IBM

Systems Reference Library

IBM System/360 Operating System
Assembler [F] Programmer's Guide

Program Number 360S-AS-037

This publication complements the IBM System/360
Operating System Assembler Language publication. It
provides a guide to program assembling, linkage
editing, executing, interpreting listings, assembler
programming considerations, diagnostic messages, and
object output cards.
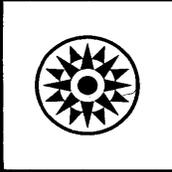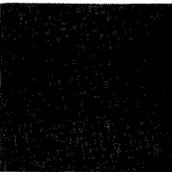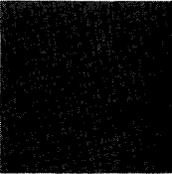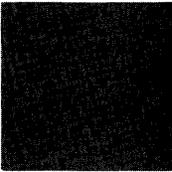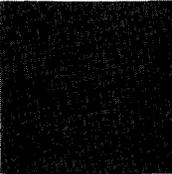
PREFACE

This publication is oriented to the F level assembler program (the assembler) functioning in the IBM System/360 Operating System (Primary Control Program, MFT, and MVT).

This publication is divided into an introduction and four sections which describe the following:

1. Assembler options and data set requirements.
2. Use of IBM-provided cataloged procedures for assembling; assembling and linkage editing; assembling, linkage editing, and executing assembler language source programs.
3. Use and interpretation of the assembler listing.
4. Programming considerations.

In addition, the appendixes provide a list and explanation of diagnostic messages, a description of object output cards, and a sample program listing.

Other System Reference Library publications in the IBM System/360 Operating System series provide fuller, more detailed discussions of the topics introduced in this publication: a careful reading of the publication IBM System/360 Operating System: Concepts and Facilities, Form C28-6535, is recommended. Knowledge of the assembler language is assumed. Where appropriate, the reader is directed to the following publications:

IBM System/360 Operating System: Job Control Language (Form C28-6539)

IBM System/360 Operating System: Storage Estimates (Form C28-6551)

IBM System/360 Operating System: Linkage Editor (Form C28-6538)

IBM System/360 Operating System: Supervisor and Data Management Services (Form C28-6646)

IBM System/360 Operating System: Supervisor and Data Management Macro-Instructions (Form C28-6647)

IBM System/360 Operating System: TESTRAN (Form C28-6648)

IBM System/360 Operating System: Messages, Completion Codes, and Storage Dumps (Form C28-6631)

IBM System/360 Operating System: Assembler Language (Form C28-6514)

IBM System/360 Operating System: Utilities (Form C28-6586)

IBM System/360 Operating System: FORTRAN IV (E), Library Subprograms (Form C28-6596)

IBM System/360 Operating System: System Programmer's Guide (Form C28-6550)

IBM System/360 Operating System: FORTRAN IV (E) Programmer's Guide (Form C28-6603)

IBM System/360 Operating System: COBOL (E) Programmer's Guide (Form C24-5029)

References to these publications are usually by a short title, e.g., Linkage Editor or Data Management Services.

CONTENTS

ILLUSTRATIONS

Figures

Tables

Through the medium of job control statements, the programmer specifies job requirements directly to the operating system, thus eliminating many of the functions previously performed by the operating personnel. The job consists of one or more job steps. For example, the job of assembling, linkage-editing, and executing a source program involves three job steps:

1. Translating the source program, i.e., executing the assembler component of the operating system to produce an object module.
2. Processing the output of the assembler, i.e., executing the linkage-editor component of the operating system to produce a load module.
3. Executing the assembled and linkage-edited program, i.e., executing the load module.

A procedure is a sequence of job control language statements specifying a job. Procedures may enter the system via the input stream or from a library of procedures, which are previously defined and contained in a procedure library. The input stream is the flow of job control statements and, optionally, input data entering the system from one input device. At the sequential scheduling system level of the operating system, only one input stream may exist at a time. (For a description of the operating system environment see IBM System/360 Operating System: Concepts and Facilities.)

The job definition (JOB), execute (EXEC), data definition (DD), and delimiter (/*) job control statements are shown in this publication as they are used to specify assembler processing. Detailed explanations of these statements are given in IBM System/360 Operating System: Job Control Language.

Operating system factors influencing program preparation, such as terminating the program, saving and restoring general registers, and linking of independently produced object modules, are discussed in Programming Considerations, as are guides to determine whether assembler dictionary sizes and complexity limitations of source statements will be exceeded.

ASSEMBLER OPTIONS AND DATA SET REQUIREMENTS

ASSEMBLER OPTIONS

The programmer may specify the following
assembler options in the PARM= field of
the EXEC statement. They must appear
between two apostrophes, separated by commas
with no imbedded blanks. They can appear
in any order and, if an entry is ommitted, a
standard setting will be assumed as shown
below under Default Entry.

```
          'DECK    LOAD,   LIST   TEST,   XREF,                RENT'
PARM=      or      or      or     or       or    LINECNT=nn,   or
          'NODECK,NOLOAD,NOLIST,NOTEST,NOXREF,              NORENT'
```

These options are defined as follows:
DECK -- The object module is placed on
    the device specified in the SYSPUNCH DD
    statement.
LOAD -- The object module is placed on
    the device specified in the SYSGO DD
    statement.

NOTE: Specification of the parameter
LOAD causes object output to be written
on a data set with ddname SYSGO. This
action occurs independently of the output
on SYSPUNCH caused by the parameter DECK.
The output on SYSGO and SYSPUNCH is iden-
tical except that SYSPUNCH is closed with
a disposition of LEAVE, and SYSGO is
closed with a disposition of REREAD.

LIST -- An assembler listing is produced.
TEST -- The object module contains the
    special source symbol table required
    by the test translator (TESTRAN)
    routine.
XREF -- The assembler produces a cross-
    reference table of symbols as part of
    the listing.
RENT -- The assembler checks for a possible
    coding violation of program re-
    enterability.

The prefix,NO, is used with the above
options to indicate which options are not
wanted. If contradictory options are
entered, e.g., LIST, NOLIST, the rightmost
option, NOLIST, is used.

LINECNT=nn This parameter specifies the
    number of lines to be printed between
    headings in the listing. The permis-
    sible range is 01 to 99 lines.

The following is an example of specify-
ing assembler options:

```
EXEC  PGM=IEUASM,PARM='LOAD,NODECK,TEST'
```

DEFAULT ENTRY

If no options are specified the assembler
assumes the following default entry:

```
PARM='NOLOAD,DECK,LIST,NOTEST,XREF,LINECNT=55,NORENT'
```

The cataloged procedures discussed in
this guide assume the default entry. How-
ever, the programmer may override any or
all of the default options (see Overriding
Statements in Cataloged Procedures).

ASSEMBLER DATA SET REQUIREMENTS

The assembler requires the following four
data sets:

● SYSUT1, SYSUT2, SYSUT3 -- utility data
    sets used as intermediate external
    storage.

● SYSIN -- an input data set containing
    the source statements to be processed.

In addition to the above, four additional
data sets may be required:

● SYSLIB -- a data set containing macro
    definitions (for macro definitions not
    defined in the source program) and/or
    source coding to be called for through
    COPY assembler instructions.

● SYSPRINT -- a data set containing output
    text for printing (unless NOLIST option
    is specified).

● SYSPUNCH -- a data set containing object
    module output usually for punching (un-
    less NODECK option is specified).

● SYSGO -- a data set containing object module output usually for the linkage editor (only if LOAD option is specified).

The above data sets are described in the following text. The ddname that must be used in the DD statement describing the data set appears as the heading for each description.

Ddnames SYSUT1, SYSUT2, SYSUT3

These utility data sets are used by the assembler as intermediate external storage devices when processing the source program. The input/output device(s) assigned to these data sets must be capable of sequential access to records. The assembler does not support multi-volume utility data sets. Refer to the Storage Estimate manual for the space required.

Ddname SYSIN

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

Ddname SYSLIB

From this data set, the assembler obtains macro definitions and assembler language statements to be called by the COPY assembler instruction. It is a partitioned data set and each macro definition or sequence of assembler statements is a separate member, with the member name being the macro instruction mnemonic or COPY code name. The data set may be defined as SYS1.MACLIB or a user's private macro definition or COPY library. SYS1.MACLIB contains macro definitions for the system macro instructions provided by IBM. A user's private library may be concatenated with SYS1.MACLIB. The two libraries must have the same attributes, i.e., the same blocking factors, block sizes, and record formats. The Job Control Language publication explains the concatenation of data sets.

Ddname SYSPRINT

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

Ddname SYSPUNCH

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

Ddname SYSGO

This is a DASD, magnetic tape, or card punch data set used by the assembler. It contains the same output text as SYSPUNCH. It is used as input for the linkage editor and may also be used as a punch device (see NOTE under Assembler Options).

