that name will be used in all references in the model statements.
- If two or more ANOPs have equally short names, the symbol which is first defined will be kept.
- Superfluous ANOPs are lost.
- Superfluous parentheses and periods are lost (AIF, SETx).
- The original column pattern may be changed for statements which are not generated.
- A dummy ANOP will always be generated for each sequence symbol definition.
- If LCLX statements were interspersed with ordinary statements, their comments will not be placed in the original place. Instead, they will be placed following the last declaration.

Note: A de-edited macro will not be changed by being edited and then being de-edited.

Interpreting 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. Part(s) of the listing can be suppressed by using other options.

The six parts of DOS/VS Assembler listing are:

- External Symbol Dictionary (ESD)
- Dummy Section Dictionary
- Source and Object Program
- Relocation Dictionary (RLD)
- Symbol Cross Reference Table
- Diagnostics and Statistics

The function and purpose of each of them as well as the individual details are explained in the following text and illustrations.

External Symbol Dictionary

The external symbol dictionary (ESD) section of the listing describes the contents of the ESD records passed to the linkage editor in the object module produced by the assembler. It describes all the control sections in the module and identifies the external symbol defined in it.

This section helps you find references between modules in a multi-module program. It may be particularly helpful in debugging the execution of large overlay programs constructed from several modules, to check the ESD section of the assembler listings.

The ESD section is described in detail. For a full understanding of the terms and concepts used in the explanations of each head refer to OS/VS and DOS/VS Assembler Language.

```
                                  EXTERNAL SYMBOL DICTIONARY
 (1)      (2)          (3)  (4)    (5)    (6)
SYMBOL    TYPE          ID  ADDR  LENGTH LD-ID

          PC (PRIVATE) 01 000000 000988
```

(1) SYMBOL — The name of the symbol described by the entry. (Only for types ER, LD, SD, and WX).

(2) TYPE — The various type designators are defined as:

- CM — Common control section. A control section defined by a COM statement.
- ER — Strong external reference. The entry describes a symbol that appears in the operand field of an EXTRN statement or was defined as a V-type address constant.
- LD — External name. The entry describes a symbol that appears in the operand field of an ENTRY statement.
- PC — Unnamed control section (private code). The entry describes a control section that has not been assigned any name. Unnamed control sections are generated as a result of an unnamed START or CSECT statement or by the omission of any of these statements at the beginning of the program.
- SD — Named control section. The entry describes a control section identified by a START or CSECT statement with a label in the named field.
- WX — Weak external reference. The entry describes a symbol appearing in the operand field of a WXTRN statement.

(3) ID — The external symbol dictionary identification number (ESDID). The number is a unique two-digit hexadecimal number identifying the entry. The number is used for cross-reference between the relocation dictionary and the external symbol dictionary. (Only for types CM, ER, SD, WX, and PC.)

(4) ADDR — The address of the item. The place in the module where the item described by the entry is defined. (Only for types CM, LD, PC and SD.)

(5) LENGTH — The length, in bytes of the assembled control section. (Only for entries of type CM, PC, and SD.)

(6) LD-ID — The ESDID number assigned to the entry for the control section in which this entry is defined (Only for entries of type LD.)

Figure 6. External Symbol Dictionary (ESD)

Dummy Section Dictionary

The dummy section dictionary lists all the DSECTs in the program.

1. **SYMBOL** — Name of the DSECT
2. **ID** — External symbol dictionary identification number (ESDID) of the DSECT.
3. **LENGTH** — Length of the DSECT (in bytes).

```
                                DUMMY SECTION DICTIONARY

  (1)          (2)    (3)
SYMBOL          ID   LENGTH

CCBADR          FF   000048
CCWTCB          FE   000070
CCWBLOCK        FD   000048
VCCWADR         FC   000009
TICCCW          FB   000008
SABADR          FA   000005
PUBADR          F9   000009
CHNTBL          F8   000002
```

Figure 7. Dummy Section Dictionary

Source and Object Program

The third section of the listing contains the source statements of the module, together with object code produced by the assembler for each of the source statements. The location counter values and the object code listed for each statement will help you locate any errors in a main storage dump. You also have the possibility to check that your macros have been expanded correctly.

The source and machine language statements section is described in detail in the following figure. That figure discusses many language details that you may not be familiar with. For a complete understanding of all items in the figure, refer to the description of the individual instructions and features in OS/VS and DOS/VS Assembler Language.

```
(1)   PROGRAM-1 (3)     (4)          (5)       (6)                                  (7)                          (8)
 LOC   OBJECT CODE      ADDR1 ADDR2  STMT      SOURCE STATEMENT                     DOS/VS ASSEMBLER V 03.2 00.00 72-01-18
                (2)
                                     36        DCDS  F'101',&C,X'F',B'1101'                                     01300000
000000 000000650F                    37+SYMB1  DC    F'101',X'F'                                                00450000
000008                               38+       DS    F'101',X'F'                                                00500000
00000D 000000
000010 000000650F0D                  39+SYMB2  DC    F'101',X'F',B'1101'                                        00650000
000018                               40+       DS    F'101',X'F',B'1101'                                        00700000
00001E 0000
000020 000000650F0D0F                41+SYMB3  DC    F'101',X'F',B'1101',X'F'                                   00450000
000028                               42+       DS    F'101',X'F',B'1101',X'F'                                   00500000
00002F 00
000030 000000650F0D0F0D              43+SYMB4  DC    F'101',X'F',B'1101',X'F',B'1101'                           00650000
000038                               44+       DS    F'101',X'F',B'1101',X'F',B'1101'                           00700000
```

LOC (1) Called the location counter value (address in hex notation) of the associated code. Exceptions are: for COM, CSECT, and DSECT statements, this field contains the beginning address of the control section. For a LTORG statement this field contains the location assigned to the literal pool. This field is blank for: ENTRY, EXTRN, WXTRN, END, ORG, EQU, and USING statements.

OBJECT CODE (2) The machine language code produced from the source statement on the same line. These entries are left-justified. They can be either machine instructions or assembled constants. Machine instructions are printed in full with a blank inserted after every four digits (two bytes). Assembled constants are printed in full only if PRINT DATA has been specified (see PRINT assembler instruction in OS/VS and DOS/VS Assembler Language.)

PROGRAM-1 (3) The title defined in the operand field of the TITLE statement.

ADDR1 ADDR2 (4) The effective address (the result of adding together a base register and a displacement value) for:

ADDR1 the first operand of an SI or SS type machine instruction.
ADDR2 the second operand of an RX or SS type instruction, or third operand of an END, ORG, or EQU statement. For USING: the first operand value.

Both address fields contain six digits: however, if the high order digit is zero, it is not printed.

STMT (5) The source statement number. Used to cross-reference between this section and the diagnostics section. A plus sign after a number indicates that the instruction was generated from a macro instruction.

SOURCE STATEMENT (6) Columns 1-80 of the source statements. All source statements with the exception of listing control statements (EJECT, PRINT, etc.) are printed. The following items apply to this column:

- The listing control statement PRINT is not printed.
- Macro definitions called from a source statement library are not listed.

Figure 8. Source and Object Program (Part 1 of 2)

- Statements generated as the result of a macro instruction, follow the macro instruction in the listing, They are identified by a plus sign (+) to the left of them.
- Assembler or machine instructions that contain variable symbols, except in macro definitions, are listed twice, once as they appear in the input, and once with values substituted for the variable symbols.
- An error indicator ***ERROR*** follows a statement in error. An error message in the diagnostic messages section explains the error.
- MNOTE messages are listed in-line. An MNOTE indicator message appears in the diagnostic messages section of the listing unless the first operand of the MNOTE statement is an asterisk.
- When an error is found in a source macro definition (a definition contained in the same source module), it is treated as any other assembler error: the error indicator is placed after the statement in error, and a diagnostic message appears in the diagnostic section of the listing.
- When a macro is encountered during the expansion of a macro, the error indicator appears after the last statement generated before the error was encountered, and the associated diagnostic message is placed in the diagnostic section.
- Literals that have not been assigned locations by means of LTORG statements are listed after the END statement.

DOS/VS ASM V 03.2 (7) Identification of the assembler variant used for this assembly.

00.00 72-01-18 (8) The time and date when the assembly is run.

Figure 8. Source and Object Program (Part 2 of 2)

Relocation Dictionary

The relocation dictionary (RLD) section of the listing describes the contents of the RLD records passed to the linkage editor in the object module. The entries describe the address constants in the module that are affected by the program relocation. The section helps you find the relocatable constants in your program. This is especially useful if you are writing a self-relocating program. The RLD table can also be used to locate the V-type address constants in the program. You look up the name of the constant in the ESD table and from there you get the ESDID. You can find its address in the RLD table. (The V-type address constants are not in the cross-reference table.)

RELOCATION DICTIONARY

(1) ESDID FOR ADDR CON	(2) ESDID FOR REF SYMBOL	(3) TYPE	(4) LENGTH	(5) ADDRESS
01	+01	A	4	000108
01	+01	A	4	000120
01	+01	A	4	000140
01	+01	A	4	000148
01	+01	A	4	000168

(1) **ESDID FOR ADDR CON**

The external symbol dictionary identification number (ESDID) assigned to the ESD entry for the control section in which the address constant appears.

(2) **ESDID FOR REF SYMBOL**

The external symbol dictionary identification number (ESDID) assigned to the ESD entry for the control section in which the referenced symbol is defined.
+ sign: positive relocation
−sign: negative relocation

(3) **TYPE** The type of the address constant (A, Y, CCW, or V).

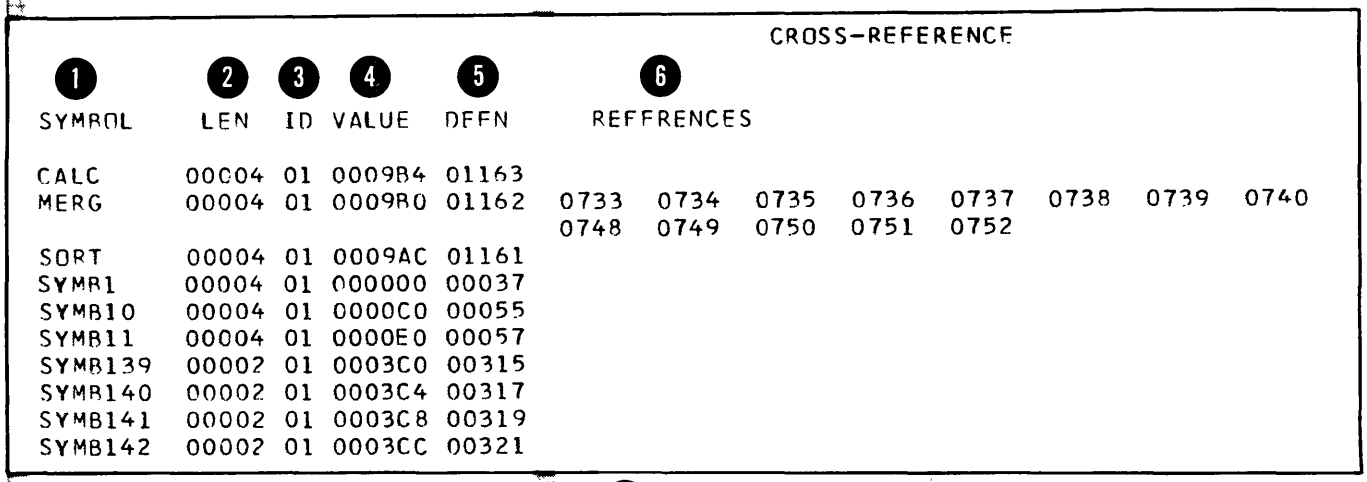
(4) **LENGTH** The length of the address constant.

(5) **ADDRESS** The address where the constant is stored, that is, the location counter value given to the definition of the constant.

Figure 9. The Relocation Dictionary

Cross-Reference Table

The cross reference section of the listing lists the symbols used in the module, indicating where they are defined, and where they are referenced. This is a useful tool in checking the logic of the program; it helps you see if your data references and branches are in order. The cross-reference table is produced only if the option XREF is in effect (see figure 3 which lists and defines the options available).



CROSS-REFERENCE

❶ SYMBOL	❷ LEN	❸ ID	❹ VALUE	❺ DEFN	❻ REFERENCES
CALC	00004	01	0009B4	01163	
MERG	00004	01	0009B0	01162	0733 0734 0735 0736 0737 0738 0739 0740 0748 0749 0750 0751 0752
SORT	00004	01	0009AC	01161	
SYMB1	00004	01	000000	00037	
SYMB10	00004	01	0000C0	00055	
SYMB11	00004	01	0000E0	00057	
SYMB139	00002	01	0003C0	00315	
SYMB140	00002	01	0003C4	00317	
SYMB141	00002	01	0003C8	00319	
SYMB142	00002	01	0003CC	00321	

❶ SYMBOL — The name of the symbol used in the module.

❷ LEN — The length (decimal notation), in bytes, of the field represented by the symbol.

❸ ID — The external symbol dictionary identification number (ESDID) related to the symbol (e.g., the CSECT where it is defined). Blank for absolute equates.

❹ VALUE — Either the address represented by the symbol, or the value to which the symbol is equated.

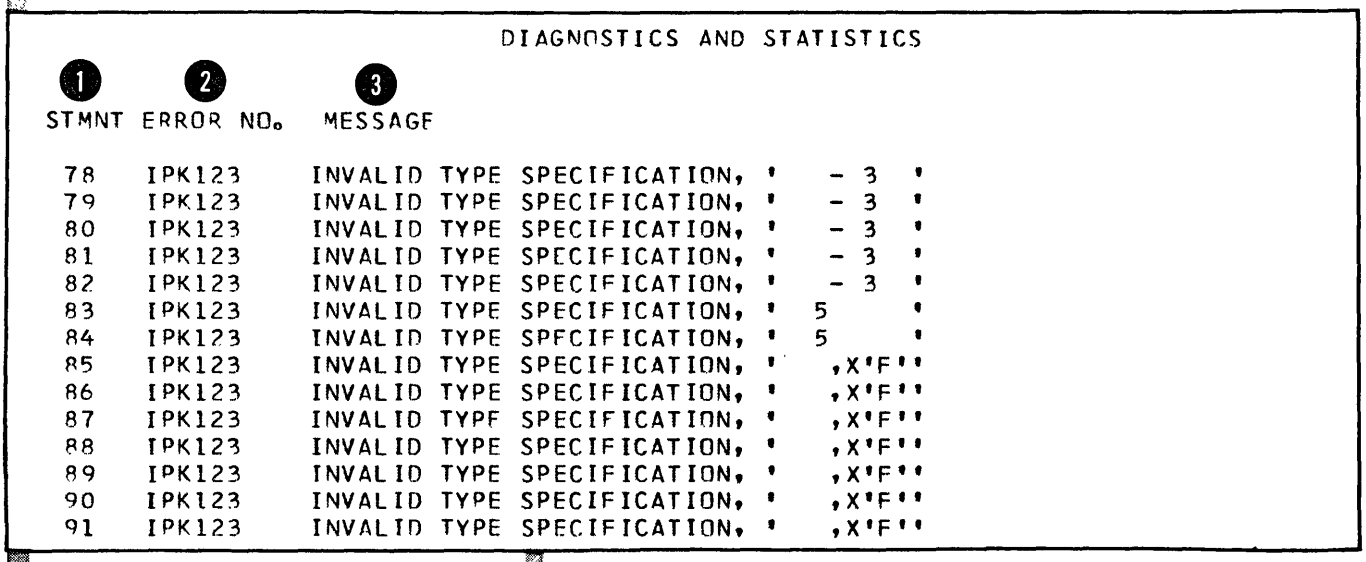
All symbols in the module, except those appearing in the operand field of V-type address constants are included. Thus, symbols that are not listed in the source and machine language statements section because of a PRINT OFF or PRINT NOGEN instruction will appear in the cross-reference section. (For a description of V-type address constant and the PRINT instruction refer to OS/VS and DOS/VS Assembler Language.) The cross-reference section also lists undefined and duplicate symbols for which error indicators and diagnostics have been issued. Following the symbols, all literals are listed.