DEFINING DATA SET CHARACTERISTICS

Before a data set can be made available to a problem program, descriptive information defining the data set must be placed into a data control block for the access routines. Sources of information for the data control block are keyword operands in the DCB macro instruction or, in some cases, the DD statement, data set label, or user's problem program. General information concerning data set definition is contained in the Data Management Services manual (see Preface). Characteristics of data sets supplied by the DCB macro instruction are described in the Data Management Macro-Instructions manual (see Preface).

The specific information that must be supplied depends upon the data set organization and access method. The following access methods are used to process the assembler data sets:

Access Method	Data Sets
QSAM (Queued Sequential)	SYSPRINT, SYSPUNCH, SYSGO, SYSIN
BSAM (Basic Sequential)	SYSUT1, SYSUT2, SYSUT3
BPAM (Basic Partitioned)	SYSLIB

Table 1 summarizes the assembler capabilities and restrictions on record length

● Table 1. Data Set Characteristics

	SYSIN	SYSLIB	SYSPRINT	SYSPUNCH	SYSGO	SYSUT1 SYSUT2 SYSUT3
LRECL	Fixed at 80	Fixed at 80	Fixed at 121	Fixed at 80	Fixed at 80	N/A
RECFM ①	User must specify in LABEL or DD card F, FS, FBS, FB, FBST, FBT	User must specify in LABEL or DD card F, FS, FBS, FB, FBST, FBT	Set by assembler: if BLKSIZE=LRECL, RECFM=FSM; if BLKSIZE>LRECL, RECFM=FBSM	Set by assembler: if BLKSIZE=LRECL, RECFM=FS; if BLKSIZE>LRECL, RECFM=FBS	Set by assembler: if BLKSIZE=LRECL, RECFM=FS; if BLKSIZE>LRECL, RECFM=FBS	Fixed for U
BLKSIZE ②	User must specify in LABEL or DD card, must be a multiple of LRECL	User must specify in LABEL or DD card, must be a multiple of LRECL	Optional, but must be a multiple of LRECL; if omitted BLKSIZE=LRECL	Optional, but must be a multiple of LRECL; if omitted BLKSIZE=LRECL	Optional, but must be a multiple of LRECL; if omitted BLKSIZE=LRECL	User can not specify; maximum of 4000, minimum of 1739
BUFNO	Optional; if omitted 2 is used	Set by assembler to 1	Optional; if omitted 2 is used	Optional; if omitted 3 is used for unit record and 1 for other devices	Optional; if omitted 3 is used for unit record and 1 for other devices	User can not specify; either 1 or 2
For 44K availability	BLKSIZE times BUFNO can not be greater than 3600	BLKSIZE can not be greater than 3600	BLKSIZE times BUFNO can not be greater than 1210	BLKSIZE times BUFNO can not be greater than 400	BLKSIZE times BUFNO can not be greater than 400	
For calculating core requirements	L_1 = BLKSIZE times BUFNO	L_2 = BLKSIZE	L_3 = BLKSIZE times BUFNO	L_4 = BLKSIZE times BUFNO	L_5 = BLKSIZE times BUFNO	

③   Minimum core required for the assembler = $\begin{cases} L_1 + L_2 + 37000 \\ L_3 + L_4 + L_5 + 37000 \end{cases}$

                   whichever is greater

③   Maximum core that the assembler can effectively use = $L_4 + L_5 + 535,000$

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

②   Blocking is not allowed on unit record devices. Blocking on other direct access can not be greater than the track size unless T is specified on RECFM

③   For MVT environment add 5,000 for core required

and format, and blocksize and buffering
facilities available to the user. The
values shown in Table 1 are based upon the
minimum core requirements of Assembler F
(44K), which will allow a symbol table
length of approximately 7000 bytes. If
more than 44K is available, the block sizes
and buffer numbers can be increased. How-
ever, if the user specifies a combination
of blocking and buffering which does not
leave room for the symbol table, abnormal
termination of the task may occur (ABEND
804) when the assembler attempts to issue
a GETMAIN macro instruction.

In addition to the data set character-
istics shown in Table 1, the following
options are available to the user (refer to
the Supervisor and Data Management Macro-
Instructions publication). Options not
shown below are fixed by the assembler and
cannot be specified.

Data Sets	Options
SYSIN, SYSPUNCH, SYSPRINT, SYSGO	DEVD (device type) BFALN (buffer boundary alignment) BUFL (buffer length) EROPT (error option)
SYSUT1, 2, 3	DEVD (device type) OPTCD (optional ser- vice for validity checking and chained scheduling)

RETURN CODES

Table 2 shows the return codes issued by
the assembler for use with the COND= para-
meter of JOB or EXEC statements. The
COND= parameter is explained in the Job
Control Language publication.

The return code issued by the assembler
is the highest severity code that is:

1. Associated with any error detected
   by the assembler (see Appendix A for
   diagnostic messages and severity
   codes).
2. Associated with MNOTE messages pro-
   duced by macro instructions.
3. Associated with an unrecoverable I/O
   error occurring during the assembly.

If a permanent I/O error occurs on any of
the assembler files or a DD card for a
required data set is missing, a message is
printed on the operator's console and an
ABEND with a user ABEND code of 20 is given
by the assembler. This terminates the
assembly.

Table 2. Return Codes

Return Code	Explanation
0	No errors detected
4	Minor errors detected; successful program execution is probable
8	Errors detected; unsuccessful program execution is possible
12	Serious errors detected; unsuccessful program execution is probable
16	Critical errors detected; normal execution is impossible
20	Unrecoverable I/O error occurred during assembly or missing data sets; assembly terminated

CATALOGED PROCEDURES

This section describes three IBM-provided cataloged procedures: a procedure for assembling (ASMFC), a procedure for assembling and linkage editing (ASMFCL), and a procedure for assembling, linkage editing, and executing (ASMFCLG). The procedures rely on conventions regarding the naming of device classes. These conventions, shown in Table 3, must be incorporated into the system at system generation time.

Table 3. Device Naming Conventions

Device Classname	Devices Assigned
SYSSQ	Any devices allowing sequential access to records for reading and writing
SYSDA	Direct-access devices
SYSCP	Card punches

To use cataloged procedures, EXEC statements naming the desired procedures are placed in the input stream following the JOB statement. Subsequently, the specified cataloged procedure is brought from a procedure library and merged into the input stream.

The System Programmer's Guide discusses the placing of procedures in the procedure library.

CATALOGED PROCEDURE FOR ASSEMBLY (ASMFC)

This procedure requests the operating system to load and execute the assembler. The name ASMFC must be used to call this procedure. The result of execution is an object module, in punched card form, and an assembler listing.

In the following example, input enters via the input stream. The statements entered in the input stream to use this procedure are:

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

The statements of the ASMFC procedure are brought from the procedure library and merged into the input stream.

Figure 1 shows the statements that make up the ASMFC procedure.

```
1
 //ASM        EXEC  PGM=IEUASM,REGION=50K

2
 //SYSLIB     DD    DSNAME=SYS1.MACLIB,DISP=SHR

3
 //SYSUT1     DD    UNIT=SYSSQ,SPACE=(1700, (400,50))

4
 //SYSUT2     DD    UNIT=SYSSQ,SPACE=(1700, (400,50))

5
 //SYSUT3     DD    UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),         X
 //                 SPACE=(1700,(400,50))

6
 //SYSPRINT   DD    SYSOUT=A

7
 //SYSPUNCH   DD    SYSOUT=B
          ----------
          ----------
```

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

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

3 4 5 These statements specify the assembler utility data sets. The device classname used here, SYSSQ, may represent a collection of tape drives, or direct-access units, or both. The I/O units assigned to this name are specified by the installation when the system is generated. A unit name, e.g., 2311 may be substituted for SYSSQ.

The SEP= subparameter in statement 5 and the SPACE= parameter in statements 3, 4, and 5 are effective only if the device assigned is a direct-access device: otherwise they are ignored. The space required is dependent on the make-up of the source program. The Job Control Language publication explains space allocation.

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

7 This statement describes the data set that will contain the object module produced by the assembler.

Figure 1. Cataloged Procedure for Assembly (ASMFC)

CATALOGED PROCEDURE FOR ASSEMBLY AND LINKAGE--EDITING (ASMFCL)

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

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

The following example assumes input to the assembler via the input job stream. It also makes provision in the //LKED job step for concatenating the input to the linkage editor from the //ASM job step with any additional linkage editor input in the input job stream. This additional input can be a previously produced object module which is to be linked to the object module produced by job step //ASM.

An example of the statements entered in the input stream to use this procedure is:

The procedure is brought from the procedure library and merged into the input stream.

Figure 2 shows the statements that make up the ASMFCL procedure. Only those statements not previously discussed are explained.

```
//ASM       EXEC  PGM=IEUASM,PARM=LOAD,REGION=50K

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

  //SYSUT1    DD    UNIT=SYSSQ,SPACE=(1700,(400,50))

  //SYSUT2    DD    UNIT=SYSSQ,SPACE=(1700,(400,50))

  //SYSUT3    DD    UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),        X
  //                SPACE=(1700,(400,50))

  //SYSPRINT  DD    SYSOUT=A

  //SYSPUNCH  DD    SYSOUT=B
1
  //SYSGO     DD    DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(100,50)),  X
  //                DISP=(MOD,PASS)
2
  //LKED      EXEC  PGM=IEWL,PARM=(XREF,LIST,NCAL),REGION=96K,       X
  //                COND=(8,LT,ASM)
3
4 //SYSLIN    DD    DSNAME=&LOADSET,DISP=(OLD,DELETE)
  //          DD    DDNAME=SYSIN
5
  //SYSLMOD   DD    DSNAME=&TEMP(PDS).UNIT=SYSDA,SPACE=(1024,(50,20,1)),  X
  //                DISP=(MOD,PASS)
6
  //SYSUT1    DD    UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))
7
  //SYSPRINT  DD    SYSOUT=A,DCB=(,BLKSIZE=121)
      ------------
```

[1] In this procedure the SYSGO DD statement describes a temporary data set -- the object module -- which is to be passed to the linkage editor.

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

[3] This statement identifies the linkage editor input data set as the same one produced as output by the assembler.

[4] This statement is used to concatenate any input to the linkage editor from the input stream with the input from the assembler.

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

[6] This statement specifies the utility data set for the linkage editor.

[7] This statement identifies the standard output class as the destination for the linkage editor listing.

Figure 2. Cataloged Procedure for Assembling and Linkage Editing (ASMFCL)

●Cataloged Procedures   7

CATALOGED PROCEDURE FOR ASSEMBLY,
LINKAGE-EDITING, AND EXECUTION
(ASMFCLG)

This procedure consists of three job
steps: assembling, linkage editing, and
executing.

Figure 3 shows the statements that make
up the ASMFCLG procedure. Only those
statements not previously discussed are
explained in the figure.

The name ASMFCLG must be used to call
this procedure. Assembler and linkage
editor listings are produced.

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

```
//jobname      JOB
//stepname     EXEC PROC=ASMFCLG
//ASM.SYSIN    DD    *
                 |
               source program statements
                 |
/*               |
//LKED.SYSIN   DD    *        ]
                 |             }  necessary only if linkage
               object module or   editor is to combine modules
               linkage editor     or read linkage editor control
               control statements  information from the job stream
/*               |
//GO.ddname    DD    (parameters)  ]
//GO.ddname    DD    (parameters)  }
//GO.ddname    DD    *             ]  only if
                 |                    necessary
               problem program input
/*               |
```

```
//ASM        EXEC   PGM=IEUASM,PARM=LOAD,REGION=50K

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

//SYSUT1     DD     UNIT=SYSSQ,SPACE=(1700,(400,50))

//SYSUT2     DD     UNIT=SYSSQ,SPACE=(1700,(400,50))

//SYSUT3     DD     UNIT=(SYSSQ,SEP=(SYSUT2,SYSUT1,SYSLIB)),         X
//                  SPACE=(1700,(400,50))

//SYSPRINT   DD     SYSOUT = A

//SYSPUNCH   DD     SYSOUT=B

//SYSGO      DD     DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(80,(100,50)),  X
//                  DISP=(MOD,PASS)

1 //LKED     EXEC   PGM=IEWL.PARM=(XREF,LET,LIST,NCAL),REGION=96K,   X
//                  COND=(8,LT,ASM)

//SYSLIN     DD     DSNAME=&LOADSET,DISP=(OLD,DELETE)
//           DD     DDNAME=SYSIN

2 //SYSLMOD   DD     DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)), X
//                  DISP=(MOD,PASS)

//SYSUT1     DD     UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),SPACE=(1024,(50,20))

//SYSPRINT   DD     SYSOUT=A,DCB=(,BLKSIZE=121)

3 //GO       EXEC   PGM=*.LKED.SYSLMOD,COND=((8,LT,ASM) , (4,LT,LKED))
           ----------
           ----------
```

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

2 The output of the linkage editor is specified as a member of a temporary data set, residing on a direct-access device, and is to be passed to a succeeding job step.

3 This statement initiates execution of the assembled and linkage edited program. The notation *.LKED.SYSLMOD identifies the program to be executed as being in the data set described in job step LKED by the DD statement named SYSLMOD. When running with MVT (Option 4) the REGION parameter can be calculated with the help of the Storage Estimates publication (see preface).

Figure 3. Cataloged Procedure for Assembly, Linkage Editing and Execution (ASMFCLG)

OVERRIDING STATEMENTS IN CATALOGED PROCEDURES

Any parameter in a cataloged procedure can be overriden except the PGM= parameter in the EXEC statement. Such overriding of statements or fields is effective only for the duration of the job step in which the statements appear. The statements, as stored in the procedure library of the system, remain unchanged.

Overriding for the purposes of re-specification, addition, or nullification is accomplished by including in the input stream statements containing the desired changes and identifying the statements to be overriden.

EXEC Statements

The PARM= and COND= parameters can be added or, if present, re-specified by including in the EXEC statement calling the procedure the notation PARM.stepname=, or COND.stepname=, followed by the desired parameters. "Stepname" identifies the EXEC statement within the procedure to which the modification applies. Overriding the PGM= parameter is not possible.

If the procedure consists of more than one job step, a PARM.stepname= or COND. stepname= parameter may be entered for each step. The entries must be in order, i.e., PARM.step1=, PARM.step2=, etc.

DD Statements

All parameters in the operand field of DD statements may be overridden by including in the input stream (following the EXEC card calling the procedure) a DD statement with the notation //stepname.ddname in the name field. "Stepname" refers to the job step in which the statement identified by "ddname" appears.

Examples

In the assembly procedure ASMFC (Figure 1), the production of a punched object deck could be suppressed and the UNIT= and SPACE= parameters of data set SYSUT1 re-specified, by including the following statements in the input stream:

```
//stepname       EXEC    PROC=ASMFC,                 X
//                       PARM.ASM=NODECK

//ASM.SYSUT1     DD      UNIT=2311,                   X
//                       SPACE=(200,(300,40))

//ASM.SYSIN      DD      *
```

In procedure ASMFCLG (Figure 3), suppressing production of an assembler listing and adding the COND= parameter to the EXEC statement, which specifies execution of the linkage editor, may be desired. In this case, the EXEC statement in the input stream would appear as follows:

```
//stepname       EXEC    PROC=ASMFCLG,               X
//                       PARM.ASM=(NOLIST,LOAD),      X
//                       COND.LKED=(8,LT,stepname.ASM)
```

NOTE: Overriding the LIST parameter effectively deletes the PARM=LOAD so this must be repeated in the override statement.

For current execution of procedure ASMFCLG, no assembler listing would be produced, and execution of the linkage editor job step //LKED would be suppressed if the return code issued by the assembler (step ASM) was greater than 8. Using the procedure ASMFCL (Figure 2) to

1. Read input from a non-labeled 9-track tape on unit 282 that has a standard blocking factor of 10;
2. Put the output listing on a labeled tape VOLID=TAPE10, with a data set name of PROG1 and a blocking factor of 5;
3. Block the SYSGO output of the assembler and use it as input to the linkage editor with a blocking factor of 5;
4. Link edit the module only if there are no errors in the assembler, i.e., COND=0;
5. Link edit on to a previously allocated and cataloged data set USER.LIBRARY with a member name of PROG, the input stream appears as follows:

```
//jobname        JOB
//stepname       EXEC    PROC=ASMFCL,                X
//                       COND.LKED=(0,NE,stepname.ASM)
//ASM.SYSPRINT   DD      DSNAME=PROG1,UNIT=TAPE,      X
//                       VOLUME=SER=TAPE10,DCB=(BLKSIZE=605)
//ASM.SYSGO      DD      DCB=(BLKSIZE=400)
//ASM.SYSIN      DD      UNIT=282,LABEL=(,NL),        X
//                       DCB=(RECFM=FSB,BLKSIZE=800)
//LKED.SYSIN     DD      DCB=stepname.ASM.SYSGO
//LKED.SYSLMOD   DD      DSNAME=USER.LIBRARY(PROG),DISP=OLD
/*
```

NOTE: The order of appearance of ddnames within job steps ASM and LKED has been preserved. Thus, SYSPRINT precedes SYSGO within step ASM. The ddname ASM.SYSIN was placed last since SYSIN does not occur at

all within step ASM. These points are covered in the section <u>Using Cataloged Procedures</u> in the <u>Job Control Language</u> manual.