❺ DEFN — The statement number of the statement in which the symbol is defined.

❻ REFERENCES — The statement numbers of the statements in which the symbol appears in the operand field.

Figure 10. The Cross-Reference Table

The diagnostics and statistics section contains the diagnostic messages as a result of error conditions encountered in the program. All the messages, their contents, and explanations are contained in the section "Diagnostic and Error Messages".



DIAGNOSTICS AND STATISTICS

1 STMNT	2 ERROR NO.	3 MESSAGE
78	IPK123	INVALID TYPE SPECIFICATION, ' - 3 '
79	IPK123	INVALID TYPE SPECIFICATION, ' - 3 '
80	IPK123	INVALID TYPE SPECIFICATION, ' - 3 '
81	IPK123	INVALID TYPE SPECIFICATION, ' - 3 '
82	IPK123	INVALID TYPE SPECIFICATION, ' - 3 '
83	IPK123	INVALID TYPE SPECIFICATION, ' 5 '
84	IPK123	INVALID TYPE SPECIFICATION, ' 5 '
85	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
86	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
87	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
88	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
89	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
90	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''
91	IPK123	INVALID TYPE SPECIFICATION, ' ,X'F''

If an MNOTE (see the description of the MNOTE statement in OS/VS and DOS/VS Assembler Language) message other than MNOTE with an asterisk in the operand (or without a severity code) is issued by a macro, the diagnostic message indicates where the MNOTE statement is found in the source and machine language statements section of the listing.

1. **STMNT** — The statement number of the statement flagged. For certain types of errors the statement number is not given.
2. **ERROR NO.** — The message identifier. It consists of the three characters IPK and three numeric characters, giving a unique number to the message.
3. **MESSAGE** — The text of the message. Many messages include a segment of the statement in error in the message.

Figure 11. Diagnostics and Statistics

Main and Auxiliary Storage Requirements

The DOS/VS Assembler requires a minimum of 20K bytes of main storage. Auxiliary storage requirements may be estimated by the following formula:

```
SYS001:     MAX(60xITXT + 60, SM + 60xLM)
SYS002:     MAX(40xETXT, 60xITXT + 60xSM)
SYS003:     60xOTXT if option NOXREF
            100xOTXT if option XREF
```

MAX = Choose the greater of the two expressions separated by the comma

ITXT = Total number of statements on SYSIPT

OTXT = Total number of statements on SYSLST (with PRINT GEN)

SM = Number of source macro statements

ETXT = (OTXT - number of comments - SM)

LM = Number of statements in library macros used by the program

The de-editor requires 26K bytes of main storage. SYS001 and SYS002 are used.

The edited macro library requires about 20% more storage than the corresponding library in blank-compressed source format.

Performance Considerations

The DOS/VS assembler dynamically allocates storage space for workareas and tables. Thus, within certain limits, the assembler will use as much as possible of the virtual partition. Since retrieval of data from workareas is often random, excessive paging will occur if the virtual partition is much larger than the Page Pool. With large assemblies, use the SIZE parameter in the EXEC job control statement to avoid excessive paging.

You should also see the section "Performance Considerations" in Introduction to DOS/VS.

Configuration Specifications

The configuration required for the assembler in a DOS/VS system is:

- A System/370 machine with the standard System/370 instruction set.
- At least 20K bytes of main storage allocated for the assembler.
- DASD for SYSRES (system library).
- DASD for three workfiles (SYS001, SYS002, SYS003).
- Card reader, tape or DASD for input (SYSIPT).

The following devices are required only if the corresponding assembler option is used:

- Card punch, tape or DASD for object code output and edited macro output (SYSPCH).
- Printer, tape or DASD for listing output (SYSLST).
- DASD for object code link file output (SYSLNK).
- DASD for private library (SYSSLB).

Note: DASD is any direct-access device supported by DOS/VS.

Files Used by the Assembler

SYSRDR contains job control statements for the job control program.

SYSIN is a combination of SYSRDR and SYSIPT. Must be used if SYSRDR and SYSIPT input is on the same disk extent.

SYSIPT contains input for processing program (for example, source code for the assembler). Normally the same device as SYSRDR.

SYSLST receives printed output, for example, the assembler listing.

SYSOUT is a combination of SYSLST and SYSPCH. Must be permanent assignment (cannot be assigned by an // ASSGN statement).

SYSPCH receives punched output from language translators, for example, object decks from the assembler. It also receives edited macro deck produced by the assembler with the EDECK option.

SYSLNK contains input for the linkage editor. Language translators write object modules on SYSLNK, if they are to be produced by the linkage editor in the same job.

SYSLOG is used for communication between the system and the operator. A few assembler diagnostic messages may appear here.

SYS001, SYS002, SYS003 are programmer work files used by processing programs. The assembler uses them for intermediate storage during processing.

SYSRES contains the Disk Operating System. The assembler and other processing programs are in the core image library, object modules in the relocatable library, and macro definitions in the source statement library. These libraries can be replaced by, or concatenated with, private libraries (SYSRLB, SYSCLB, and SYSSLB).

SYSSLB is a private source statement library. It is concatenated with the library on SYSRES. The assembler first searches the private library, if assigned, and then the SYSRES library for a macro not found in the source code.

SYSCLB is a private core image library. Available only if specified when the system was generated. Must be a permanent assignment.

The following figure shows the data flow in connection with assembler execution.

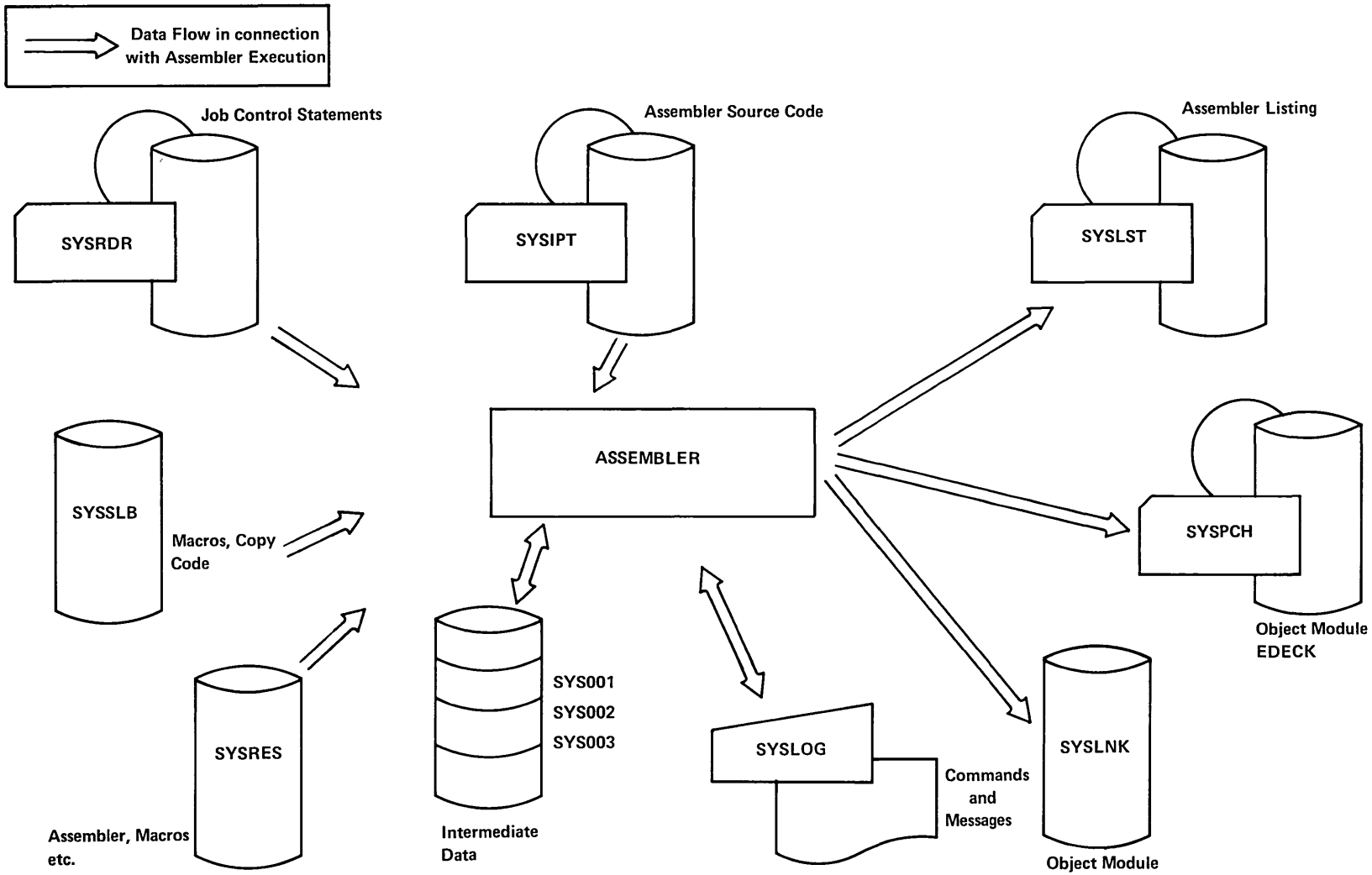


Figure 12. Data Flow in Connection With Assembler Files

Object Deck Output

Listed below are the card groups that make up the object deck produced by the assembler. The groups are listed in the order in which they appear in the output deck. Also included in this section are descriptions of the REP and EDECK card formats.

Note: No object deck will be produced when the option NODECK is used.

Card Group	Remarks
Reproduced Cards	These reproduced cards result from REPRO or PUNCH instructions located before START.
External Symbol Dictionary (ESD)	Contains all the symbol and storage assignments for an object module. For a detailed account of the ESD, see the publication System Control Statements.
Problem Program	Consists of text (TXT) and reproduced cards. The reproduced cards result from REPRO or PUNCH instructions located after START.
Relocation Dictionary (RLD)	Produced if relocatable constants are present.
END Card	Produced as the last card of the output deck.

Object Deck Identification

The 4-character assembly identification label punched into the name entry of the first TITLE card in the source program is punched into columns 73-76 of each record in the object deck (except in reproduced cards). If there is no label, these columns are left blank.

Object Deck Sequencing Numbering

An assembler-generated sequence number is punched into columns 77-80 of each card in the object deck (except in reproduced cards).

ESD Card Format

The format of the ESD card is as follows:

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 (cols. 17-64)
13-14	Blank
15-16	ESDID of first SD, 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 3 bytes - Address 1 byte Blank 3 bytes - length, ESDID or blank
65-72	Blank
73-76	Deck ID (from first TITLE card)
77-80	Card sequence number

TEXT (TXT) Card Format

The format of the TXT card is as follows:

Columns	Contents
1	12-2-9 punch
2-4	TXT
5	Blank
6-8	Relative address of first byte in information field
9-10	Blank
11-12	Byte count--number of bytes in information field (cols 17-72)
13-14	Blank
15-16	ESDID
17-72	56-byte information field
73-76	Deck ID (from first TITLE card)
77-80	Card sequence number

RLD Card Format

The format of the RLD card is as follows:

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 (cols 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
73-76	Deck ID (from first TITLE card)
77-80	Card 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.

END Card Format

The format of the END card is as follows:

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)
17-72	Blank
73-76	Deck ID (from first TITLE card)
77-80	Card sequence number

REP Card Format

If you wish to modify your program after it has been assembled you can do this by means of a REP card, which must be included in the object module which it modifies. The format of the REP card is as follows:

Columns	Contents
1	12-2-9 punch Identifies this as a loader card
2-4	REP - Replace text card
5-6	Blank
7-12	Assembled address of the first byte to be replaced (hexadecimal). Must be right-justified with leading zeros if needed to fill the field.
13	Blank
14-16	External symbol identification number (ESDID) of the control section (SD) containing the text (hexadecimal). Must be right-justified with leading zeros if needed to fill the field.
17-70	From 1 to 11 4-digit hexadecimal fields separated by commas, each receiving two bytes A blank indicates the end of information in this card.
71-72	Blank
73-80	Can be used for program identification

EDECK Card Format

The format of the EDECK card is as follows:

Columns	Contents
1	Column pointer to first record
2-69	Edited text
70-76	Blank
77-80	Sequence number

Diagnostic and Error Messages

This section lists all the diagnostic and error messages that can be issued by the assembler. The messages are listed sequentially.

How to Use This Section

If 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 section. The entry for the message will give you the following items:

- The message number and text of the message.
- Explanation, telling you why the message was issued.
- Assembler action, telling you how the assembler reacted to the error.
- Programmer response to correct the error.
- Operator response to correct the error (only for some messages).

The following explains in more detail the various items of each message entry in this section.

THE MESSAGE ITSELF

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

- Statement number, telling you which statement contained the error.
- The message identification number.
- The text of the message.

STATEMENT NUMBER: For messages IPK230-IPK250 no statement number is given, either because the error cannot be associated with any specific statement, or because the assembler does not have access to the statement number when the error was found.

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

TEXT: The text of the message tells you which error the assembler has encountered. In some messages a number denoted by an 'n' is inserted to identify a position in an operand field where an error occurs. In some messages a character string taken from the source statement is inserted in the message. Such a string is denoted as 'XXXXXXXX' in the message text. The string normally starts at the point where the assembler has discovered an error. However, it does not stop immediately after the operand or character in error has been listed. The character string normally ends with the first blank, or after eight characters have been listed. The explanation for each message is subdivided under the following four headings:

EXPLANATION

This item gives the probable cause of the message. An error message is usually issued at the point where the assembler can no longer make sense of the input, not necessarily at the point where the real error occurred. Thus if you want to code:

```
DC C'HALLO'
```

but instead code

```
DC F'HALLO'
```

the assembler will flag 'HALLO' as an invalid data field rather than F as an invalid type specification.

ASSEMBLER ACTION

This item tells you how the assembler reacts to the error, and what default actions are taken.

PROGRAMMER RESPONSE

This item tells you how to correct the statement in error.

OPERATOR RESPONSE

For messages that are printed on the operator's console, this item tells the operator how to correct certain errors. The operator will not change your source deck; however, he may change the partition size, and assign new devices etc.

Note: Each assembler module will list a maximum of four errors per statement.

IPK001 END STATEMENT IN MACRO OR COPY CODE

Explanation: An END statement is found in a macro definition or in code that is inserted by means of the COPY instruction.

Assembler Action: The statement is treated as comments.

Programmer Response: Remove the END statement from the macro definition or the copy book. Make sure that an END statement is included at the end of your source module.

IPK002 ICTL NOT FIRST STATEMENT

Explanation: The ICTL statement is used in a statement that is not the first statement in the source module.

Assembler Action: The statement is processed as comments

Programmer Response: Remove the ICTL statement, or make it the first statement of the program.

IPK003 STATEMENT INCORRECTLY PLACED, MUST BE IN MACRO DEFINITION

Explanation: A MEND, MEXIT, MNOTE, or internal macro comments (.*) statement appears in open code. These statements are allowed only in macro definitions.

Assembler Action: The statement is processed as comments.

Programmer Response: Remove the statement, or put it in a macro definition.