To assemble two programs, link edit the two assemblies into one load module and execute the load module, entering at PROG, the input stream appears as follows:

```
//stepname1    EXEC    PROG=ASMFC,PARM.ASM='LOAD'
//ASM.GO       DD      DSNAME=&LOADSET,UNIT=SYSSQ,        X
//                     SPACE=(80,(100,50)),               X
//                     DISP=(MOD,PASS),DCB=(BLKSIZE=400)
//ASM.SYSIN    DD      *
                       ǀ
                       ǀ
                       ǀ
                       source program 1 statements
                       ǀ
                       ǀ
/*                     ǀ
//stepname2    EXEC    PROC=ASMFCLG
//ASM.SYSGO    DD      DCB=(BLKSIZE=400)
//ASM.SYSIN    DD      *
                       ǀ
                       ǀ
                       ǀ
                       source program 2 statements
                       ǀ
                       ǀ
                       ǀ
/*                     ǀ
//LKED.SYSLIN  DD      DCB=(RECFM=FB,BLKSIZE=400)
//LKED.SYSIN   DD      *
               ENTRY   PROG
/*
//GO.ddname            dd cards for GO step
```

The overriding step with ddname =LKED.SYSLIN is necessary whenever output from an assembly and output from at least one or more processors (including the Assembler) is placed on SYSLIN. The DCB=(RECFM=FB,...) allows the linkage editor to process <u>all</u> the blocked input. Otherwise it will stop reading SYSLIN whenever it encounters a partial block. Such a situation arises if, say, the first assembly produces 22 cards. These 22 cards result in 4 full blocks of 400 bytes (5 times 80) plus 1 partial block of 160 bytes (2 times 80).

The <u>Job Control Language</u> and <u>System Programmer's Guide</u> publications provide additional description of overriding techniques.

●Cataloged Procedures   9.1

The assembler listing (Figure 4) consists of five sections, ordered as follows: external symbol dictionary items, the source and object program statements, relocation dictionary items, symbol cross reference table, and diagnostic messages. In addition, three statistical messages may appear in the listing:

1. After the diagnostics, a statements-flagged message indicates the total number of statements in error. It appears as follows: nnn STATEMENTS FLAGGED IN THIS ASSEMBLY.

2. After the statements-flagged message, the assembler prints the highest severity code encountered (if non-zero). This is equal to the assembler return code. The message appears as follows: nn WAS HIGHEST SEVERITY CODE.

3. After the severity code, the assembler prints a count of lines printed, which appears as follows: nnn PRINTED LINES. This is a count of the actual number of 121-byte records generated by the assembler; it may be less than the total number of printed and blank lines appearing on the listing if the SPACE n assembler instruction is used. For a SPACE n that does not cause an eject, the assembler inserts n blank lines in the listing by generating n/3 blank 121-byte records -- rounded to the next lower integer if a fraction results; e.g., for a SPACE 2, no blank records are generated. The assembler does not generate a blank record to force a page eject.

In addition to the above items, the assembler prints the deck identification and current date on every page of the listing. If the timer is available, the assembler prints the time of day to the left of the date on page 1 of the ESD listing. This is the time when printing starts, rather than the start of the assembly, and is intended only to provide unique identification for assemblies made on the same day. The time is printed as hh.mm, where hh is the hour of the day (midnight beginning at 00), and mm is the number of minutes past the hour.

EXTERNAL SYMBOL DICTIONARY (ESD)

This section of the listing contains the external symbol dictionary information passed to the linkage-editor in the object module. The entries describe the control sections, external references, and entry points in the assembled program. There are six types of entries, shown in Table 4, along with their associated fields. The circled numbers refer to the corresponding heading in the sample listing (Figure 4). The X's indicate entries accompanying each type designation.

Table 4. Types of ESD Entries

① SYMBOL	② TYPE	③ ID	④ ADDR	⑤ LENGTH	⑥ LD ID
X	SD	X	X	X	-
X	LD	-	X	-	X
X	ER	X	-	-	-
-	PC	X	X	X	-
-	CM	X	X	X	-
X	XD	X	X	X	-

1. This column contains the name of every external dummy section, control section, entry point, and external symbol.

2. This column contains the type designator for the entry, as shown in the table. The type designators are defined as:

   SD--Names section definition. The symbol appeared in the name field of a CSECT or START statement.
   LD--The symbol appeared as the operand of the ENTRY statement.
   ER--External reference. The symbol appeared as the operand of an EXTRN statement, or was defined as a V-type address constant.
   PC--Unnamed control section definition.
   CM--Common control section definition.
   XD--External dummy section.

3. This column contains the external symbol dictionary identification number (ESDID). The number is a unique two-digit hexadecimal number identifying the entry. It is used by the LD entry of the ESD and by the relocation dictionary for cross-referencing the ESD.

4. This column contains the address of the symbol (hexadecimal notation) for SD- and LD-type entries, and zeros for ER-type entries. For PC- and CM-type entries, it indicates the beginning address of the control section. For

EXTERNAL SYMBOL DICTIONARY

EXAM SYMBOL ①	TYPE ②	ID ③	ADDR ④	LENGTH ⑤	LD ID ⑥
SAMPLR	SD	01	000000	000388	

EXAM ⑦   SAMPLE PROGRAM ⑧   Page 3 ⑨

LOC ⑩	OBJECT CODE ⑪	ADDR1 ⑫	ADDR2 ⑬	STMT ⑭	SOURCE STATEMENT	F 14FEB66 ⑮ 4/11/66 ⑯
000000	47F0 F00A		0000A	59+BEGIN	B 10(0,15) BRANCH AROUND ID	
000004	05			60+	DC AL1(5)	
000005	C2C5C7C9D5			61+	DC CL5'BEGIN' IDENTIFIER	⑰
00000A	90EC D00C		0000C	62+	STM 14,12,12(13) SAVE REGISTERS	
00000E	05C0			63	BALR R12,0 ESTABLISH ADDRESSABILITY OF PROGRAM	SAMPL057
000010				64	USING *,R12 AND TELL THE ASSEMBLER WHAT BASE TO USE	SAMPL058

EXAM ⑦   RELOCATION DICTIONARY   Page 1 ⑨

POS.ID ⑱	REL.ID ⑲	FLAGS ⑳	ADDRESS ㉑	4/11/66 ⑯
01	01	0C	0001FC	
01	01	0C	00020C	
01	01	0C	00021C	
01	01	0C	0002D4	
01	01	0C	000334	

EXAM ⑦   CROSS-REFERENCE   Page 1 ⑨

SYMBOL ㉒	LEN ㉓	VALUE ㉔	DEFN ㉕	REFERENCES ㉖	4/11/66 ⑯
BEGIN	00004	000000	0059	0156 0158 0174 0184 0186 0220	
EXIT	00004	00007E	0096	0111	
HIGHER	00002	0000F4	0130	0125	
IHB0005	00001	00007B	0093	0090	
IHB0005A	00002	00007C	0094	0089	

XAM ⑦   DIAGNOSTICS   Page 1 ⑨

TMT ㉗	ERROR CODE ㉘	MESSAGE ㉙	4/11/66 ⑯
19	IEU025	NEAR OPERAND COLUMN 7--RELOCATABILITY ERROR	
21	IEU035	NEAR OPERAND COLUMN 9--ADDRESSABILITY ERROR	

2   STATEMENTS FLAGGED IN THIS ASSEMBLY
8   WAS HIGHEST SEVERITY CODE
261   PRINTED LINES

ure 4.   Assembler Listing

XD-type entries, it indicates the alignment by printing a number one less than the number of bytes in the unit of alignment, e.g., 7 indicates double word alignment.

5.   This column contains the assembled length, in bytes, of the control section (hexadecimal notation).

6.   This column contains, for LD-type entries, the identification (ID) number assigned to the ESD entry that identifies the control section in which the symbol was defined.

SOURCE AND OBJECT PROGRAM

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

7. This is the four-character deck iden-
    tification. It is the symbol that
    appears in the name field of the first
    TITLE statement. The assembler
    prints the deck identification and
    date (item 16) on every page of the
    listing.
8. This is the information taken from the
    operand field of a TITLE statement.

    NOTE: TITLE, SPACE and EJECT state-
    ments will not appear in the source
    listing unless the statement is con-
    tinued onto another card. Then the
    first card of the statement is printed.
    However, any of these three types of
    statements, if generated as macro in-
    struction expansion, will never be
    listed regardless of continuation.

9. Listing page number. Each section of
    the listing starts with page 1.
10. This column contains the assembled
    address (hexadecimal notation) of the
    object code.
11. This column contains the object code
    produced by the source statement. The
    entries are always left-justified.
    The notation is hexadecimal. Entries
    are machine instructions or assembled
    constants. Machine instructions
    are printed in full with a blank
    inserted after every four digits
    (two bytes). Constants may be only
    partially printed (see the PRINT
    assembler instruction in the As-
    sembler Language publication).
12. These two columns contain effective
    addresses (the result of adding to-
    gether a base register value and dis-
    placement value):

    a. The column headed ADDR1 contains
        the effective address for the
        first operand of an SS in-
        struction.
    b. The column headed ADDR2 contains
        the effective address of the
        second operand of any instruc-
        tion referencing storage.

        Both address fields contain six
    digits; however, if the high-order
    digit is a zero, it is not printed.
13. This column contains the statement
    number. A plus sign (+) to the right
    of the number indicates that the state-
    ment was generated as the result of
    macro instruction processing.

14. This column contains the source pro-
    gram statement. The following items
    apply to this section of the listing:

    a. Source statements are listed,
        including those brought into the
        program by the COPY assembler
        instruction, and including macro
        definitions submitted with the
        main program for assembly.
        Listing control instructions are
        not printed, except for the
        following case: PRINT is listed
        when PRINT ON is in effect and a
        PRINT statement is encountered.
    b. Macro definitions obtained from
        SYSLIB are not listed.
    c. The statements generated as the
        result of a macro instruction
        follow the macro instruction in
        the listing.
    d. Assembler or machine instructions
        in the source program that con-
        tain variable symbols are listed
        twice: as they appear in the
        source input, and with values
        substituted for the variable
        symbols.
    e. Diagnostic messages are not list-
        ed in-line in the source and
        object program section. An error
        indicator, ***ERROR***, follows
        the statement in error. The
        message appears in the diagnostic
        section of the listing.
    f. MNOTE messages are listed in-line
        in the source and object program
        section. An MNOTE indicator
        appears in the diagnostic section
        of the listing for MNOTE state-
        ments other than MNOTE*. The
        MNOTE message format is severity
        code, message text.
    g. The MNOTE* form of the MNOTE
        statement results in an in-line
        message only. An MNOTE indicator
        does not appear in the diagnostic
        section of the listing.
    h. When an error is found in a
        programmer macro definition, it
        is treated the same as any other
        assembly error: the error
        indication appears after the
        statement in error, and a diag-
        nostic is placed in the list of
        diagnostics. However, when
        an error is encountered during the
        expansion of a macro instruction
        (system- or programmer-defined),
        the error indication appears in
        place of the erroneous statement,
        which is not listed. The error
        indication follows the last
        statement listed before the

erroneous statement was encountered, and the associated diagnostic message is placed in the list of diagnostics.

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

j. If the END statement contains an operand, the transfer address appears in the location column (LOC).

k. In the case of COM, CSECT, and DSECT statements, the location field contains the beginning address of these control sections, i.e., the first occurrence.

l. In the case of EXTRN, ENTRY, and DXD instructions, the location field and object code field are blank.

m. For a USING statement, the location field contains the value of the first operand.

n. For LTORG and ORG statements, the location field contains the location assigned to the literal pool or the value of the ORG operand.

o. For an EQU statement, the location field contains the value assigned.

p. Generated statements always print in normal statement format. Because of this, it is possible for a generated statement to occupy three or more continuation lines on the listing. This is unlike source statements, which are restricted to two continuation lines.

15. This column contains the identifier of the assembler (F) and the date when this version was released by Systems Development Division to DPD Program Information Department.

16. Current date (date run is made).

17. Identification-sequence field from the source statement.

RELOCATION DICTIONARY

This section of the listing contains the relocation dictionary information passed to the linkage editor in the object module. The entries describe the address constants in the assembled program that are affected by relocation.

18. This column contains the external symbol dictionary ID number assigned to the ESD entry that describes the control section in which the address constant is used as an operand.

19. This column contains the external symbol dictionary ID number assigned to the ESD entry that describes the control section in which the referenced symbol is defined.

20. The two-digit hexadecimal number in this column is interpreted as follows:

First Digit. A zero indicates that the entry describes an A-type or Q-type address constant. A one indicates that the entry describes a V-type address constant. A three describes a CXD entry.
Second Digit. The first three bits of this digit indicate the length of the constant and whether the base should be added or subtracted:

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

21. This column contains the assembled address of the field where the address constant is stored.

CROSS REFERENCE

This section of the listing information concerns symbols which are defined and used in the program.

22. This column contains the symbols.

23. This column states the length (decimal notation), in bytes, of the field occupied by the symbol value.

24. This column contains either the address the symbol represents, or a value to which the symbol is equated.

25. This column contains the statement number of the statement in which the symbol was defined.

26. This column contains the statement numbers of statements in which the symbol appears as an operand. In the case of a duplicate symbol, the assembler fills this column with the message:

****DUPLICATE****

The following notes apply to the cross-reference section:

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

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

- In the case of an undefined symbol, the assembler fills columns 23, 24, and 25 with the message:

    ****UNDEFINED****.

DIAGNOSTICS

This section contains the diagnostic messages issued as a result of error conditions encountered in the program. The text, severity code, and explanatory notes for each message are contained in Appendix A.

27. This column contains the number of the statement in error.
28. This column contains the message identifier.
29. This column contains the message, and, in most cases, an operand column pointer that indicates the vicinity of the error. In the following example, the approximate location of the addressability error occurred in the 9th column of the operand field:

Example:

STMT	ERROR CODE	MESSAGE
21	IEU035	NEAR OPERAND COLUMN 9 -- ADDRESSABILITY ERROR

The following notes apply to the diagnostic section:

- An MNOTE indicator of the form MNOTE STATEMENT appears in the diagnostic section if an MNOTE statement other than MNOTE* is issued by a macro instruction. The MNOTE statement itself is in-line in the source and object program section of the listing. The operand field of an MNOTE* is printed as a comment, but does not appear in the diagnostic section.

- A message identifier consists of six characters and is of the form:
  IEUxxx
    IEU identifies the issuing agent as Assembler F, and xxx is a unique number assigned to the message.

NOTE: Editing errors in system macro instructions are discovered at the time the macro instruction is read from the library, i.e., after the END statement. To determine the location of these errors it is necessary to punch all system macro instructions, including inner macro instructions, and insert them in the source program as programmer macro instructions. To aid in debugging, it is advisable to run all macro instructions as programmer macro instructions before incorporating them as system macro instructions.

This section consists of a number of discrete subjects about assembler language programming.

SAVING AND RESTORING GENERAL REGISTER CONTENTS

A problem program should save the values contained in the general registers upon commencing execution and, upon completion, restore to the general registers these same values. Thus, as control is passed from the operating system to a problem program and, in turn, to a subprogram, the status of the registers used by each program is preserved. This is done through use of the SAVE and RETURN system macro instructions.

The SAVE macro instruction should be the first statement·in the program. It stores the contents of registers 14, 15, and 0 through 12 in an area provided by the program that passes control. When a problem program is given control, register 13 points to an area in which the general register contents should be saved.

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

At completion, the problem program restores the contents of general registers 14, 15 and 0-12 by use of the RETURN system macro instruction (which also indicates program completion). The content of register 13 must be restored before execution of the RETURN macro instruction.

The coding sequence that follows illustrates the basic process of saving and restoring the registers. A complete discussion of the SAVE and RETURN macro instructions and the saving and restoring of registers is contained in the Data Management Services and Data Management Macro-Instructions publications (see Preface).

Name	Operation	Operand
BEGIN	SAVE	(14,12)
	.	
	.	set up base register
	.	
	ST	13,SAVEBLK+4
	LA	13,SAVEBLK
	.	
	.	
	L	13,SAVEBLK+4
	RETURN	(14,12)
SAVEBLK	DC	18F'0'

PROGRAM TERMINATION

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

In addition to indicating program completion and restoring registers, the RETURN macro instruction may also pass a return code -- a condition indicator that may be used by the program receiving control. If the return is to the operating system, the return code is compared against the condition stated in the COND= parameter of the JOB or EXEC statements. If return is to another problem program, the return code is available in general register 15, and may be used as desired. Register 13 should be restored before issuing the RETURN macro instruction.

The RETURN system macro instruction is discussed in detail in the Supervisor and Data Management Macro-Instructions publication.

PARM FIELD ACCESS

Access to information in the PARM field of an EXEC statement is gained through general register 1. When control is given to the problem program, general register 1 contains the address of a full word which, in turn, contains the address of the data area containing the information.

The data area consists of a half word containing the count (in binary) of the number of information characters, followed by the information field. The information field is aligned to a full-word boundary. The following diagram illustrates this process.
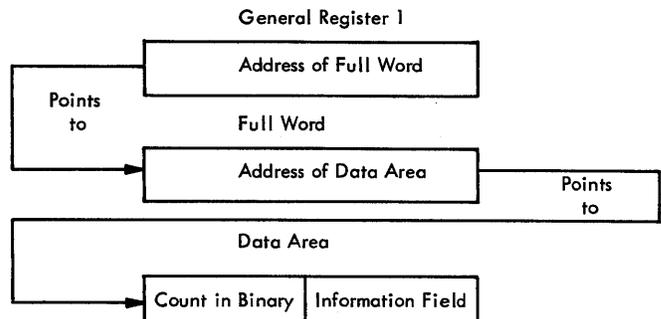
MACRO DEFINITION LIBRARY ADDITIONS

Source statement coding, to be retrieved by the COPY assembler instruction, and

macro definitions may be added to the macro library. The IEBUPDAT utility program is used for this purpose. Details of this program and its control statements are contained in the Utilities publication. The following sequence of job control statements can be used to call the utility program and identify the needed data sets. It is assumed that the job control statements, IEBUPDAT program control statements, and data are to enter the system via the input stream.

```
//jobname    JOB
//stepname   EXEC  PGM=IEBUPDAT,PARM=NEW
//SYSUT2     DD    DSNAME=SYS1.MACLIB,DISP=OLD
//SYSPRINT   DD    SYSOUT=A
//SYSIN      DD    *
             .
             .
             .
IEBUPDAT control statements and source statements or
macro-definitions to be added to the macro-library
(SYS1.MACLIB)
             .
             .
             .
/*  (delimiter statement)
```

LOAD MODULE MODIFICATION - ENTRY POINT RESTATEMENT

If the editing functions of the linkage editor are to be used to modify a load module, the entry point to the load module must be restated when the load module is reprocessed by the linkage editor. Otherwise, the first byte of the first control section processed by the linkage editor will become the entry point. To enable restatement of the original entry point, or designation of a new entry point, the entry point must have been identified originally as an external symbol, i.e., appeared as an entry in the external symbol dictionary. External symbol identification is done automatically by the assembler if the entry point is the name of a control section or START statement; otherwise, an assembler ENTRY statement must be used to identify the entry point name as an external symbol.