IPK004 COMMENTS BETWEEN MACRO AND PROTOTYPE STATEMENTS

Explanation: The macro header (MACRO) instruction is followed by a comments statement (.* or *) . The macro header must be immediately followed by a macro prototype statement.

Assembler Action: The comments statement is ignored. It is not generated when the macro is generated.

Programmer Response: Put the comments statement after the prototype statement.

IPK005 STATEMENT INCORRECTLY PLACED

Explanation: One of the following errors has occurred:

- A macro header (MACRO) instruction appears too late in the program. It can only be used to identify the beginning of a macro definition, and the macro definitions must all be placed at the beginning of the source module. The only instructions that can precede them are: ICTL, ISEQ, EJECT, PRINT, TITLE, SPACE, and comments statements.

- A GBLx or LCLx instruction in the macro definition does not follow immediately after the macro prototype statement.
- A GBLx instruction is preceded by an LCLx instruction.
- A GBLx or LCLx instruction in open code does not precede the first control section.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure your MACRO, GBLx, and LCLx, instructions are placed according to the rules given in the explanation.

IPK006 ILLEGAL NAME FIELD

Explanation: The name field is not a sequence symbol or blank, which is required by this instruction.

Assembler Action: The name field is ignored.

Programmer Response: Make sure the name field is either a sequence symbol or blank.

IPK007 SOURCE RECORD OUT OF SEQUENCE

Explanation: The input sequence-checking specified by the ISEQ instruction has determined that this record is out of sequence. The sequence field of this record is not higher than the sequence field of the preceding record.

Assembler Action: The statement is flagged and assembled. The sequence of the rest of the statement is checked relative to the sequence of the statements before this statement.

Programmer Response: Put the record in the proper sequence.

IPK008 UNPAIRED APOSTROPHE

Explanation: An ending apostrophe is missing in this statement, or an illegal attribute reference is found in the statement.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a terminating apostrophe or correct the attribute reference. An opening or ending apostrophe must be single, that is, it must not be immediately followed or preceded by another single apostrophe. Double apostrophes are used to specify the character in a quoted string (between the opening and terminating apostrophes).

IPK009 TOO MANY CONTINUATION LINES

Explanation: This statement occupies more than three records.

Assembler Action: The excessive continuation lines are treated as comments.

Programmer Response: Check for an unintentional continuation indicator in the column after the end column (usually in column 72). Do not use more than two continuation lines for a statement.

IPK010 OP CODE MISSING

Explanation: The first or only record of a statement does not contain any operation code, followed by at least one blank.

Assembler Action: The statement is processed as comments.

Programmer Response: If this record is intended to be a comments statement, supply an asterisk in the begin column. If the record is intended to be an instruction, supply an opcode followed by at least 1 blank in the first record of the statement.

IPK011 INVALID OP CODE

Explanation: The specified operation code does not consist of 1-8 alphameric characters, the first of which is alphabetic.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the operation code is a valid ordinary symbol as described in the explanation.

IPK012 MEND NOT PRECEDED BY MACRO IN THIS COPY BOOK

Explanation: In code inserted by means of the COPY instruction, a MEND instruction is encountered for which there is no corresponding MACRO instruction in this copy book.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure that a macro always starts and ends in the same copy book. If a MACRO statement is found in a copy book, the corresponding MEND statement must also be in that copy book.

IPK013 CONTINUATION LINE MISSING

Explanation: End of file was encountered when the assembler was trying to read an expected continuation line.

Assembler Action: The statement is processed as if no continuation mark had been indicated in the continuation column.

Programmer Response: Add the missing continuation line(s), or remove the erroneous continuation mark, whichever is applicable.

IPK014 SYMBOLIC PARAMETER 'xxxxxxxx' TOO LONG

Explanation: The specified symbolic parameter in a macro prototype statement is too long. It must not consist of more than eight characters. The first eight characters of the invalid symbolic parameter are identified in the message.

Assembler Action: The rest of the macro definition is checked for errors, but the macro is considered undefined.

Programmer Response: Make sure that all symbolic parameters consist of an ampersand followed by 1 to 7 alphameric characters, the first of which is alphabetic.

IPK015 SYMBOLIC PARAMETER 'xxxxxxxx' DOES NOT START WITH AMPERSAND

Explanation: The specified symbolic parameter does not start with an ampersand (&).

Assembler Action: The rest of the macro definition is checked for errors, but the macro is considered undefined.

Programmer Response: Make sure that all symbolic parameters consist of an ampersand followed by 1-7 alphameric characters, the first of which is alphabetic.

IPK016 SECOND CHARACTER OF SYMBOLIC PARAMETER 'xxxxxxxx' NOT A LETTER

Explanation: The second character of the specified symbolic parameter is not alphabetic.

Assembler Action: The rest of the macro definition is checked for errors, but the macro is considered undefined.

Programmer Response: Make sure that all symbolic parameters consist of an ampersand followed by 1-7 alphameric characters, the first of which is alphabetic.

IPK017 SYMBOLIC PARAMETER 'xxxxxxxx' CONTAINS NON-ALPHAMERIC CHARACTER

Explanation: The specified symbolic parameter contains an invalid character. Only alphameric characters (A through Z, 0 through 9, #, $) are allowed in symbolic parameters.

Assembler Action: The rest of the macro definition is checked for errors, but the macro is considered undefined.

Programmer Response: Make sure that all symbolic parameters consist of an ampersand followed by 1-7 alphameric characters, the first of which is alphabetic.

IPK018 INVALID OPCODE IN PROTOTYPE STATEMENT

Explanation: The mnemonic operation code of a prototype statement is (a) not a valid symbol, (b) is the same as the opcode of another macro definition in the source program, (c) is the same as the opcode of a machine instruction or assembler instruction.

Assembler Action: The macro definition will be checked for errors just as if the opcode was correct; but when the macro is called it is treated as undefined.

Programmer Response: Make sure that the prototype opcode consists of 1-8 alphameric characters starting with an alphabetic character, and that the prototype opcode is different from other prototype, machine, and assembler opcodes.

IPK019 KEYWORD OPERAND PRECEDES POSITIONAL OPERAND 'xxxxxxxx'

Explanation: In a macro prototype statement or a macro definition a keyword operand has been placed before the positional operand identified in the message. All positional operands must appear before the keyword operands in the statement. If no operand is identified in the message a comma indicating an omitted positional operand has been found after the first keyword operand.

Assembler Action: If the error is found in a prototype statement, all positional operands after the first keyword operand are considered undefined. The rest of the macro definition is then checked for errors, but the macro is considered undefined. Is the error is found in a macro instruction, the macro is not generated.

Programmer Response: Make sure all positional operands in a macro prototype statement or macro instruction precede all keyword operands.

IPK020 TOO MANY LEVELS OF PARENTHESIS IN OPERAND 'xxxxxxxx'

Explanation: The operand expression identified in the message contains more than five levels of parentheses. The text inserted in the message is limited to eight characters.

Assembler Action: If the error is found in a prototype statement, the rest of the macro definition is checked for errors, but the macro is considered undefined. If the error is found in a macro instruction, the macro is not generated.

Programmer Response: Change the expression to delete one or more levels of parentheses.

IPK021 UNPAIRED PARENTHESIS IN OPERAND 'xxxxxxxx'

Explanation: The keyword parameter default value specified in a macro prototype or a macro instruction operand value contains an unpaired left or right parenthesis not surrounded by apostrophes. Only the first eight characters of the operand value are inserted in the message.

Assembler Action: If the error is found in a prototype statement, the rest of the macro definition is checked for errors, but the macro is considered undefined. If the error is found in a macro instruction, the macro is not generated.

Programmer Response: If you want to specify an unpaired parenthesis, make sure it appears with apostrophes. Otherwise make sure a left parenthesis is always followed by a right parenthesis with which it is paired.

IPK022 INVALID SUBLIST 'xxxxxxxx' IN ALTERNATE STATEMENT FORMAT

Explanation: The termination of a macro prototype or macro instruction sublist written in the alternate statement format for sublists is invalid, either because the closing right parenthesis is missing, or because something other than a comma or a blank follows the closing right parenthesis; only the first eight characters of the sublist are inserted in the message list.

Assembler Action: If the error is found in a prototype statement, the rest of the macro is checked for errors, but the macro is considered undefined. If the error occurs in a macro instruction, the macro is not generated.

Programmer Response: If a sublist is intended, make sure that the sublist is terminated by a right parenthesis followed by a comma or a blank. If a character string is intended, use the normal statement format instead.

IPK023 PARAMETER VALUE 'xxxxxxxx' EXCEEDS 255 CHARACTERS

Explanation: The specified value is too long. The parameter value specified in a macro prototype statement (as a keyword parameter default value) or a macro instruction is limited to 255 characters. The text inserted in the message contains only the first eight characters.

Assembler Action: If the error is found in a macro prototype statement, the rest of the macro definition is checked for errors, but the macro is considered undefined. If the error is found in macro instruction, the macro is not generated.

Programmer Response: Limit the length of the parameter to 255 characters, or separate the value into two or more parameters.

IPK024 UNPAIRED APOSTROPHE

Explanation: An unpaired apostrophe is found in a parameter value specified in a macro prototype statement (as a keyword parameter default value) or a macro instruction. Single apostrophes in parameter values must be specified with double apostrophes appearing inside paired apostrophes, unless they are used to specify attribute references in arithmetic expressions.

Assembler Action: If the error is found in a macro prototype statement, the rest of the macro is checked for errors, but the macro is considered undefined. If the error is found in a macro instruction, the macro is not generated.

Programmer Response: Make sure all apostrophes are paired or double, or belong to attribute references.

IPK025 TOO MANY OPERANDS

Explanation: Too many operands found in a macro prototype statement or a macro instruction or too many sub-operands in a sublist. The maximum number allowed is 200.

Assembler Action: If the error is found in a macro prototype statement, the rest of the macro definition is checked for errors, but the macro is considered undefined. If the error is found in a macro instruction, the macro is not generated. Only the first eight characters of the default value are inserted in the message.

Programmer Response: Reduce the number of operands or include some of the operands in sublists or, if too many sub-operands, split the sublist into two or more.

IPK026 INVALID NAME FIELD 'xxxxxxxx'

Explanation: The name field of a macro prototype statement or a macro instruction is invalid. The name field of a prototype statement must either be blank or contain a variable symbol specifying a name field parameter. The name field of a macro instruction must either be blank, or contain a sequence symbol, or a valid ordinary symbol, or one or more variable symbols that result in a valid ordinary symbol after substitution and concatenation. Only the first eight characters of the default value are inserted in the message.

Assembler Action: If the error is found in a macro prototype statement, the rest of the macro definition is checked for errors, but the macro is considered undefined. If the error is found in a macro instruction, the macro is not generated.

Programmer Response: Supply a valid name field as described in the explanation.

IPK027 NON-BLANK CHARACTER FOUND BEFORE CONTINUE COLUMN

Explanation: On a continuation record, that is, a record following after the first record of a statement occupying several records (lines), one or more characters have been encountered in the begin column or in the column between the begin column (usually column 1) and the continue column (usually column 16). These columns must be blank.

Assembler Action: The characters appearing before the continue column are ignored.

Programmer Response: If the record is intended as a continuation record, make sure the statement is continued in the correct column. If the record is not meant to be continue record, check for an unintentional continuation indicator in the preceding record (usually in column 72).

IPK028 INVALID KEYWORD PARAMETER DEFAULT VALUE 'xxxxxxxx'

Explanation: The default value specified for a keyword parameter in a macro prototype statement is invalid. The value must not contain variable symbols, and any ampersands must be double, that is, each sequence of consecutive ampersands must contain an even number of ampersands. Only the first eight characters of the default value are inserted in the message.

Assembler Action: The rest of the macro definition is checked for errors, but the macro is considered undefined.

Programmer Response: Delete variable symbols from the default value, make ampersands double.

IPK029 INVALID KEYWORD IN MACRO INSTRUCTION, 'xxxxxxxx'

Explanation: A keyword of a macro instruction does not consist of a 1-7 alphameric characters, the first of which is alphabetic, or a macro instruction operand contains an equal sign outside quotes or parentheses.

Assembler Action: The rest of the macro is checked for errors, but the macro is considered undefined.

Programmer Response: Make sure that all keywords consist of a letter followed by 0-6 alphameric characters.

IPK031 NAME FIELD NOT BLANK

Explanation: The name field is not blank, which is required by this instruction.

Assembler Action: The statement is processed as comments.

Programmer Response: Remove the statement from the macro definition. Make sure all your macro definitions end with a MEND instruction.

IPK032 STATEMENT INCORRECTLY PLACED, MUST NOT BE IN MACRO DEFINITION

Explanation: A statement has been found in a macro definition which is not allowed to appear in a macro definition.

Assembler Action: The statement is processed as comments.

Programmer Response: Remove the statement from the macro definition. Make sure all your macro definitions end with a MEND instruction.

IPK033 INVALID ISEQ OR ICTL OPERAND

Explanation: One of the following errors has occurred:

- The operand field of an ISEQ instruction is invalid. It must either be a blank or consist of two decimal self-defining terms that do not fall between the begin and end columns, and the first value must not be greater than the second.
- The operand field of the ICTL statement is invalid. It must consist of one to three decimal self-defining terms, the first of which must be in the range 1-40, the second in the range 41-80, and the third must be in the range 2-40 and greater than the first.

Assembler Action: The statement is processed as comments.

Programmer Response: Correct the operand field according to the rules given in the explanation.

IPK034 INVALID COPY OPERAND

Explanation: The operand of a COPY instruction is not an ordinary symbol.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid ordinary symbol that corresponds to the name of a book in the copy code library. Ordinary symbols consist of 1-8 alphameric characters, the first of which is alphabetic.

IPK035 TOO MANY COPY NEST LEVELS

Explanation: More than three nesting levels of COPY instructions have been coded. Nesting occurs when a COPY instruction is coded in a book that is inserted by means of another COPY instruction.

Assembler Action: The last COPY instruction is processed as comments.

Programmer Response: Reduce the number of nesting levels by including some of the COPY books physically in the source module.

IPK036 COPY BOOK NOT IN LIBRARY

Explanation: The ordinary symbol specified in the operand of this COPY instruction is not the name of a copy book in a source statement library that is assigned to this job.

Assembler Action: The statement is processed as comments.

Programmer Response: Check that the operand is correct, assign the proper source statement library, or catalog the missing copy book.

IPK037 UNEXPECTED END-OF-FILE ON SYSSLB

Explanation: End-of-file was encountered in the source statement library before the end of a book had been reached. Since the end-of-file indicator is normally found only at the end of the COPY code library, the message indicates that the source statement library has been destroyed.

Assembler Action: Processing of the copy book is terminated. If the error occurs inside a source macro definition, a MEND instruction is generated.

Programmer Response: Re-construct the source statement library.

IPK038 MEND STATEMENT MISSING, HAS BEEN ADDED

Explanation: End-of-file occurred on SYSIPT during the processing of a macro definition, or a MEND instruction terminating a macro definition is missing.

Assembler Action: A MEND and an END instruction are inserted.

Programmer Response: Insert the missing MEND instruction or check for an unintentional end-of-file indicator in the source module.

IPK039 END STATEMENT NOT IMMEDIATELY FOLLOWED BY END-OF-FILE

Explanation: The END statement identifying the end of the source module is not immediately followed by an end-of-data indicator statement (/*).

Assembler Action: The records appearing between the END statement and the end-of-data indicator are not processed by the assembler.

Programmer Response: Move the END statement, or make sure your JCL statements are properly placed.

IPK040 END STATEMENT MISSING, HAS BEEN ADDED

Explanation: No END statement was found in the source module.