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

● By placing the entry point symbol in the operand field of an EXTRN statement and an END statement in the new object module.

● By using an END statement in the new object module to designate a new entry point in the new object module.

● By using a linkage editor ENTRY statement to designate either the original entry point or a new entry point for the load module.

Further discussion of load module entry points is contained in the Linkage Editor publication.

OBJECT MODULE LINKAGE

Object modules, whether Assembler-, FORTRAN-, or COBOL-generated, may be combined by the linkage editor to produce a composite load module, provided each object module conforms to the data formats and linkage conventions required. This topic discusses the use of the CALL system macro instruction to link an assembler language "main" program to subprograms produced by FORTRAN and COBOL. The Supervisor and Data Management Macro-Instructions publication contains additional details concerning linkage conventions and the CALL system macro instruction.

Figure 5 shows the statements used to establish the assembler program linkage to the called subprograms.

If any input/output operations are performed by called subprograms, appropriate DD statements for the data sets used by the subprograms must be supplied. See the FORTRAN IV (E) Programmer's Guide publication for explanation of the DD statements used to describe data sets for FORTRAN programs and a description of the special FORTRAN data set record formats. The COBOL (E) Programmer's Guide publication provides DD statement information for COBOL programs.

DICTIONARY SIZE AND SOURCE STATEMENT COMPLEXITY

This section describes the composition of the assembler dictionaries and their entry sizes, and describes methods for determining if the limits on source statement complexity will be exceeded.

Dictionary entries, e.g., sequence symbol names, prototype symbolic parameters, vary in length. Therefore, the number of entries a dictionary can hold is determined by the types of entries.

Source statement complexity -- the number of symbols, characters, operators, delimiters, references to length attributes, self-defining terms, literals, and expressions appearing in a source statement -- determines whether or not the source statement can be successfully processed.

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

[1] This is an example of OS linkage convention. See the publication Supervisor and Data Management Services for details.

[2] The symbol used for "name" in this statement is:

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

    b. The name defined by the following COBOL statements in the procedure division:

        ENTER LINKAGE. ENTRY'name'.

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

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

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

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

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

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

▶Figure 5.   Linkage Statements

DICTIONARIES USED IN CONDITIONAL ASSEMBLY AND MACRO INSTRUCTION EXPANSION

To accomplish macro instruction expansion and conditional assembly, the assembler constructs a general dictionary consisting of two parts: one global dictionary for the entire program, and an area for all of the local dictionaries.

The global dictionary contains one entry for each machine operation code, extended mnemonic operation code, assembler operation code, macro instruction, and global SET variable symbol.

The local dictionary area consists of one local dictionary for each different macro definition in the program, and one local dictionary for the main portion of the program (those statements not within a macro definition, also called "open code."). The contents of the local dictionaries are described in subsequent paragraphs.

The capacity of the general dictionary (global dictionary and all local dictionaries) is up to 64 blocks of 1024 bytes each. The division of the dictionary into global and local sections is done dynamically: as the global dictionary becomes larger, it occupies blocks taken from the local dictionary area. Thus, the global dictionary is always core resident. As it

expands into the local dictionary area, the local dictionaries may overflow onto a utility file. The size of the dictionaries in core depends upon core availability. The minimum core allocation is three blocks for the global dictionary and two blocks for each local dictionary.

Each block in the global and local dictionaries contains complete entries. Any entry not fitting into a block is placed in the next block; the remaining bytes in the current block are not used.

The global and local dictionaries take two forms: one when the dictionary entries are collected, i.e., picked up during the initial scan of the source program, and one during the actual conditional assembly and macro generation, i.e., generation time. The following text describes the global and local dictionaries at both collection time and generation time.

Global Dictionary at Collection Time

One global dictionary is built for the entire program. It contains machine operation codes, extended mnemonic operation codes, assembler operation codes, macro instruction mnemonics, and global SET variable symbols. One entry is made for each. The size of each type of entry is shown in Table 5.

Table 5.   Global Dictionary Entries at
           Collection Time

Entry	Size
Each machine operation code**	5 bytes plus mnemonic*
Each extended mnemonic operation code or assembler operation**	6 bytes plus mnemonic*
Each macro mnemonic operation code	10 bytes plus mnemonic*
Each global SET variable symbol	6 bytes plus name*

*One byte is used for each character in the name or mnemonic.

**For the first two types of entries, a total of $06FE_{16}$ (1790_{10}) bytes of core is required.

Fixed overhead for this dictionary is
   8 bytes for the first block
   4 bytes for each succeeding block
   5 bytes for the last block.

Local Dictionaries at Collection Time

For the main portion of the program (those statements not within a macro definition), one local dictionary is constructed in which ordinary symbols, sequence symbols, and local SET variable symbols are entered. In addition, one local dictionary is constructed for each different macro definition in the program. These local dictionaries contain one entry for each local SET variable symbol, sequence symbol, and prototype symbolic parameter declared within the macro definition. If a sequence symbol is defined before it is referenced, an extra entry for the symbol is made. Table 6 shows the size of each type of entry.

Table 6.   Local Dictionary Entries at
           Collection Time

Entry	Size
Each sequence symbol	10 bytes plus name*
Each local SET variable symbol	6 bytes plus name*
Each prototype symbolic parameter	5 bytes plus name*
Each ordinary symbol appearing in the main portion of the program.	10 bytes plus name*

*One byte is used for each character in the name or mnemonic.

Fixed overhead for this dictionary is
   8 bytes for the first block (if in the main program)
  32 bytes for the first block (if in a macro definition)
   4 bytes for each succeeding block
   5 bytes for the last block

Global Dictionary at Generation Time

The sizes of the global dictionary entries at generation time are shown in Table 7.

18

Table 7. Global Dictionary Entries at
Generation Time

Entry	Size
Each macro mnemonic operation code	3 bytes
Each global SETA symbol (dimensioned)	1 byte plus 4N*
Each global SETA symbol (undimensioned)	4 bytes
Each global SETB symbol (dimensioned)	1 byte plus (N/8)* (N/8 is rounded to the next highest integer)
Each global SETB symbol (undimensioned)	1 bit
Each global SETC symbol (dimensioned)	1 byte plus 9N*
Each global SETC symbol (undimensioned)	9 bytes

*N = dimension

Fixed overhead for this dictionary is
4 bytes plus word alignment.

Local Dictionaries at Generation Time

Table 8 shows the sizes of the various
entries appearing in the local dictionaries
at generation time.

Table 8. Local Dictionary Entries at
Generation Time.

Entry	Size
Each sequence symbol	5 bytes
Each local SETA symbol (dimensioned)	1 byte plus 4N*
Each local SETA symbol (undimensioned)	4 bytes
Each local SETB symbol (dimensioned)	1 byte plus (N/8)* (N/8 is rounded to the next highest integer)
Each local SETB symbol (undimensioned)	1 bit
Each local SETC symbol (dimensioned)	1 byte plus 9N*
Each local SETC symbol (undimensioned)	9 bytes
Each ordinary symbol appearing in the main portion of the program.**	5 bytes

*N=dimension
**These entries appear only in the main
 program local dictionary.

Fixed overhead for this dictionary is
20 bytes plus word alignment.

Additional Dictionary Requirements

The generation time global dictionary and
the generation time local dictionary for
the main portion of the program must be
resident in main storage.
  In addition, if the program contains any
macro instructions, main storage is re-
quired for the largest local dictionary of
the macro definitions being processed.
Furthermore, during processing of macro
definitions containing inner macro instruc-
tions, main storage is required for the
generation time local dictionaries for the
inner macro instructions contained within
the macro definition.
  In addition to those requirements speci-
fied for the local dictionary of the main
portion of the program, each macro defini-
tion local dictionary requires space for
entries shown in Table 9.

Table 9. Macro Definition Local
Dictionary Parameter Table

Entry	Size
Each character string (1)	3 bytes plus L
Each hexadecimal, binary, decimal, and character self-defining term (2)	7 bytes plus L
Each symbol (3)	9 bytes plus L
Each sublist	9 bytes plus 3N bytes plus Y

L = Length of BCD entry in bytes
N = Number of entries in sublist
Y = $E_1 + E_2 + E_3 + \ldots E_n$
    where E = size of an entry (formats 1,2, and 3 above)

Fixed overhead for the macro definition
local dictionary parameter table is 22
bytes. Each nested macro instruction also
requires space in its local dictionary for
the following:

Parameter pointer list	8 bytes plus 2N (N = the number of operands)
Pointers to parameter pointer list and parameter table	8 bytes plus word alignment

SOURCE STATEMENT COMPLEXITY

The complexity of a source statement is
limited both by the macro generator and the
assembler portions of the assembler. The

following topics provide the information
necessary to determine if statement-
complexity limitations for either portion
of the assembler are being exceeded.

Macro Generation and Conditional Assembly Limitation

For any statement which

1. Is a conditional assembly statement,
2. Is a DC or DS statement,
3. Is an EXTRN statement,
4. Contains a sequence symbol or a
   variable symbol,
5. Is not a macro instruction or proto-
   type statement,

the total number of explicit occurrences of

1. Ordinary symbols (includes machine
   mnemonics, assembler mnemonics, con-
   ditional assembly mnemonics, and macro
   instruction mnemonics),
2. Variable symbols,
3. Sequence symbols,

must not exceed 50 for the entire state-
ment.
    For macro instructions and prototype
statements the number of occurrences of

ordinary symbols, variable symbols, and
sequence symbols must not exceed 50 in the
name and operation fields combined; or in
each operand unless the operand is a sub-
list, in which case the limit is applied
to each sublist operand. In any operand if
a character string has the same form as a
symbol, it is counted as a symbol.

Examples of Counts:

&B2 SETB (T'NAME EQ 'W') count=3 (&B2,SETB,NAME)

EXTRN A,B,C,&C   count=5 (EXTRN,A,B,C,&C)

Assembler Portion Limitations

1. Generated statements may not exceed 236
   characters. Statement length includes
   name, operation, operand, and comments.
   If a comments field exists, the blank
   separating the operand and comments
   field is included in the statement
   length. The statement is truncated if
   it exceeds 236 characters.
2. DC, DS, DXD, and literal DCs cannot
   contain more than 32 operands per
   statement.

Code	Message	Explanation	Severity Code
IEU001	DUPLICATION FACTOR ERROR	A duplication factor is not an absolute expression, or is zero in a literal: * in duplication factor expression; invalid syntax in expression.	12
IEU002	RELOCATABLE DUPLI-CATION FACTOR	A relocatable expression has been used to specify the duplication factor.	12
IEU003	LENGTH ERROR	The length specification is out of permissible range or specified invalidly; * in length expression; invalid syntax in expression; no left-parenthesis delimiter for expression.	12
IEU004	RELOCATABLE LENGTH	A relocatable expression has been used to specify length.	12
IEU005	S-TYPE CONSTANT IN LITERAL	Self-explanatory.	8
IEU006	INVALID ORIGIN	The location counter has been reset to a value less than the starting address of the control section; ORG operand is not a simply relocatable expression or specifies an address outside the control section.	12
IEU007	LOCATION COUNTER ERROR	The location counter has exceeded $2^{24}-1$, or passed out of control section in negative direction (3 byte arithmetic).	12
IEU008	INVALID DISPLACEMENT	The displacement in an explicit address is not an absolute value within the range of 0 to 4095.	8
IEU009	MISSING OPERAND	Self-explanatory	12
IEU010	INCORRECT REGISTER SPECIFICATION	The value specifying the register is not an absolute value within the range 0-15, an odd register is specified where an even register is required, or a register was used where none can be specified.	8
IEU011	SCALE MODIFIER ERROR	The scale modifier is not an absolute express-ion or is too large, negative scale modifier for floating point, * in scale modifier expression; invalid syntax or illegally specified scale modifier.	8
IEU012	RELOCATABLE SCALE MODIFIER	A relocatable expression has been used to specify the scale modifier.	8
IEU013	EXPONENT MODIFIER ERROR	The exponent is not specified as an absolute expression or is out of range; * in exponent modifier expression; invalid syntax; illegally specified exponent modifier.	8
IEU014	RELOCATABLE EXPONENT MODIFIER	A relocatable expression has been used to specify the exponent modifier.	8

Code	Message	Explanation	Severity Code
IEU015	INVALID LITERAL USAGE	A valid literal is used illegally, e.g., it specifies a receiving field or a register, or it is a Q-type constant.	8
IEU016	INVALID NAME	A name entry is incorrectly specified, e.g., it contains more than 8 characters, it does not begin with a letter, or has a special character imbedded.	8
IEU017	DATA ITEM TOO LARGE	The constant is too large for the data type or for the explicit length; operand field for packed DC exceeds 32 characters and for zoned DC exceeds 16 characters (excluding decimal points).	8
IEU018	INVALID SYMBOL	The symbol is specified invalidly, e.g., it is longer than 8 characters.	8
IEU019	EXTERNAL NAME ERROR	A CSECT and DSECT statement have the same name, or a symbol is used more than once in an EXTRN or the name field of DXD statements.	8
IEU020	INVALID IMMEDIATE FIELD	The value of the immediate operand exceeds 255, or the operand requires more than one byte of storage.	8
IEU021	SYMBOL NOT PREVIOUSLY DEFINED	Self-explanatory.	8
IEU022	ESDTABLE OVERFLOW	The combined number of control sections and dummy sections plus the number of unique symbols in EXTRN statements and V-type constants exceeds 255. (A DSECT which appears as XD makes two entries).	12
IEU023	PREVIOUSLY DEFINED NAME	The symbol which appears in the name field has appeared in the name field of a previous statement.	8
IEU024	UNDEFINED SYMBOL	A symbol being referenced has not been defined in the program.	8
IEU025	RELOCATABILITY ERROR	A relocatable or complex relocatable expression is specified where an absolute expression is required, an absolute expression or complex relocatable expression is specified where a relocatable expression is required, or a relocatable term is involved in multiplication or division.	8
IEU026	TOO MANY LEVELS OF PARENTHESES	An expression specifies more than 5 levels of parentheses.	12
IEU027	TOO MANY TERMS	More than 16 terms are specified in an expression.	12
IEU028	REGISTER NOT USED	A register specified in a DROP statement is not currently in use.	4

Code	Message	Explanation	Severity Code
IEU029	CCW ERROR	Bits 37-39 of the CCW are set to non-zero.	8
IEU030	INVALID CNOP	An invalid combination of operands is specified.	12
IEU031	UNKNOWN TYPE	Incorrect type designation is specified in a DC, DS, or literal.	8
IEU032	OP-CODE NOT ALLOWED TO BE GENERATED	Self-explanatory.	8
IEU033	ALIGNMENT ERROR	Referenced address is not aligned to the proper boundary for this instruction, e.g., START operand not a multiple of 8.	4
IEU034	INVALID OP-CODE	Syntax error, e.g., more than 5 characters in operation field, not followed by blank on first card, missing.	8
IEU035	ADDRESSABILITY ERROR	The referenced address does not fall within the range of a USING instruction.	8
IEU036	(No message is assigned to this number)		
IEU037	MNOTE STATEMENT	This indicates that an MNOTE statement has been generated from a macro definition. The text and severity code of the MNOTE statement will be found in line in the listing.	Variable
IEU038	ENTRY ERROR	A symbol in the operand of an ENTRY statement appears in more than one ENTRY statement, it is undefined, it is defined in a dummy section or in blank common, or it is equated to a symbol defined by an EXTRN statement.	8
IEU039	INVALID DELIMITER	This message can be caused by any syntax error, e.g., missing delimiter, special character used which is not a valid delimiter, delimiter used illegally, operand missing, i.e., nothing between delimiters, unpaired parentheses, imbedded blank in expression.	12
IEU040	GENERATED RECORD TOO LONG	There are more than 236 characters in a generated statement.	12
IEU041	UNDECLARED VARIABLE SYMBOL	Variable symbol is not declared in a defined SET symbol statement or in a macro prototype.	8
IEU042	SINGLE TERM LOGICAL EXPRESSION IS NOT A SETB SYMBOL	The single term logical expression has not been declared as a SETB symbol.	8
IEU043	SET SYMBOL PREVIOUSLY DEFINED	Self-explanatory.	8
IEU044	SET SYMBOL USAGE INCONSISTENT WITH DECLARATION	A SET symbol has been declared as undimensioned, but is subscripted, or has been declared dimensioned, but is unsubscripted.	8