Assembler Action: An END statement is inserted at the end of the input.

Programmer Response: Supply an END statement at the end of your source module, or make sure that no end-of-data indicator (/*) has been placed inside your source module.

IPK041 MEND STATEMENT MISSING IN COPY BOOK, HAS BEEN ADDED

Explanation: A source macro definition was coded in a copy book, but the macro trailer (MEND) statement to indicate the end of the macro definition was not found in the copy book. The whole macro definition must be coded within one copy book.

Assembler Action: A MEND instruction is inserted at the end of the copy book.

Programmer Response: Make sure that a macro always starts and ends in the same copy book. If a MACRO statement is found in a copy book, the corresponding MEND statement must also be in that copy book.

IPK042 STATEMENT COMPLEXITY EXCEEDED

Explanation: A conditional assembly statement of a macro instruction operand has more than 50 variable symbols. Or the generated edited text string length of one statement which contains subscripted SETC variable symbols exceeds 255 bytes.

Programmer Response: Do not use more than 50 variable symbol references in the same statement, or in a macro instruction operand. In the latter case, rearrange the statement to contain fewer subscripted variable symbols.

Programmer Response: Do not use more than 50 variable symbol references in the same statement or a macro instruction operand.

IPK043 OPERAND MISSING

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

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid operand.

IPK044 INVALID SYNTAX IN SET SYMBOL DECLARATION 'xxxxxxxx'

Explanation: In a SET symbol declaration, a variable symbol is invalid, a comma separating two symbols is missing, or a character other than a blank terminates the field. The text inserted in the message gives eight characters, starting with the character at which the error is found.

Assembler Action: The symbol in which the error is found and the rest of the statements are ignored.

Programmer Response: Make sure the operand field contains only valid variable symbols (possibly dimensioned), separated by commas.

IPK045 INVALID DIMENSION 'xxxxxxxx'

Explanation: The dimension of a SET symbol is incorrectly specified. The dimension specification must follow immediately after the variable symbol and be an unsigned decimal value in the range 1-255 enclosed in parentheses.

Assembler Action: The symbol with the invalid dimension and the rest of the statements are ignored.

Programmer Response: Correct the subscript according to the rules given in the explanation.

IPK046 DIMENSION TOO LARGE, 'xxxxxxxx'

Explanation: A SET symbol declaration specifies a dimension that is greater than 255. The string inserted in the message contains up to eight characters, starting with the dimension value.

Assembler Action: The symbol with the invalid dimension is ignored.

Programmer Response: Break up the SET symbol array into two or more arrays by using additional SET symbols.

IPK047 VARIABLE SYMBOL DUPLICATES SYSTEM VARIABLE SYMBOL OR PREVIOUS DEFINITION, 'xxxxxxxx'

Explanation: The first or only variable symbol in the specified string is either:

- a symbolic parameter, which is identical to a system variable symbol or another symbolic parameter specified in the same macro prototype statement; or
- a SET symbol, which is identical to a system variable symbol, a symbolic parameter specified in the same macro definition, or another SET symbol declared in the same macro definition or open code.

Assembler Action: The flagged definition of the variable symbol is ignored, as well as any further operands in the statement. All references to the symbol are treated as references to the first definition of the variable.

Programmer Response: Make sure that all variable symbols within a macro definition or open code are unique within that scope. Do not define system variable symbols or symbolic parameters or SET symbols. The system variable symbols are:

&SYSECT	&SYSLIST
&SYSNDX	&SYSPARM

IPK048 INVALID SYNTAX IN CONDITIONAL ASSEMBLY STATEMENT 'xxxxxxxx'

Explanation: A conditional assembly statement or a statement with variable symbol substitution contains a syntax error, for example:

- Invalid or misplaced characters in an expression.
- The statement is terminated before its logical end. This could be caused by an unintentional blank inside an expression.
- The sequence symbol in an AGO or AIF operand does not consist of a period, followed by a letter and 1-6 letters or digits; or both. The string in the message contains up to eight characters starting where an error is found.

Assembler Action: The statement is processed as comments.

Programmer Response: The first character of the string in the message tells you where the syntax error was found. Correct the error.

IPK049 'xxxxxxxx' IS AN INVALID VARIABLE SYMBOL

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

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid variable symbol.

IPK050 INVALID ATTRIBUTE REFERENCE 'xxxxxxxx'

Explanation: The attribute reference is illegal for the type of attribute in this context; for example:

- A reference inside a macro definition refers to an ordinary symbol
- An attribute reference refers to a SET symbol
- A K or N attribute reference refers to an ordinary symbol.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the attribute reference is correct, that this type or attribute can refer to this type of symbol, that the reference is properly placed, etc.

IPK051 INCORRECT VARIABLE SYMBOL IN NAME FIELD

Explanation:

- This symbol is declared to be of a type different from the type specified by the operation code in this statement; or
- a system variable symbol or symbolic parameter appears in the name field of the SETx instruction.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the declaration is correct, or change the operation code of this statement. Do not use system variable symbols in the name field of SETx instructions.

IPK052 NAME FIELD MISSING

Explanation: This statement requires a name field, but none is found.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the proper symbol in the name field.

IPK053 NAME FIELD NOT A SEQUENCE SYMBOL

Explanation: This statement requires a sequence symbol in the name field, but no valid sequence symbol is found.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid sequence symbol.

IPK054 INVALID NAME FIELD, MUST NOT CONTAIN SEQUENCE SYMBOL OR BLANK

Explanation: The name field of this statement does not contain an ordinary symbol or one or more variable symbols that result in a valid ordinary symbol.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid ordinary symbol, or make sure that the result of variable symbol substitution and concatenation is a valid ordinary symbol.

IPK055 UNPAIRED LEFT PARENTHESIS

Explanation: A left parenthesis in this statement does not have a corresponding right parenthesis.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the missing right parenthesis, or delete the superfluous left parenthesis.

IPK056 TOO MANY LEVELS OF PARENTHESES

Explanation: This expression has more than five levels of parentheses.

Assembler Action: The statement is processed as comments.

Programmer Response: Reduce the number of levels of parentheses. Use additional SETA instructions, if necessary.

IPK057 COUNT OR NUMBER ATTRIBUTE IN OPEN CODE

Explanation: A count (K') or number (N') attribute has been encountered in open code. These attributes can only appear in macro definitions.

Assembler Action: The statement is processed as comments.

Programmer Response: Do not use the count or number attribute in open code.

IPK058 INVALID SUBSTRING NOTATION 'xxxxxxxx'

Explanation: The comma, or ending right parenthesis in a substring notation is missing. The string in the message consists of up to eight characters, starting where the error is found.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the substring notation consists of two arithmetic expressions, separated by commas and enclosed in parentheses.

IPK059 ILLEGAL USE OF SYSTEM VARIABLE SYMBOL

Explanation: The specified system variable symbol is illegal in this context.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure that &SYSLIST, and &SYSNDX are not used in open code.

IPK060 SINGLE TERM IN LOGICAL EXPRESSION NOT SETB

Explanation: A single term in this logical expression is invalid. A logical term must be either an arithmetic relation, a character relation, or a SETB variable. Logical terms are combined into logical expressions by logical operators (AND, OR, and NOT).

Assembler Action: The statement is processed as comments.

Programmer Response: Check the logical expression for omitted relational terms (EQ, LT, etc.) or mispunched characters or terms.

IPK061 INCOMPLETE LOGICAL EXPRESSION 'xxxxxxxx'

Explanation: An expression in this statement ended prematurely because of one of the following errors:

- Unpaired parenthesis; or
- Illegal character; or
- Illegal operator; or
- Operator not followed by a term

Assembler Action: The statement is treated as comments.

Programmer Response: Correct the logical expression.

IPK062 INVALID SELF-DEFINING TERM, 'xxxxxxxx'

Explanation: A self-defining term is incorrectly specified. It must be:

- 1-8 decimal digits whose value is in the range 0-16,777,215; or
- 1-24 binary digits, enclosed by apostrophes and preceded by the character B; or
- 1-6 hexadecimal digits enclosed by apostrophes and preceded by the character X; or
- 1-3 characters, enclosed by apostrophes and preceded by the character C. The string in the message is up to eight characters starting with the invalid self-defining term.

Assembler Action: The statement is processed as comments.

Programmer Response: Correct the term according to the rules given in the explanation.

IPK063 VALUE OF SELF-DEFINING TERM TOO LARGE, 'xxxxxxxx'

Explanation: The value of a decimal self-defining term in this statement is not in the range 0-16,777,215.

Assembler Action: The statement is processed as comments.

Programmer Response: Specify a value in the range specified in the explanation.

IPK064 OPEN CODE ATTRIBUTE REFERENCE TO 'xxxxxxxx', WHICH IS NOT A VALID ORDINARY SYMBOL

Explanation: This attribute reference does not specify a valid ordinary symbol. Any attribute reference used outside macro definitions must specify an ordinary symbol.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid ordinary symbol that is defined in the program.

IPK065 SET SYMBOL USE INCONSISTENT WITH ITS DECLARATION,'xxxxxxxx'

Explanation: Either the declaration specifies this symbol as dimensioned, but in this statement the symbol is used as undimensioned, or the declaration specifies this symbol as undimensioned, but in this statement the symbol is used as dimensioned. The string in the message consists of up to eight characters starting with the symbol in error.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure that your use of SET symbols is consistent with its declaration.

IPK066 PREVIOUSLY DEFINED SEQUENCE SYMBOL

Explanation: The sequence symbol specified in the name field of this statement has already been defined within the macro definition or open code.

Assembler Action: The name field is ignored.

Programmer Response: Supply a sequence symbol that is unique within this macro definition or open code.

IPK067 UNPAIRED RIGHT PARENTHESIS

Explanation: An expression in this statement contains a right parenthesis that is not matched by a preceding left parenthesis.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the missing left parenthesis, or delete the right parenthesis.

IPK068 VARIABLE SYMBOL UNDEFINED, 'xxxxxxxx'

Explanation: The first or only variable symbol in the string inserted in the message has not been declared as a global or local SET symbol within this macro definition or open code, has not been defined as a symbolic parameter within this macro definition, and is not a valid system variable symbol.

Assembler Action: The statement is processed as comments.

Programmer Response: Define the symbol as a symbolic parameter or a SET symbol. Remember that any global variable symbols used in macro definitions must be declared within the definition.

IPK069 SOURCE MACRO PREVIOUSLY DEFINED

Explanation: The operation code specified in the prototype statement is identical to the operation code of another source macro defined earlier in the program.

Assembler Action: The flagged macro definition has been checked for errors. It cannot be generated.

Programmer Response: Supply a unique operation code for this definition.

IPK070 UNDEFINED SEQUENCE SYMBOL

Explanation: The sequence symbol used in this instruction is not defined within this macro definition or open code.

Assembler Action: No conditional assembly branch is taken.

Programmer Response: Define the symbol in the name field within the macro definition or open code (depending on where it is used), or use a sequence symbol that is already defined.

IPK071 ILLEGAL LENGTH ATTRIBUTE REFERENCE

Explanation:

- The symbol specified in a length attribute reference (L') is not the name of a valid machine instruction, control section definition, CCW instruction, DS instruction, or DC instruction.
- The symbol specified in a length attribute reference is the name of a DC or DS instruction containing variable symbols in the modifier field.

Assembler Action: The length attribute reference is set to one.

Programmer Response: Make sure the length attribute references a symbol for which length attribute references are valid.

IPK072 ILLEGAL SCALE ATTRIBUTE REFERENCE

Explanation: The symbol referenced by a scale attribute reference (S') is not found in the name field of a valid fixed-point, floating-point, or decimal DC or DS instruction.

Assembler Action: The scale attribute reference is set to zero.

Programmer Response: Make sure the scale attribute references a symbol for which scale attribute references are valid.

IPK073 ILLEGAL INTEGER ATTRIBUTE REFERENCE

Explanation: The symbol referenced by an integer attribute reference (I') is not found in the name field of a valid fixed-point, floating-point, or decimal DC or DS instruction.

Assembler Action: The integer attribute reference is set to zero.

Programmer Response: Make sure the integer attribute references a symbol for which attribute references are valid.

IPK074 OVERFLOW DURING ADDITION IN ARITHMETIC EXPRESSION

Explanation: During the evaluation of an arithmetic expression the addition of two terms produces a result that falls outside the range of -2^{31} through $2^{31}-1$.

Assembler Action: The result of the addition is set to zero.

Programmer Response: Make sure all the values in this expression are valid. Try to avoid overflow by adjusting the sequence in which the terms are placed in the expression. If necessary, separate the expression into two or more expressions (using SETA instructions), so that each of them is evaluated individually before they are combined.

IPK075 OVERFLOW DURING SUBTRACTION IN ARITHMETIC EXPRESSION

Explanation: During the evaluation of an arithmetic expression, the subtraction of two terms produces a result that falls outside the range of -2^{31} through $2^{31}-1$.

Assembler Action: The result of the subtraction is set to zero.

Programmer Response: Make sure that all values in the expression are valid. Try to avoid overflow by adjusting the sequence in which the terms are placed in the expression. If necessary, separate the expression into two or more expressions (using SETA instructions), so that each of them is evaluated separately before they are combined.

IPK076 OVERFLOW DURING MULTIPLICATION IN ARITHMETIC EXPRESSION

Explanation: During the evaluation of an arithmetic expression the multiplication of two terms produces a result that falls outside the range of -2^{31} through $2^{31}-1$.

Assembler Action: The result of the multiplication is set to zero.

Programmer Response: Make sure all the values in the expression are valid. Try to avoid overflow by adjusting the sequence in which the terms are placed in the expression. If necessary, separate the expression into two or more expressions (using SETA instructions), so that each of them is evaluated separately before they are combined.

IPK077 CHARACTER STRING USED IN ARITHMETIC EXPRESSION TOO LONG

Explanation: The character string used as an arithmetic term is longer than eight characters.

Assembler Action: The value of the SETC variable is replaced by zero in the arithmetic expression.

Programmer REsponse: Make sure that any variable symbol used in arithmetic expressions have a value of 1-8 characters.

IPK078 CHARACTER STRING USED IN ARITHMETIC EXPRESSION CONTAINS NON-DECIMAL CHARACTER

Explanation: A non-decimal character is found in the value of a parameter or SETC symbol used in arithmetic term.

Assembler Action: The value of the parameter or SETC variable is replaced by zero in the arithmetic expression.

Programmer Response: Make sure that any parameter or SETC symbols used in arithmetic expressions have a value of 1-8 decimal characters.

IPK079 NULL CHARACTER STRING USED IN ARITHMETIC EXPRESSION

Explanation: The value of a SETC symbol used as an arithmetic term is a null string.

Assembler Action: The value of the SETC symbols used in arithmetic expressions have a value of 1-8 decimal characters.

Programmer Response: Make sure that any SETC symbols used in arithmetic expressions have a value of 1 - 8 decimal characters.

IPK080 PARAMETER SUBSCRIPT OUT OF RANGE

Explanation: A symbolic parameter subscript value is outside the range 1-200.

Assembler Action: The reference is treated as a reference to an omitted operand; that is, the value of a null string is assigned to it.

Programmer Response: Supply a subscript value in the range 1-200.

IPK081 LENGTH OF CONCATENATED STRING EXCEEDS 255 CHARACTERS

Explanation: During the concatenation of strings, an intermediate string exceeding 255 characters is generated.

Assembler Action: The first 255 characters are used as the intermediate result.

Programmer Response: Make sure that the total length of two strings concatenated with each other does not exceed 255 characters. If needed, change the sequence of string evaluation, by performing substring operation before concatenation.

IPK082 SUBSCRIPT EXCEEDS DECLARED DIMENSION

Explanation: An arithmetic expression used to specify the subscript of a SET symbol has a value that exceeds the value specified in the declaration of the symbol.

Assembler Action: If the error is found in a conditional assembly statement, the statement is processed as comments. The error is found during substitution in one of the fields (name, operation, or operand) of a model statement. The whole field is replaced by a null value. If the error is found during substitution in a macro instruction operand, the operand is set to null value, but any other operands in the operand field are generated.

Programmer Response: Make sure the arithmetic expression has a value in the range of 1 through the declared dimension of the SET symbol.

IPK083 SUBSCRIPT ZERO OR NEGATIVE

Explanation: An arithmetic expression used to specify the subscript of a SET symbol has a value that is zero or negative.

Assembler Action: If the error is found in a conditional assembly statement, the statement is processed as comments. The error is found during substitution in one of the fields (name, operation, and operand) of a model statement, the whole field is replaced by a null value. If the error is found during substitution in a macro instruction operand, the operand is set to a null value, but any other operands in the operand field are generated.

Programmer Response: Make sure that the arithmetic expression has a value in the range of 1 through the declared dimension of the SET symbol.

IPK084 ACTR LIMIT EXCEEDED

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

Assembler Action: If a macro is being generated, its generation is terminated. If open code is being processed, all remaining statements are processed as comments.

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

IPK085 FIRST SUBSTRING EXPRESSION ZERO OR NEGATIVE

Explanation: The arithmetic expression used to specify the starting character for a substring operation has a zero or negative value.

Assembler Action: The result of the substring operation is a null string.

Programmer Response: Make sure the arithmetic expression used to specify the starting character of the substring has a positive value not exceeding the length of the character string.

IPK086 FIRST SUBSTRING EXPRESSION EXCEEDS STRING LENGTH

Explanation: The arithmetic expression used to specify the starting character for a substring operation has a value greater than the length of the string.

Assembler Action: The result of the substring operation is a null string.

Programmer Response: Make sure the arithmetic expression used to specify the starting character of the substring has a positive value not exceeding the length of the character string.

IPK087 SECOND SUBSTRING EXPRESSION NEGATIVE

Explanation: The arithmetic expression used to specify the length of a substring has a negative value.

Assembler Action: The result of the substring operation is a null string.

Programmer Response: Make sure the arithmetic expression used to specify length has a zero positive value, and that the specified length does not extend beyond the end of the character string.

IPK089 SETC OPERAND TOO LONG

Explanation: The character string value in a SETC operand contains more than eight characters.

Assembler Action: Only the first eight characters are assigned to the SETC symbol.

Programmer Response: Make sure that the value of the SETC expression contains no more than eight characters.

IPK090 SYSLIST SUBSCRIPT NEGATIVE

Explanation: The arithmetic expression used to specify a &SYSLIST subscript has a negative value.

Assembler Action: The reference is treated as a reference to an omitted operand; that is, the value of a null string is assigned to it.

Programmer Response: Supply a non-negative value in the &SYSLIST subscript.

IPK091 PARAMETER VALUE INVALID FOR LENGTH ATTRIBUTE REFERENCE

Explanation:

- A length attribute reference specifies a symbolic parameter whose value is not the name of a machine instruction, control section definition, CCW instruction, DS instruction, or DC instruction; or
- A length attribute reference specifies a symbolic parameter whose value is the name of a DS or DC instruction containing variable symbols in the modifier field.

Assembler Action: The length attribute reference is set to one.

Programmer Response: Make sure that the referenced macro instruction operand is a symbol for which length attribute references are valid, or delete the length attribute reference from the macro definition.

IPK092 PARAMETER VALUE INVALID FOR SCALE ATTRIBUTE REFERENCE

Explanation: A scale attribute reference specifies a symbolic parameter whose value is not the name of a fixed-point, floating-point, or decimal DC or DS instruction.

Assembler Action: The scale attribute reference is set to zero.

Programmer Response: Make sure that the referenced macro instruction operand is a symbol for which scale attribute references are valid, or delete the scale attribute reference from the macro definition.

IPK093 PARAMETER VALUE INVALID FOR INTEGER ATTRIBUTE REFERENCE

Explanation: An integer attribute reference specifies a symbolic parameter whose value is not the name of a fixed-point, floating-point, or decimal DC or DS instruction.

Assembler Action: The integer attribute reference is set to zero.

Programmer Response: Make sure that the referenced macro instruction operand is a symbol for which integer attribute references are valid, or delete the integer attribute reference from the macro definition.

IPK094 PARAMETER VALUE INVALID IN ARITHMETIC EXPRESSION

Explanation: The value of a symbolic parameter used in an arithmetic expression is not a valid self-defining term, or 1-8 decimal characters created by variable symbol substitution in the macro instruction.

Assembler Action: The symbolic parameter is replaced by the value of zero in the arithmetic expression.

Programmer Response: Make sure the referenced macro instruction operand is a valid self-defining term or 1-8 decimal characters created by substitution, or remove the symbolic parameter from the arithmetic expression in the macro definition.

IPK095 TOO MANY ERRORS IN THIS STATEMENT

Explanation: During the processing of a conditional assembly statement or a statement with variable symbol substitution, more than five errors are detected. Messages are issued only for the first five errors.

Assembler Action: If more errors are found, they will not be flagged.

Programmer Response: Correct the indicated errors, and check for further errors beyond the point indicated by the fifth error message. Any additional errors will be detected in the next assembly.

IPK096 GENERATED STATEMENT TOO LONG

Explanation: The total length of the statement exceeds 248 characters after generation.

Assembler Action: The statement is processed as comments. If any part of the remarks field falls outside the space allowed for this statement, no part of the remarks field is listed.

Programmer Response: Make sure that the total length of a statement after generation does not exceed 248 characters.

IPK097 UNDEFINED OP CODE, OR MACRO NOT FOUND

Explanation: The operation code of this statement does not correspond to any of the following:

- A machine instruction operation code
- An assembler instruction operation code
- The operation code of a valid library macro or a valid source macro

Assembler Action: The statement is processed as comments.

Programmer Response: Change the operation code to a valid machine, assembler, or macro operation code, or correct the corresponding macro definition. If the error occurred for a library macro, make sure the correct source statement library is assigned.

IPK098 KEYWORD PARAMETER 'xxxxxxxx' DUPLICATED OR NOT DEFINED

Explanation: A keyword parameter appears more than once in a macro instruction, or a keyword parameter appears in a macro instruction in whose definition it is not defined as a keyword parameter. The message will also be given if an equal sign not enclosed in a quoted string or within parentheses appears in a positional parameter.

Assembler Action: If the keyword parameter is duplicated, the first parameter value is accepted. If the parameter is undefined, it is ignored.

Programmer Response: Delete the keyword from the macro instruction, define the parameter in the macro definition, or enclose the equal sign within apostrophes or parentheses.

IPK099 TOO MANY MACROS CALLED

Explanation: The dictionary space available to the assembler is not large enough to generate all the different macros that are called in the source module.

Assembler Action: The whole assembly is processed as comments.

Programmer Response: Increase the size of the partition allocated to the assembly, or separate the module into smaller modules to be assembled separately.

IPK100 TOO MANY MACROS CALLED OR TOO MANY VARIABLE SYMBOLS

Explanation: The dictionary space available to the assembler is not large enough to contain all the different macros, local variable symbols in open code, and global variable symbols that are used in this source module.

Assembler Action: The whole assembly is processed as comments.

Programmer Response: Increase the size of the partition allocated to assembly, or separate the module into smaller source modules to be assembled separately, making sure that references to a variable symbol all remain in the same module.

IPK101 DICTIONARY SPACE FOR VARIABLE SYMBOLS EXHAUSTED 'xxxxxxxx'

Explanation: The dictionary space available to the assembler is not enough to contain all the different macros and global variable symbols of the entire assembly, plus all the local SET symbols and symbolic parameters used in the macro being generated.

Assembler Action: The generation of the macro is terminated.

Programmer Response: Increase the size of the partition allocated to assembly, reduce the number of local variable symbols, or separate the module into smaller source modules to be assembled separately.

IPK102 SEQUENCE SYMBOL UNDEFINED

Explanation: A sequence symbol used in the operand of an AGO or AIF instruction is not defined in the name field of an instruction in the same macro definition or open code.

Assembler Action: The next sequential instruction is processed.

Programmer Response: Define the sequence symbol in the macro, or use a sequence symbol that is already defined.

IPK103 REFERENCE TO GLOBAL VARIABLE SYMBOL WITH INCONSISTENT DECLARATION OF TYPE OR DIMENSION

Explanation: A global variable symbol is used whose declaration within this macro or open code is inconsistent with a previous declaration of the same global symbol. The inconsistency occurred either in the type or the dimension specification.

Assembler Action: The statement is processed as comments. If any part of the remarks field falls outside the space allowed for this statement, no part of the remarks field is listed.

Programmer Response: Make sure that all declarations of a global variable symbol are identical; for example, a global symbol cannot be declared as a SETB symbol in one macro and a SETA symbol in another; and it cannot be declared as dimensioned in one macro and undimensioned in another, or as having a dimension of 50 in one macro and 85 in another.

IPK104 OP CODE 'xxxxxxxx' GENERATED

Explanation: One of the following assembler operation codes has been created by substitution: COPY, END, ICTL, ISEQ, PRINT, REPRO, MACRO, MEND, MEXIT, ANOP, SETA, SETB, SETC, AIF, AIFB, AGO, AGOB, GBLA, GBLB, GBLC, LCLA, LCLB, and LCLC. These operation codes are not allowed to be generated.

Assembler Action: The generated statement is processed as comments.

Programmer Response: Make sure that none of the operation codes listed in the explanation is created by substitution.

IPK105 GENERATED OP CODE 'xxxxxxxx' UNDEFINED OR INVALID

Explanation: The operation code created by substitution is not a valid machine or assembler instruction operation code (macro instructions are not allowed to be generated).

Assembler Action: The generated statement is processed as comments.

Programmer Response: Make sure that the generation in the operation field results in a valid operation code.

IPK106 GENERATED OP CODE IS BLANK

Explanation: The operation code created by substitution contains no characters or only blank characters.

Assembler Action: The generated statement is processed as comments.

Programmer Response: Make sure that substitution results in a valid machine or assembler operation code.

IPK107 MACRO 'xxxxxxxx' NOT EXPANDABLE DUE TO ERROR IN DEFINITION

Explanation:

Source Macro: The prototype statement of the macro definition contains errors.

Library Macro: The library macro contains an error, defined by another error message (of the type that has no statement number).

Assembler Action: The statement is processed as comments.

Programmer Response:

Source Macro: Correct the prototype statement.

Library Macro: Edit and catalog the macro again.

IPK108 MACRO 'xxxxxxxx' NOT EXPANDABLE DUE TO ERROR IN MACRO INSTRUCTION

Explanation: An error has been found in a substitution expression in a macro instruction operand.

Assembler Action: The statement is processed as comments.

Programmer Response: Check the other error messages on this macro instruction and correct the error(s).

IPK109 INVALID OR ILLEGAL NAME FIELD

Explanation: Either the name field is not blank and does not contain a valid ordinary symbol, that is, one to eight alphameric characters the first of which is alphabetic, or there is an ordinary symbol in a name field that should only contain a sequence symbol or blank (CNOP, ORG, END, USING, DROP).

Assembler Action: The name field is ignored.

Programmer Response: Supply a valid ordinary symbol, delete the characters from the name field, or, if you want to write a comments statement, supply an asterisk in the begin column, or remove entries in name field.

IPK110 NAME FIELD TOO LONG

Explanation: The symbol in the name field exceeds eight characters.

Assembler Action: The name field is ignored.

Programmer Response: Make sure the name field is blank or contains a valid ordinary symbol, that is, 1-8 alphameric characters the first of which is alphabetic.

IPK111 GENERATED SEQUENCE SYMBOL

Explanation: A period is found in the name field of a generated statement.

Assembler Action: The name field is ignored.

Programmer Response: Make sure the value generated in the name field is either a valid ordinary symbol or blank.

IPK112 INVALID CHARACTER IN CONSTANT, 'xxxxxxxx'

Explanation: Subfield 4 (constant) of a DC or DS instruction, or literal contains characters that are invalid for the type of constant specified in subfield 2.

Assembler Action: The DS or DC instruction is processed as comments, or the literal is ignored.

Programmer Response: Make sure the characters used to specify the value of the constant are valid for this type of constant. Either change the type or value specification.

IPK113 SYMBOL 'xxxxxxxx' TOO LONG

Explanation: The specified symbol contains more than eight characters. Only the first eight characters of the symbol are identified in the message.

Assembler Action: The operand is checked for further errors, but this operand and any further operands are ignored when the object code is generated.

Programmer Response: Supply a valid symbol.

IPK114 RIGHT PARENTHESIS MISSING 'xxxxxxxx'

Explanation: In the operand indicated in the message, a left parenthesis not matched by a right parenthesis has been found, or the parentheses have been incorrectly placed in the operand; for example, MVC (1,2),3(4) will cause this message to be issued. The first character in the string inserted in the message indicates where the right parenthesis was expected. If only a blank appears in the string the right parenthesis was expected at the end of the operand.

Assembler Action: The operand and the rest of the operand field is ignored.

Programmer Response: Make sure the parentheses are paired and correctly placed.

IPK115 UNPAIRED APOSTROPHE

Explanation: No terminating apostrophe has been found to end the quoted string in this statement.

Assembler Action: The statement is ignored.

Programmer Response: Supply the missing apostrophe.

IPK116 INVALID SELF-DEFINING TERM, 'xxxxxxxx'

Explanation: The first or only self-defining term specified in the text inserted in the message contains invalid characters or a null value (for example,B'102',X'').

Assembler Action: The operand in which the term appears is ignored.

Programmer Response: Supply a valid decimal, binary, hexadecimal, or character self-defining term.

IPK117 VALUE OF SELF-DEFINING TERM 'xxxxxxxx' TOO LARGE

Explanation: The value of the specified self-defining term is either too long or too large. Valid rules are:

- 1-8 decimal digits whose value is in the range 0-16,777,215 or
- 1-24 binary digits, enclosed by apostrophes and preceded by the character B; or
- 1-6 hexadecimal digits enclosed by apostrophes and preceded by the character X; or
- 1-3 characters, enclosed by apostrophes and preceded by the character C.

Assembler Action: The operand in which the term appears is ignored.

Programmer Response: Correct the term according to the rules given in the explanation.

IPK118 ILLEGAL ATTRIBUTE REFERENCE, 'xxxxxxxx'

Explanation: The length attribute reference does not specify a valid ordinary symbol or location counter reference (*). The first character of the inserted string indicates the point where the illegal attribute reference was found.

Assembler Action: The operand is ignored.

Programmer Response: Supply a valid ordinary symbol or location counter reference.

IPK119 TOO MANY OPERATORS, 'xxxxxxxx'

Explanation: More than 15 operators have been found in the operand specified by the message. The first character of the inserted string indicates the point where too many operators have been encountered.

Assembler Action: The operand is ignored.

Programmer Response: Limit the number of operators. If necessary, use EQU instructions to break up the expression into smaller expressions.

IPK120 TOO MANY LEVELS OF PARENTHESES, 'xxxxxxxx'

Explanation: More than five levels of parentheses are used in an expression in the operand specified by the message. The first character in the string indicates the point where too many levels of parentheses have been encountered.

Assembler Action: The operand is ignored.

Programmer Response: Limit the number of levels of parentheses. If necessary, use EQU instructions to break up the expression into smaller expressions, each of which is evaluated separately.

IPK121 ILLEGAL CHARACTER IN EXPRESSION, 'xxxxxxxx'

Explanation: In an expression an invalid character has been found in or instead of a term. The first character of the text inserted in the message identifies the invalid character.

Assembler Action: The operand in error and the rest of the statement are ignored.

Programmer Response: Supply a valid term.

IPK122 INVALID DELIMITER, 'xxxxxxxx'

Explanation: An operand or sub-operand is not delimited by a comma, a left or right parenthesis, or a blank. The first character of the text inserted in the message identifies the invalid delimiter.

Assembler Action: The operand in error and the rest of the statement are ignored.

Programmer Response: Make sure all the delimiters in the statement are correct.

IPK123 INVALID TYPE SPECIFICATION, 'xxxxxxxx'

Explanation: The type specified in subfield 2 of a DC or DS instruction or a literal is invalid or missing. The text inserted in the message consists of up to eight characters, starting at the point where a valid type specification is expected.

Assembler Action: For DC or DS instructions the statement is processed as comments. For constants in a literal, zeros are generated in the output module.

Programmer Response: Supply a valid type specification (A, B, C, D, E, F, H, L, P, S, V, X, Y, or Z).

IPK124 INVALID SYMBOL IN ENTRY, EXTRN, or WXTRN STATEMENT

Explanation: An ENTRY, EXTRN, or WXTRN instruction contains an invalid symbol. The operand field of these statements must consist of one or more ordinary symbols, separated by commas. Any ordinary symbols defined by the ENTRY instruction must also appear in the name field of an instruction in this source module.

Assembler Action: The operand in error and the rest of the statement are ignored.

Programmer Response: Make sure that the operand field follows the rules given in the explanation.

IPK125 DATA ITEM TOO LONG, 'xxxxxxxx'

Explanation: The constant value specified in subfield 4 of a P- or Z-type constant contains too many characters. A P-type constant is limited to 31 decimal characters, and a Z-type constant is limited to 16 decimal characters. The first character in the string indicates the point where the illegal constant value was found.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete the exponent modifier or change to type specification.

IPK126 EXPONENT MODIFIER USED ILLEGALLY, 'xxxxxxxx'

Explanation: An exponent modifier is specified for a DC or DS instruction operand, or literal that is not a fixed-point, floating-point, or decimal constant. The character string inserted in the message consists of up to eight characters starting with the illegal exponent modifier.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete the exponent modifier or change the type specification.

IPK127 SCALE MODIFIER USED ILLEGALLY, 'xxxxxxxx'

Explanation: A scale modifier is specified for a DC or DS instruction operand or literal, which is not a fixed-point, floating-point, or decimal type constant.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete the scale modifier, or change the type specification.

IPK128 CONSTANT FIELD MISSING OR PRECEDED BY INVALID FIELD, 'xxxxxxxx'

Explanation: In a DC instruction operand or literal, either illegal characters are found between the modifier and constant subfield, or the constant subfield does not contain any nominal value. The first character in the string indicates the point where the illegal constant value was found.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete the invalid characters, or supply a nominal value, or change the operation code to DS if you only want to specify a data area.

IPK129 INVALID DUPLICATION FACTOR OR MODIFIER, 'xxxxxxxx'

Explanation: a syntax error was found in the duplication factor subfield (subfield 1) or the modifier subfield (subfield 3) of this DC or DS instruction operand or literal. The first character in the string indicates the point where the illegal constant value was found.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the syntax in the expression is correct.

IPK130 S-TYPE CONSTANT IN LITERAL

Explanation: An S-type constant is specified in a literal.

Assembler Action: Zeros are generated in the output module.

Programmer Response: Use a DC instruction to specify the S-type constant, and supply the name of the DC instruction in the operand field instead of the literal, or change the type of the constant in the literal.

IPK131 ILLEGAL USE OF LITERAL

Explanation: A literal is used illegally. A literal can only be used as a relocatable operand in a machine instruction; and it cannot be used in an operand that represents the receiving field of an instruction or in a shift or I/O instruction.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Use a DC or DS instruction instead.

IPK132 LENGTH OR DUPLICATION FACTOR ZERO IN LITERAL

Explanation: The length specified in the modifier field or the duplication field of this literal has a zero value. These fields must specify absolute values in literals.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the value is greater than zero.

IPK133 TOO MANY SYMBOLS IN STATEMENT

Explanation: More than 50 symbols have been specified in an ENTRY, EXTRN, or WXTRN instruction.

Assembler Action: The first 50 symbols are processed.

Programmer Response: Place the excessive operands in additional ENTRY, EXTRN, or WXTRN instructions.

IPK134 STATEMENT COMPLEXITY EXCEEDED, 'xxxxxxxx'

Explanation:

- More than one operand has been specified in a DC or DS instruction; or
- The constant subfield of a DC or DS operand or literal contains too many symbols or terms or both. The maximum number of symbols and terms that can be handled by the assembler is around 30.
  The first character in the string indicates the point where the illegal character was found.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure only one operand is coded, or break up the constant subfield into two or more statements.

IPK135 SYMBOL 'xxxxxxxx' PREVIOUSLY DEFINED

Explanation: The ordinary symbol defined in this instruction either by appearing in the name field, or by appearing in the operand field of an EXTRN or WXTRN instruction, has already been defined within the source module.

Assembler Action: The second definition is ignored.

Programmer Response: Supply a name that does not conflict with any other symbols in the program.

IPK136 ARITHMETIC OVERFLOW (IN OPERAND n)

Explanation: During the evaluation of an expression, a value has been reached which is outside of the range -2^{31} through $2^{31}-1$.

Assembler Action: The expression is ignored.

Programmer Response: Rearrange the terms of expression to avoid overflow. If necessary, use EQU statements to separate the expression into smaller expressions that can be evaluated separately and then combined.

IPK137 EXPRESSION COMPLEXLY RELOCATABLE (IN OPERAND n)

Explanation: A complexly relocatable expression is used in the operand field of an EQU, CNOP, or ORG instruction or in the modifier subfield of a DC or DS instruction operand.

Assembler Action: The statement is processed as comments.

Programmer Response: Correct the expression so that it is simply relocatable (EQU, ORG) or absolute (EQU, CNOP, modifiers).

IPK138 TOO FEW OPERANDS

Explanation: This statement requires more operands than are supplied.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the missing operand(s).

IPK139 INVALID DUPLICATION FACTOR (IN OPERAND n)

Explanation: The duplication factor is relocatable or zero in a DC literal.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply an absolute value (not zero if DC literal).

IPK140 INVALID LENGTH MODIFIER (IN OPERAND n)

Explanation: The length modifier is either too large, zero, or relocatable. The maximum value allowed for the length modifier varies with the type specified for this operand.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the length modifier is an absolute value in the range allowed for this type of constant.

IPK141 INVALID SCALE MODIFIER (IN OPERAND n)

Explanation: The expression used to specify the scale modifier is relocatable.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Make sure the scale modifier expression specifies an absolute value, and that any symbols used in it have been previously defined.

IPK142 INVALID EXPONENT MODIFIER (IN OPERAND n)

Explanation: The expression used to specify the exponent modifier is relocatable.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Make sure the exponent modifier expression specifies an absolute value, and that any symbols used in it have been previously defined.

IPK143 INVALID CNOP OPERAND

Explanation: One or both of the operands in this CNOP instruction are invalid. Only the following combinations are allowed: 0 and 4, 2 and 4, 0 and 8, 2 and 8, 4 and 8, and 6 and 8.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply one of the combinations listed in the explanation.

IPK144 INVALID END OPERAND

Explanation: The operand of the END instruction is invalid. It must be a simply relocatable expression whose value represents an address within an ordinary control section (that is, not a dummy or common control section) in this source module, or an external reference

Assembler Action: The operand field is ignored.

Programmer Response: Supply a valid operand as described in the explanation.

IPK145 RELOCATABLE TERM IN DIVIDE OR MULTIPLY OPERATION (IN OPERAND n)

Explanation: A relocatable term is used in a multiply or divide operation in an expression in this statement.

Assembler Action: The statement is processed as comments.

Programmer Response: Use only absolute terms in divide or multiply operations.

IPK146 NAME MISSING

Explanation: The name is missing in this EQU instruction.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid name field.

IPK147 INVALID START STATEMENT

Explanation:

- The operand field is not blank or an absolute value; or
- The START instruction does not identify the beginning of the first control section in this source module; it was preceded by another START instruction, a CSECT instruction, or a statement that causes an unnamed control section (private code) to be initiated.

Assembler Action: The statement is processed as a CSECt statement.

Programmer Response: Make sure any operand specified is an absolute value, and that the START instruction initiates the first control section in the source module.

IPK148 ILLEGAL SYMBOL 'xxxxxxxx' IN ENTRY STATEMENT

Explanation: The specified symbol is not defined as a relocatable symbol within an ordinary control section (not a dummy or common control section), or is defined as an EXTRN symbol or has already appeared as an entry statement.

Assembler Action: The symbol is ignored.

Programmer Response: Make sure the ENTRY operand is a valid external name as defined in the explanation.

IPK149 SYMBOL 'xxxxxxxx' NOT PREVIOUSLY DEFINED

Explanation: The specified symbol appears in an EQU, ORG, or CNOP operand or in the modifier subfield of a DC or DS instruction, but has not been defined prior to this use. These fields require that any symbols used in them are previously defined.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the definition of this symbol precedes this statement.

IPK150 VALUE OF ORG OPERAND LESS THAN CONTROL SECTION STARTING ADDRESS

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

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the operand of the ORG instruction is a positive relocatable expression, greater than the starting address in the control section.

IPK151 LOCATION COUNTER OVERFLOW

Explanation: The location counter value is greater or equal to X'FFFFFF'.

Assembler Action: The location counter is carried in three bytes. When overflow occurs, the location counter will not be updated and every statement which causes overflow will be flagged.

Programmer Response: The probable cause of the error is a high ORG instruction value or a high START instruction value. Correct the value or split up the control section.

IPK152 ORG OPERAND VALUE NOT WITHIN THIS CONTROL SECTION

Explanation: The operand of an ORG instruction is not a simply relocatable expression whose value falls within the current control section.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the resulting value of the expression in the operand field falls within the control section where the ORG is coded. Any relocatable symbols defined in other control sections must be paired (that is, each such term must be matched by another term from the same control section with the opposite sign).

IPK153 ILLEGAL USE OF LOCATION COUNTER REFERENCE

Explanation: A location counter reference (*) is used in the modifier subfield of a DC or DS instruction or literals or in the operand of a CNOP instruction.

Assembler Action: The statement is processed as comments.

Programmer Response: Remove the invalid location counter reference.

IPK154 TOO MANY ENTRY SYMBOLS

Explanation: The number of ENTRY operands specified in this source module exceeds 10.

Assembler Action: ENTRY operands encountered in the rest of the assembly are ignored.

Programmer Response: Reduce the number of ENTRY operands, or separate the module into two or more modules.

IPK155 TOO MANY EXTERNAL SYMBOLS

Explanation: Too many entries have been made in the external symbol dictionary. Only 255 entries can be made for the following: control sections, dummy sections, common control sections, and external references (EXTRN, WXTRN, V-type constants). ENTRY operand are not counted towards this maximum, but the number of entry operands must not exceed 100.

Assembler Action: No more symbols are entered in the external symbol dictionary. The rest of the source module is assembled as part of the control section currently being processed.

Programmer Response: Reduce the number of ESD items, or separate the source module into two or more modules.

IPK156 SYMBOL 'xxxxxxxx' UNDEFINED

Explanation: The specified symbol has not been defined within the module; that is, it has not appeared in the name field of an instruction or in the operand field of an EXTRN or WXTRN instruction.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Make sure the symbol is defined, or use a symbol that has already been defined.

IPK157 ERROR IN DEFINITION OF LITERAL

Explanation: An error has been detected in the definition of the literal. The error can be either duplication factor, length, scale, exponent, location counter, or arithmetic overflow error, or a symbol not previously defined.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Correct the literal.

IPK158 TOO MANY OPERANDS

Explanation: Too many operands have been coded for this statement.

Assembler Action: If the statement is an assembler instruction, the excessive operands are ignored. If the statement is a machine instruction, zeros are generated in the object module.

Programmer Response: Delete the excessive operands; make sure that the format of the operand field is correct.

IPK159 TOO FEW OPERANDS

Explanation: This instruction requires more operands than are specified.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Supply the right number of operands; make sure that the format of the operand field is correct.

IPK160 COMPLEXLY RELOCATABLE EXPRESSION IN CONSTANT n ,OPERAND m

Explanation: A complexly relocatable expression has been used in an operand or S-type constant where a simply relocatable or absolute expression is required.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Supply a valid simply relocatable or absolute expression.

IPK161 OPERAND n NOT A CURRENT BASE REGISTER

Explanation: The register specified in the operand of this DROP instruction is not a current base register, either because it has not been specified as a base register by a previous USING instruction, or because it has been encountered in a previous DROP instruction.

Assembler Action: The operand is ignored.

Programmer Response: Make sure the operand is currently being used as a base register.

IPK162 INCONSISTENT VALUES IN OPERANDS 1 AND 2

Explanation: The values in operands 1 and 2 of this USING instruction are inconsistent. If the second operand is 0, the first operand must not specify an absolute value other than 0.

Assembler Action: The second operand is ignored.

Programmer Response: Change the first operand so that it specifies a relocatable value or the absolute value 0, or change the value of the second operand, so that it does not specify register 0.

IPK163 ADDRESSABILITY ERROR IN CONSTANT m OPERAND n

Explanation: An address specified in the operand of this statement is not covered by any base register, that is, it does not appear in the range of a USING instruction; that is, it does not appear in the range of a USING instruction.

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

Programmer Response: Make sure the address of the symbol in the operand falls within the ranges of a address instruction; it must be within the first 4,095 bytes of the address specified in the USING instruction.

IPK164 REGISTER VALUE IN OPERAND n NOT 0 OR 4

Explanation: The indicated operand does not specify the right register value; the value must be either 0 or 4.

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

Programmer Response: Supply the value 0 or 4.

IPK165 REGISTER VALUE IN OPERAND n NOT EVEN

Explanation: The indicated operand does not specify an even register value.

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

Programmer Response: Specify an even numbered register in the range 0-14, or for floating-point instructions, in the range 0-6.

IPK166 REGISTER VALUE IN OPERAND n OUT OF RANGE

Explanation: The register number specified in this operand is not in the range required by this instruction.

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

Programmer Response: Specify a value in the range 0-15, or for CLCL, MVCL, shift double instructions, and operand R1 of M, MR, D, and DR instructions, in the range 0-14, or, for floating-point instructions, in the range 0-6.

IPK167 REGISTER VALUE IN OPERAND n NOT ABSOLUTE

Explanation: The register number specified in this operand is not an absolute value.

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

Programmer Response: Specify an absolute value.

IPK168 MASK VALUE IN OPERAND n OUT OF RANGE

Explanation: The value specified as a mask in this operand is not in range 0-15.

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

Programmer Response: Specify a value in the range 0-15.

IPK169 MASK VALUE IN OPERAND n NOT ABSOLUTE

Explanation: The value specified as a mask in this operand is relocatable.

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

Programmer Response: Specify an absolute value in the range 0-15.

IPK170 IMMEDIATE VALUE IN OPERAND n OUT OF RANGE

Explanation: The value specified as an immediate value is negative or too high. For an SRP instruction the allowable range is 0-9, and, for other instructions the allowable range is 0-255.

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

Programmer Response: Specify an absolute value in the range described in the explanation.

IPK171 IMMEDIATE VALUE IN OPERAND n NOT ABSOLUTE

Explanation: The value specified as an immediate value is relocatable.

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

Programmer Response: Supply an absolute value in the range 0-9 (for SRP) or 0-255 (for other instructions).

IPK172 DISPLACEMENT VALUE IN {CONSTANT / OPERAND n} OUT OF RANGE

Explanation: The displacement value in the specified operand or S-type constant is not in the range 0-4095.

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

Programmer Response: Make sure the displacement is specified as an absolute value in the range 0-4,095.

IPK173 DISPLACEMENT VALUE IN {CONSTANT / OPERAND n} NOT ABSOLUTE

Explanation: The displacement value in the specified operand or S-type constant is relocatable.

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

Programmer Response: Make sure the displacement is specified as an absolute value in the range 0-4,095.

IPK174 INDEX REGISTER VALUE IN OPERAND n OUT OF RANGE

Explanation: The value specified in the index register subfield of this operand is not in the range 0-15.

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

Programmer Response: Make sure the index register is specified as an absolute value in the range 0-15.

IPK175 INDEX REGISTER VALUE IN OPERAND n NOT ABSOLUTE

Explanation: The value specified in the index register subfield of this operand is relocatable.

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

Programmer Response: Make sure the index register is specified as an absolute value in the range 0-15.

IPK176 BASE REGISTER VALUE IN {CONSTANT / OPERAND n} OUT OF RANGE

Explanation: The value specified in the base register subfield of this operand or S-type constant is not in the range 0-15.

Assembler Action: Zeros are generated instead of the instruction or constant in the object module.

Programmer Response: Make sure the base register is specified as an absolute value in the range 0-15.

IPK177 BASE REGISTER VALUE IN {CONSTANT / OPERAND} n NOT ABSOLUTE

Explanation: The value specified in the base register subfield of this operand or S-type constant is not absolute.

Assembler Action: Zeros are generated instead of the instruction or constant in the object module.

Programmer Response: Make sure the base register is specified as an absolute value in the range 0-15.

IPK178 LENGTH VALUE IN OPERAND n OUT OF RANGE

Explanation: The value specified in the length subfield of this operand is negative or too high. In decimal arithmetic and the SRP instruction, it must be in the range 0-16, and for logical operations, it must be in the range 0-256.

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

Programmer Response: Make sure the length specification is an absolute value in the range 0-16 (in decimal arithmetic instructions and the SRP instruction) or 0-256 (for logical operations).

IPK179 LENGTH VALUE IN OPERAND n NOT ABSOLUTE

Explanation: The value specified in the length subfield of this operand is relocatable.

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

Programmer Response: Make sure the length specification is an absolute value in the range 0-16 (in decimal arithmetic instructions and the SRP instruction) or 0-256 (for logical operations).

IPK180 INDEX REGISTER OR LENGTH VALUE SPECIFIED ILLEGALLY IN {CONSTANT / OPERAND} n

Explanation: An index register or length subfield has been specified in this operand, but the instruction does not allow any of these subfields to be specified.

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

Programmer Response: Delete the invalid subfield.

IPK181 REGISTER VALUE 0 USED IN OPERAND OTHER THAN THE SECOND

Explanation: In this USING instruction, register 0 is specified in an operand that is not the second operand. If base register 0 is to be specified in a USING instruction, it must be specified as operand 2.

Assembler Action: The operand is ignored.

Programmer Response: If you want to use register 0, make sure it is specified as the second operand.

IPK182 ALIGNMENT ERROR IN OPERAND n

Explanation: This operand refers to a storage location that is not properly aligned.

Assembler Action: Object code is generated.

Programmer Response: If the instruction flagged is not a system control instruction (SCI), the message is to be considered as a warning message only. In the case of an SCI the data area addressed by the instruction should be moved to a storage boundary required by the instruction.

On some machines an ordinary instruction will execute more slowly if the data area referenced is not aligned properly, even if alignment is not required for correct execution. In such cases it might be desirable to move the data area as described above.

Refer to the publication; System/370 Principles of Operation, Order No. GA22-7000 for details on the boundary requirements of this instruction.

IPK183 SUBFIELD SPECIFIED ILLEGALLY IN OPERAND n

Explanation: A subfield has been specified in this operand. However, the operand does not allow any subfields; it must consist of only one expression.

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

Programming Response: Make sure the operand follows the format required by this instruction.

IPK184 EXPONENT MODIFIER OUT OF RANGE IN CONSTANT n (OPERAND m)

Explanation: The value of the exponent modifier is too large or too small. The sum of the exponent modifier and the exponent specification in the constant must be in the range -85 - +75.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Make sure the total value of the exponent in the constant subfield and the exponent modifier in the modifier subfield is in the range of -85 through +75.

IPK185 SCALE MODIFIER OUT OF RANGE IN CONSTANT n (OPERAND m)

Explanation: The scale modifier is either too large or too small. For a fixed-point constant, the allowed range is -187 through +346. For an E- or D-type constant, the allowed range is 0 through 14, and for an L-type constant, the allowed range is 0 through 28.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Make sure the scale modifier value falls in the range described in the explanation.

IPK186 CHARACTERISTIC OUT OF RANGE IN CONSTANT n (OPERAND m)

Explanation: A converted floating-point constant is too large or too small for the field assigned to it. The allowable range is $7.2x10^{75}$ to $5.3x10^{-77}$.

Assembler Action: Zeros are generated in the object module.

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.

IPK187 PRECISION LOST IN CONSTANT n (OPERAND m)

Explanation: The mantissa (fraction) is lost during the construction of an L-, D-, or E-type constant, because the designated field is too small to contain any part of the mantissa after scaling.

Assembler Action: The mantissa is set to zero.

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

IPK188 INVALID SYNTAX IN DATA FIELD OF CONSTANT n (OPERAND m)

Explanation: The syntax is invalid in the present constant subfield of this operand. For instance, an E is present to designate an exponent, but no exponent is found.

Assembler Action: Zeros are generated in the source module.

Programmer Response: Correct the syntax of the statement.

IPK189 DATA ITEM TOO LARGE IN CONSTANT n (OPERAND m)

Explanation: The constant specified in the constant subfield of this DC or DS instruction operand or literal is too large for the data type or for the length specified explicitly in the length modifier.

Assembler Action: The value is truncated on the left.

Programmer Response: Change the type specification or the length modifier.

IPK190 LENGTH MODIFIER ILLEGAL WITH CONSTANT n (OPERAND m)

Explanation: An A-, or Y-type address constant has been specified with an explicit length which is correct for absolute, but not for relocatable expressions.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Change length modifier to allow expression to be relocatable or make expression absolute.

IPK191 ILLEGAL EXPRESSION IN ADDRESS CONSTANT n (OPERAND m)

Explanation: Only a simple expression is allowed in an address constant (no subfields).

Assembler Action: Zeros are generated in the object module.

Programmer Response: Supply a simple expression.

IPK192 ILLEGAL EXPRESSION IN CCW OPERAND n

Explanation: Only a simple expression is allowed as a CCW operand (no subfield allowed).

Assembler Action: Zeros are generated in the object module.

Programmer Response: Supply a simple expression.

IPK193 OPERAND 3 INVALID

Explanation: The last three bits in operand 3 of a CCW instruction are not specified as zeros.

Assembler Action: The operand is accepted as it is specified.

Programmer Response: Supply an operand 3 value in which the last three bits are zeros.

IPK194 TOO FEW OPERANDS

Explanation: Less than four operands found in a CCW instruction.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Supply the missing operand(s).

IPK195 OPERAND n NOT ABSOLUTE

Explanation: The value specified in operands 1, 3, or 4 not an absolute value.

Assembler Action: Zeros are generated instead of the CCW in the object module.

Programmer Response: Make sure the values of the expressions in operands 1, 3, and 4 are absolute.

IPK196 TOO MANY OPERANDS

Explanation: More than four operands have been found in a CCW instruction.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Delete the excessive operand(s).

IPK197 VALUE OF OPERAND n OUT OF RANGE

Explanation: The value specified for the operand identified in the message is too high or negative. The value of operand 1 must be in the range 0-255, the value of operand 3 must be in the range 0-248, and the value of operand 4 must be in the range 0-65,535.

Assembler Action: Zeros are generated instead of the CCW in the object module.

Programmer Response: Make sure the operand specifies an absolute value in the range described in the explanation.

IPK198 SYMBOL IN CCW ADDRESS OPERAND DEFINED IN DUMMY SECTION

Explanation: A symbol a in CCW address operand is defined in a dummy section. If a symbol in an expression in address operand is defined in a dummy section, the symbol must be paired with another symbol with the opposite sign defined in the same dummy section.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Delete any symbols in a CCW address operand defined in dummy sections, or make sure they are paired with other symbols defined in the same dummy section.

IPK199 DUMMY SECTION SYMBOL USED ILLEGALLY IN CONSTANT n (OPERAND m)

Explanation: A dummy section symbol appearing in the constant subfield of this address constant is defined in a dummy section. If a symbol in an expression in the constant subfield is defined in the dummy section, the symbol must be paired with another symbol with the opposite sign defined in the same dummy section.

Assembler Action: Zeros are generated in the object module.

Programmer Response: Delete any dummy section symbols, or make sure they are paired with other symbols defined in the same dummy section.

IPK200 NAME FIELD TOO LONG

Explanation: The length of the symbol in the name field exceeds eight characters.

Assembler Action: The name field is ignored.

Programmer Response: Make sure the name field is not longer than eight characters.

IPK201 NAME FIELD NOT SEQUENCE SYMBOL OR BLANK

Explanation: The name field contains something other than a valid sequence symbol or blank. The following instructions must have a blank or a sequence symbol in the name field: EJECT, PRINT, SPACE, MNOTE, PUNCH, REPRO, and TITLE (except the first TITLE statement in the module).

Assembler Action: The name field is ignored.

Programmer Response: Supply a valid sequence symbol, or leave the name field blank.

IPK202 TITLE NAME TOO LONG

Explanation: The name field of the first TITLE instruction in the program contains more than four characters that are used to specify a valid sequence symbol.

Assembler Action: The name field is ignored.

Programmer Response: Supply up to four alphameric characters in the name field, or leave the name field blank.

IPK203 TITLE NAME CONTAINS NON-ALPHAMERIC CHARACTER

Explanation: A character that is not an alphameric character has been encountered in the name field of the first TITLE statement in the program.

Assembler Action: The name field is ignored.

Programmer Response: Supply one to four alphameric characters, or leave the name field blank.

IPK204 OPERAND MISSING

Explanation: The operand field of a PRINT, PUNCH, or TITLE statement is blank.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a valid operand field.

IPK205 FIRST APOSTROPHE MISSING

Explanation: The first apostrophe in the operand of an MNOTE, PUNCH, or TITLE instruction is missing.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure the operand is a character combination enclosed in apostrophes.

IPK206 SINGLE AMPERSAND IN OPERAND

Explanation: A single ampersand which is not part of a variable symbol appears in the MNOTE, PUNCH, or TITLE operand.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure that an ampersand that is meant to be part of the operand rather than of a variable symbol in the operand is coded as a double ampersand.

IPK207 LAST APOSTROPHE MISSING

Explanation: The operand of an MNOTE, PUNCH, or TITLE instruction does not end with a single apostrophe.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the closing apostrophe.

IPK208 TITLE OR PUNCH OPERAND TOO LONG

Explanation: The operand of a TITLE or PUNCH instruction is too long. The maximum length of the TITLE operand is 100 characters, excluding the enclosing apostrophes, and the maximum length of the PUNCH operand is 80 characters, excluding the enclosing apostrophes.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply an operand which does not exceed the length described in the explanation.

IPK209 OPERAND FIELD ILLEGALLY TERMINATED

Explanation: The closing apostrophe of an MNOTE, PUNCH or TITLE operand is not immediately followed by a blank. This message can be caused by a single apostrophe coded or generated inside the enclosing apostrophes or by a missing blank between the operand field and the remarks field.

Assembler Action: The statement is processed as comments.

Programmer Response: Make sure all apostrophes inside the enclosing apostrophes are coded as double apostrophes, or supply the missing blank between the operand and the remarks field.

IPK211 NON-DECIMAL CHARACTER IN OPERAND

Explanation: The operand of a SPACE instruction contains non-decimal characters or the severity code operand of an MNOTE instruction contains characters that are not decimal or an asterisk.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a decimal value on an asterisk (for MNOTE only).

IPK212 INVALID PRINT OPERAND

Explanation: The operand of a PRINT instruction does not specify one or more of the following values: ON, OFF, GEN, NOGEN, DATA, NODATA.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply from one to three operands that do not conflict with each other. The operands are listed in the explanation.

IPK213 CONFLICTING PRINT OPERANDS

Explanation: Conflicting operands have been specified in a PRINT statement. Only one value from each of the following three pairs can be specified: ON/OFF, GEN/NOGEN, and DATA/NODATA.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete conflicting values.

IPK214 'x' IS AN INVALID DELIMITER

Explanation: An operand in a PRINT statement is not immediately followed by a comma or a blank.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply the correct delimiter.

IPK215 OPERAND FIELD INCOMPLETE

Explanation: A PRINT instruction ends with a comma followed by a blank, or an MNOTE instruction contains a severity code operand, but no message operand.

Assembler Action: The statement is processed as comments.

Programmer Response: Delete the comma, or supply the additional operand.

IPK216 MNOTE GENERATED

Explanation: An MNOTE statement specified with a severity code, or an explicitly omitted (by means of a comma) severity code has been encountered.

Assembler Action: Processing continues.

Programmer Response: Determine the cause of the message by referring to the source statements section of the listing. The MNOTE message will be written at the statement number supplied with the message.

IPK217 MNOTE SEVERITY VALUE TOO HIGH

Explanation: The severity code specified in the first operand of an MNOTE instruction is greater than 255.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply a severity code in the range 0-255, or omit the first operand.

IPK218 NULL STRING IN PUNCH OPERAND

Explanation: The operand field of a PUNCH statement contains only two apostrophes placed immediately after each other.

Assembler Action: The statement is processed as comments.

Programmer Response: Supply 1-80 characters inside the apostrophes.

IPK230 PERMANENT I/O ERROR ON SYS00x

Explanation: An unrecoverable I/O error occurred on the device to which this file is assigned.

Assembler Action: The assembly is terminated. No listing is produced.

Programmer Response: Re-assemble the program.

Operator Response: Rerun the job using a different device for the file indicated in the message.

IPK231 INVALID DEVICE FOR SYS00x

Explanation: The device assigned for this file cannot be used as a work file by the assembler.

ASsembler Action: The assembly is terminated. No listing is produced.

Programmer Response: If you have supplied an ASSGN statement for this work file, correct the ASSGN statement so that it specifies a direct-access device that can be used by the assembler (see the section "Files Used by the Assembler"). If you have not supplied any ASSGN statement, rerun the job, making sure that the workfiles are assigned to direct-access storage devices.

Operator Response: Use the ASSGN command to assign the indicated file to a direct-access storage device, and rerun the job.

IPK232 SYSxxx NOT ASSIGNED

Explanation: This file is required by the assembler, either because it is a work file, or because it is required by an option specified in the OPTION statement, but the file is not assigned or the IGNORE option is specified for the file. The IGN option is valid only for SYSPCH and SYSLST.

Assembler Action: The assembly is terminated. No listing is produced.

Programmer Response: Rerun the job, making sure that the indicated file is assigned, or change the corresponding option on the OPTION statement. OR: Execute the LISTIO command and verify the assignments. Submit an ASSGN command for the file indicated in the message, and rerun the job.

IPK233 ASSEMBLER PARTITION TOO SMALL/DE-EDITOR PARTITION TOO SMALL

Explanation: The number of bytes allocated for the assembler are not enough. The assembler must not be loaded into less than 20K bytes (26K for the de-editor). Note that in a foreground partition the assembler is always loaded immediately after the save area.

Assembler Action: The assembly is terminated. No listing is produced.

Programmer Response: Specify a larger partition for the job and rerun it.

Operator Response: Use the ALLOC command to increase the size of the partition and rerun the job.

IPK234 END OF EXTENT FOR SYS00x

Explanation: The direct access storage extent assigned for this file is not large enough. Note that multiple extents are not used for an assembler work file.

Assembler Action: The assembly is terminated. No listing is produced.

Programmer Response: If you have supplied DLBL and EXTENT statements for the file in your job, increase the extent specified in the EXTENT statement and rerun the job. If not, check the LSERV output to make sure that the standard assignment for this file specifies an extent that is large enough. If you do not want to change the EXTENT size, separate the program into two or more source modules, and assemble each module separately. OR: If the standard assignment for the file indicated in the message we are used by this job, execute LSERV, and return the output to the programmer.

IPK236 ASSEMBLER CANNOT CONTINUE/DE-EDITOR CANNOT CONTINUE

Explanation:

- If this message is preceded by other messages the preceding message explains the reason why the assembler cannot continue.
- If the message is not accompanied by other messages, an error in the logic of the assembler has been encountered.

Assembler Action: Assembly is terminated. No listing is produced. If the message is caused by an error in the assembler, a main storage dump of the assembler area is given.

Programmer Response: If the message is caused by another error message, perform the actions indicated in the description of that message. Otherwise, save your job stream, SYSLOG listing and SYSLST listing to aid in problem determination, before calling IBM.

Operator Response: If the message is preceded by another error message, ignore this message, and perform the actions indicated for this message. If this message appears alone, consider the preceding job as terminated.

IPK240 TOO MANY MACROS

Explanation: The capacity of the assembler is exceeded.

Assembler Action: All statements will be treated as comments.

Programmer Response: Separate the source module into smaller modules, and assemble each module separately.

IPK241 TOO MANY GLOBAL VARIABLE SYMBOLS

Explanation: The partition allocated to the assembler is not large enough to process the source module because too many global symbols have been used.

Assembler Action: All statements will be treated as comments.

Programmer Response: Increase the size of the partition, or reduce the number of global symbols by grouping them together in SET symbol arrays (subscripted SET symbols).

IPK242 INCONSISTENT TYPE OF GLOBAL VARIABLE SYMBOL 'xxxxxxxx' IN 'yyyyyyyy'

Explanation: The type of variable symbol specified in the declaration is inconsistent with the type specified in another macro or in open code. For example, if a global symbol is declared as a SETA symbol in one macro definition, it must be declared as a SETA symbol in all macro definitions where it is used.

Assembler Action: All declarations inconsistent with the first declaration are considered invalid. The macro definitions are processed in the order in which they appear in the source, with all outer macros first, followed by the inner macros of the first level, inner macros of the second level, etc. Open code is processed last.

Programmer Response: Make sure all global declarations are consistent.

IPK243 INCONSISTENT DIMENSION OF GLOBAL VARIABLE SYMBOL 'xxxxxxxx' IN 'yyyyyyyy'

Explanation: Either the dimensions specified in declarations of global variable symbols are different in different macros and/or open code, or a global symbol is declared as dimensional in one macro definition and undimensional in another.

Assembler Action: All declarations inconsistent with the first declaration encountered are ignored. The macro definitions are processed in the order in which they appear in the source, with all outer macros first, followed by all inner macros of the first level, all inner macros of the second level, etc. Open code is processed last.

Programmer Response: Make sure all global declarations are consistent.

IPK244 SYSSLB RECORD 'nnn' IN MACRO 'xxxxxxxx' NOT IN SEQUENCE

Explanation: This library macro definition was not in the proper order when it was cataloged; the specified record was out of sequence.

Assembler Action: The macro is not generated.

Programmer Response: Catalog the macro definition again, making sure that all the records are in the right sequence.

IPK245 MACRO 'xxxxxxxx' CATALOGED UNDER DIFFERENT NAME 'yyyyyyyy'

Explanation: This library macro definition was not cataloged under the right name; the name under which a macro is cataloged must always be identical to the operation code of the macro as it is specified in the macro prototype statement.

Assembler Action: The macro is not generated.

Programmer Response: Catalog the macro under its own name (operation code), or change the operation code to match the book name.

IPK246 UNEXPECTED END-OF-FILE ON SYSSLB AT RECORD 'nnn' IN MACRO 'xxxxxxxx'

Explanation: End-of-file was encountered in the source statement library before the end of a book had been reached, or record length is greater than 80 bytes. Since the end-of-file indicator is normally only found at the end of the sublibrary, the message indicates that the source statement library has been destroyed.

Assembler Action: The macro is not generated.

Programmer Response: Re-construct the source statement library.

IPK247 UNEXPECTED END OF BOOK AT RECORD 'nnn' IN LIBRARY MACRO 'xxxxxxxx'

Explanation: Some cards at the end of this definition were missing in this macro definition when it was cataloged.

Assembler Action: The macro is not generated.

Programmer Response: Catalog a complete version of the macro definition.

IPK248 'xxxxxxxx' NOT AN EDITED MACRO

Explanation: The macro library book that corresponds to the specified operations code is not recognized as an edited macro.

Assembler Action: The macro is not generated.

Programmer Response: Catalog an edited version of the macro definition.

IPK250 ERRORS FOUND IN MACRO 'xxxxxxxx', EDECK NOT PUNCHED

Explanation: Since errors were found in this macro definition, no edited macro is punched, even though that is requested by means of the EDECK option.

Assembler Action: The EDECK option is ignored for this macro; no edited version is punched.

Programmer Response: Correct the errors in the macro definition and assemble again.

ESERV Messages: IPK301–IPK332

IPK301 INVALID SELECT CARD

Explanation:
1. The opcode in the macro select card is not recognized as any of DSPLY, DSPCH, or PUNCH or
2. The operand field is filled up but the last operand contains only sublibrary name and no bookname.

ESERV Action: Next macro select card is read.

Programmer Response: Correct opcode.

IPK302 xxxxxxxx NOT FOUND ON LIBRARY X

Explanation: The book referred to in the above message was not found in the library specified.

ESERV Action: The next book is looked for.

Programmer Response: Correct the name of book on library.

IPK303 INVALID MACRO HEADER

Explanation: The first record of the selected book is not a header of an edited macro. The edited deck could be damaged, or the selected book is not in edited format. A copy book or a source macro definition could have been selected by mistake.

ESERV Action: The next book to be selected in the ESERV run is looked for.

Programmer Response: Check if an edited macro really is cataloged by that name. Was right sublibrary specified? If edited deck was damaged, see Note 1 at the back of this message section.

IPK304 TOO LONG BOOK NAME

Explanation: A character string with more than 8 characters with no blank or comma is encountered in the operand field of a macro select card.

ESERV Action: Next book is looked for.

Programmer Response: Correct the name of the macro in the macro select card.

IPK305 SYSPCH NOT ASSIGNED

Explanation: A macro select card with an opcode of DSPCH or PUNCH is encountered, but SYSPCH has not been assigned.

ESERV Action: PUNCH option is ignored and the de-editing continues in DSPLY mode.

Programmer Response: Assign SYSPCH.

IPK306 EDECK SERIOUSLY DAMAGED. DE-EDITING TERMINATED

Explanation: The edited macro deck has been so seriously damaged that further de-editing is not meaningful.

ESERV Action: Next macro to be de-edited is looked for.

Programmer Response: See Note 1 at the back of this message section.

IPK307 NON-BLANK CHARACTER IN COL 72

Explanation: This is a warning message. The ESERV program processes only columns 1-71 of the control records; however, column 72 will be printed as blank.

ESERV Action: Execution continues. ESERV ignores column 72.

Programmer Response: None.

IPK311 CARD nnnn OUT OF ORDER

Explanation: The cards in the EDECK are out of order; a card with lower sequence number than the preceding card has been encountered.

ESERV Action: Misplaced card is ignored, next card is read, and de-editing continues.

Programmer Response: See Note 1 at the back of this message section. Possibly the COL statement is using the wrong fields.

IPK312 CARDS(S) MISSING, nnnn-nnnn

Explanation: One or more cards are missing in the EDECK.

ESERV Action: De-editing continues with the next card.

Programmer Response: See Note 1 at the back of this message section.

IPK321 SEQUENCE NUMBER BEYOND END OF MACRO

Explanation: MEND statement has been de-edited while a macro definition statement corresponding to an operand in the preceding update control has not been found. This might depend on: 1. Referenced sequence number is not present in the macro definition. 2. The sequence field in the macro definition statement is not located in the columns specified in the COL statement.

ESERV Action: The remaining update control cards are flushed till the) END.

Programmer Response: Use the de-edited output of the macro definition to check the sequence field of the card.

IPK322 INVALID UPDATE CONTROL CARD

Explanation: A blank does not immediately follow the right parenthesis in column 1 of the card.

ESERV Action: SYSIPT is flushed to the next update control card, and updating continues.

Programmer Response: Correct or remove card in error.

IPK323 INVALID OPERATION IN UPDATE CONTROL CARD

Explanation: The operation field of the control card contains something other than COL, VER, ADD, DEL, REP, RST, or END.

ESERV Action: SYSIPT is flushed to the next update control card, and updating continues.

Programmer Response: Correct or remove card in error.

IPK324 CONTROL CARD OUT OF SEQUENCE

Explanation:

1. The first operand of a VER, ADD, DEL, REP, or RST card is smaller than the last operand of the preceding control card or,

2. The first operand of an ADD, DEL, REP, or RST card is equal to the last operand in the preceding control card which is not a VER statement, or

3. Two consecutive VER cards have the same first operand.

ESERV Action: SYSIPT is flushed to the next update control card and updating continues.

Programmer Response: Put control cards in ascending order, or correct or remove card in error.

IPK325 MISMATCH IN SELECTED FIELD

Explanation: Source card following VER cards does not match referenced statement in macro definiton.

ESERV Action: SYSIPT is flushed to the next update control card. If this control card refers to the same macro statement, it is flagged as invalid. SYSIPT is flushed to the next control card and updating continues.

Programmer Response: Use the de-edited output of the macro definition to check if the version is the expected one.

IPK326 INVALID OPERAND IN UPDATE CONTROL CARD

Explanation:

1. An operand in the control card is invalid, or

2. The second operand of a DEL or REP card is smaller than the first operand.

ESERV Action: SYSIPT is flushed to the next update control card and updating continues.

Programmer Response: Correct or remove card in error.

IPK327 END OF MACRO BEFORE END OF UPDATE CARDS

Explanation: Update control cards other than) END remain when MEND statement has been de-edited/updated.

ESERV Action: Remaining update control cards are flushed to) END.

Programmer Response: Remove or re-position cards in error.

IPK328 UPDATE TERMINATED, SYSIPT READ TO) END

Explanation: Informative message that appears after messages IPK321, IPK327, and IPK330.

ESERV Action: Not applicable.

Programmer Response: Not applicable.

IPK329 UNEXPECTED EOF SYSIPT;) END MISSING

Explanation: End of file was met before) END card was read.

ESERV Action: De-editing continues.

Programmer Response: Insert missing) END card in update deck.

IPK330 INVALID COL STATEMENT

Explanation:
1. An operand of the COL card is invalid, or

2. The COL statement is not the first update control statement.

ESERV Action: Updating will not be performed. The remaining update control cards are flushed till) END.

Programmer Response: Correct the operand or put the COL statement first in the update deck.

IPK331 SEQUENCE NUMBER IS TOO SMALL

Explanation: A sequence number in a macro definition statement is found to be greater than a sequence number reference in the preceding update control card. This might depend on:

1. Referenced sequence number is not present in the macro definition or

2. The sequence number is not present in the macro definition or

3. The sequence field in the macro statements are not located in the columns specified in the COL statement.

ESERV Action: The requested update action is ignored or terminated. SYSIPT is flushed to the next update control card and updating continues.

Programmer Response: Use the de-edited output of the macro definition to check the sequence field of the statements.

IPK332 SECOND OPERAND FOUND BEFORE FIRST

Explanation: There is no syntactical error in the control card. However, relative addressing has been used in such a way that the last macro statement to be deleted/replaced has been found before the first one.

ESERV Action: The referenced macro statement is deleted/replaced. Updating continues.

Programmer Response: Correct the update control card.

Note 1: Any of the messages IPK303, IPK306, IPK311, and IPK312 could be given if the cards in the EDECK are out of order. Run the SSERV program with DSPCH option and HEX parameter to produce a printout and a punched deck of the edited macro. Put the cards in ascending order by sequence numbers. Catalog the EDECK and run ESERV again. Three things can happen:

1. The macro is successfully de-edited.

2. Message IPK312 is given because cards are still missing. Use the de-edited output and try to reconstruct the source macro definition, by comparing it with a listing of the source macro definition.

3. Message IPK303 or IPK306 will occur. EDECK is heavily damaged that it cannot be used.

Note 2: The possibility of the ESERV program to process damaged EDECKs is restricted to the cases of missing cards and/or cards out of sequence. More serious types of destruction, like garbage punched in the edited text, will give unpredictable results from the de-editing.

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.

- Definitions made by the American National Standards Institute (ANSI). Such definitions are marked by an (*). 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 Standards Committee X3.

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

*assemble: to prepare a machine language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

*assembler: A computer program that assembles.

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

assembler language: A source language that includes symbolic machine language statements in which there is a one-to-one correspondence with the instruction formats and data formats of the computer. The Assembler language also contains statements that represent assembler instructions and macro instructions.

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

book: A group of source statements written in the assembler language and resident on either the macro or copy library.

*catalog: To enter a phase, module, or book onto one of the system libraries.

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

copy library: One of the two sublibraries of the source statement library (the other being the macro library) used by the assembler, which contains sequences of source code (books) that can be inserted into a source module by means of COPY statements.

de-editor program: A program used to convert an edited (partially processed) macro definition back to a format closely resembling source format.

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

edited macro: Source macro definition that has been partially processed by the assembler.

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).

ESERV: See de-editor program: ESERV.

execute (EXEC) statement: A job control language 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.

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

file: 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.

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.

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 step control statement. A job may specify that several job steps be executed.

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.

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.

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.

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; generates overlay structures on request; and produces executable code (a load module) that is ready to be fetched into main storage and executed.

load module: The output of a single linkage editor execution. A load module is in a format suitable for loading into virtual storage for execution.

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 library: One of the two sublibraries of the source statement library (the other being the copy library) used by the assembler, which contains IBM-supplied system macro definitions and user written macro definitions. In DOS/VS both types of definitions are in edited format.

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

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 modules).

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.

*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.

option: (See assembler option).

processing program:

1. A general term for any program that is not a control program.

2. Any program capable of operating in the problem program state. This includes IBM-distributed language translators, application programs, service programs, and user-written programs.

program:

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.

1. To write a program.

*relocation dictionary: The part of an object or load module that identifies all addresses that must be adjusted when a relocation occurs.

source macro definition: A macro definition included in a source module. A source macro definition can be entered into a program library; it then becomes a library macro definition. (See macro library, source library.)

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

source statement library: A collection of books (source macro definitions) cataloged onto the system by the Librarian. The assembler uses two such collections of books, known as the copy library and macro library.

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

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.

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

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X

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)

Guide to the DOS/VS Assembler

Order No. GC33-4024-1

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

CUT OR FOLD ALONG LINE

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)