Code	Message	Explanation	Severity Code
IEU045	ILLEGAL SYMBOLIC PARAMETER	An attribute has been requested for a variable symbol which is not a legal symbolic parameter.	8
IEU046	AT LEAST ONE RELOCAT- ABLE Y TYPE CONSTANT IN ASSEMBLY	Self-explanatory.	4
IEU047	SEQUENCE SYMBOL PREVIOUSLY DEFINED	Self-explanatory.	12
IEU048	SYMBOLIC PARAMETER PREVIOUSLY DEFINED OR SYSTEM VARIABLE SYMBOL DECLARED AS SYMBOLIC PARAMETER	Self-explanatory.	12
IEU049	VARIABLE SYMBOL MATCHES A PARAMETER	Self-explanatory.	12
IEU050	INCONSISTENT GLOBAL DECLARATIONS	A global SET variable symbol, defined in more than one macro definition or defined in a macro definition and in the source program, is inconsistent in SET type or dimension.	8
IEU051	MACRO DEFINITION PREVIOUSLY DEFINED	Prototype operation field is the same as a machine or assembler instruction or a previous prototype.	12
IEU052	NAME FIELD CONTAINS ILLEGAL SET SYMBOL	SET symbol in name field does not correspond to SET statement type.	8
IEU053	GLOBAL DICTIONARY FULL	The global dictionary is full, assembly terminated.	12
IEU054	LOCAL DICTIONARY FULL	The local dictionary is full, current macro aborted. If in open code, assembly terminated.	12
IEU055	INVALID ASSEMBLER OPTION(S) ON THE EXECUTE CARD	Self-explanatory.	8
IEU056	ARITHMETIC OVERFLOW	The intermediate or final result of an express- ion is not within the range of -2^{31} to $2^{31}-1$.	8
IEU057	SUBSCRIPT EXCEEDS MAXIMUM DIMENSION	&SYSLIST or symbolic parameter subscript exceeds 200, or is negative, or zero, or SET symbol subscript exceeds dimension.	8
IEU058	RE-ENTRANT CHECK FAILED	An instruction has been detected, which, when executed, might store data into a control section or a common area. This message is generated only when requested via control cards and merely indicates a possible re-entrant error.	4
IEU059	UNDEFINED SEQUENCE SYMBOL	Self-explanatory.	12
IEU060	ILLEGAL ATTRIBUTE NOTATION	L', S', or I' requested for a parameter whose type attribute does not allow these attributes to be requested.	8

Code	Message	Explanation	Severity Code
IEU061	ACTR COUNTER EXCEEDED	Self-explanatory.	12
IEU062	GENERATED STRING GREATER THAN 255 CHARACTERS	Self-explanatory.	8
IEU063	EXPRESSION 1 OF SUB-STRING IS ZERO OR MINUS	Self-explanatory.	8
IEU064	EXPRESSION 2 OF SUB-STRING IS ZERO OR MINUS	Self-explanatory.	8
IEU065	INVALID OR ILLEGAL TERM IN ARITHMETIC EXPRESSION	The value of a SETC symbol used in the arithmetic expression is not composed of decimal digits, or the parameter is not a self-defining term.	8
IEU066	UNDEFINED OR DUP-LICATE KEYWORD OPERAND OR EXCESSIVE POSITIONAL OPERANDS	The same keyword operand occurs more than once in the macro instruction; a keyword is not defined in a prototype statement; in a mixed mode macro instruction, more positional operands are specified than are specified in the prototype.	12
IEU067	EXPRESSION 1 OF SUB-STRING GREATER THAN LENGTH OF CHARACTER EXPRESSION	Self-explanatory.	8
IEU068	GENERATION TIME DICTIONARY AREA OVERFLOWED	See Dictionary Size and Source Statement Complexity.	12
IEU069	VALUE OF EXPRESSION 2 OF SUBSTRING GREATER THAN 8	Self-explanatory.	8
IEU070	FLOATING POINT CHARACTERISTIC OUT OF RANGE	Self-explanatory.	12
IEU071	ILLEGAL OCCURRENCE OF LCL, GBL, OR ACTR STATEMENT	LCL, GBL, or ACTR statement is not in proper place in the program.	8
IEU072	ILLEGAL RANGE ON ISEQ STATEMENT	Self-explanatory.	4
IEU073	ILLEGAL NAME FIELD	Either a statement which requires a name has been written without a name, or a statement has a name which is not allowed to have a name.	8
IEU074	ILLEGAL STATEMENT IN COPY CODE OR SYSTEM MACRO	Self-explanatory.	8

Code	Message	Explanation	Severity Code
IEU075	ILLEGAL STATEMENT OUTSIDE OF A MACRO DEFINITION	Self-explanatory.	8
IEU076	SEQUENCE ERROR	Self-explanatory.	12
IEU077	ILLEGAL CONTINUATION CARD	Either there are too many continuation cards, or there are non-blanks between the begin and continue columns on the continuation card.	8
IEU078	(No message is assigned to this number)		
IEU079	ILLEGAL STATEMENT IN MACRO DEFINITION	This operation is not allowed within a macro definition.	8
IEU080	ILLEGAL START CARD	Statements affecting or depending upon the location counter have been encountered before a START statement.	8
IEU081	ILLEGAL FORMAT IN GBL OR LCL STATE-MENTS	An operand is not a variable symbol.	8
IEU082	ILLEGAL DIMENSION SPECIFICATION IN GBL OR LCL STATEMENT	Dimension is other than 1 to 255.	8
IEU083	SET STATEMENT NAME FIELD NOT A VARIABLE SYMBOL	Self-explanatory.	8
IEU084	ILLEGAL OPERAND FIELD FORMAT	Syntax invalid, e.g., AIF statement operand does not start with a left parenthesis; operand of AGO is not a sequence symbol; operand of PUNCH, TITLE, MNOTE not enclosed in quotes.	8
IEU085	INVALID SYNTAX IN EXPRESSION	Invalid delimiter, too many terms in expression, too many levels of parentheses, two operators in succession, two terms in succession, or illegal character.	8
IEU086	ILLEGAL USAGE OF SYSTEM VARIABLE SYMBOL	A system variable symbol appears in the name field of a SET statement, is used in a mixed mode or keyword macro definition, is declared in a GBL or LCL statement, or is an unsubscripted &SYSLIST in a context other than N'&SYSLIST.	8
IEU087	NO ENDING APOSTROPHE	There is an unpaired apostrophe or ampersand in the statement.	8
IEU088	UNDEFINED CODE	Self-explanatory.	12
IEU089	INVALID ATTRIBUTE NOTATION	Syntax error inside a macro definition, e.g., the argument of the attribute reference is not a symbolic parameter.	8

Code	Message	Explanation	Severity Code
IEU090	INVALID SUBSCRIPT	Syntax error, e.g., double subscript where single subscript is required or vice versa; not right parenthesis after subscript.	8
IEU091	INVALID SELF-DEFINING TERM	Value is too large or is inconsistent with the data type, e.g., severity code greater than 255.	8
IEU092	INVALID FORMAT FOR VARIABLE SYMBOL	The first character after the ampersand is not alphabetic, or the variable symbol contains more than 8 characters.	8
IEU093	UNBALANCED PAREN-THESIS OR EXCESSIVE LEFT PARENTHESES	Self-explanatory.	8
IEU094	INVALID OR ILLEGAL NAME OR OPERATION IN PROTOTYPE STATEMENT	Self-explanatory.	12
IEU095	ENTRY TABLE OVERFLOW	Number of ENTRY symbols, i.e., ENTRY instruction operands, exceeds 100.	8
IEU096	MACRO INSTRUCTION OR PROTOTYPE OPERAND EXCEEDS 255 CHARAC-TERS IN LENGTH	Self-explanatory.	12
IEU097	INVALID FORMAT IN MACRO INSTRUCTION OPERAND OR PROTOTYPE PARAMETER	This message can be caused by: 1. Illegal "=". 2. A single "&" appears somewhere in the standard value assigned to a prototype keyword parameter. 3. First character of a prototype parameter is not "&". 4. Prototype parameter is a subscripted variable symbol. 5. Invalid use of alternate format in proto-type statement, e.g.,	12

```
            10                    16                    72
            PROTO                 &A,&B,
                         or
            PROTO                 &A,&B,                X
                                  &C
```

6. Unintelligible prototype parameter, e.g., "&A*" or "&A&&."
7. Illegal (non-assembler) character appears in prototype parameter or macro-instruction operand.

Code	Message	Explanation	Severity Code
IEU098	EXCESSIVE NUMBER OF OPERANDS OR PARAM-ETERS	Either the prototype has more than 200 param-eters, or the macro instruction has more than 200 operands.	12
IEU099	POSITIONAL MACRO INSTRUCTION OPERAND, PROTOTYPE PARAMETER OR EXTRA COMMA FOLLOWS KEYWORD	Self-explanatory.	12

Code	Message	Explanation	Severity Code
IEU100	STATEMENT COMPLEXITY EXCEEDED	More than 32 operands in a DC, DS, DXD, or literal DC, or more than 50 terms in a statement.	8
IEU101	EOD ON SYSIN	EOD before END card.	12
IEU102	INVALID OR ILLEGAL ICTL	The operands of the ICTL are out of range, or the ICTL is not the first statement in the input deck.	16
IEU103	ILLEGAL NAME IN OPERAND FIELD OF COPY CARD	Syntax error, e.g., symbol has more than 8 characters or has an illegal character.	12
IEU104	COPY CODE NOT FOUND	The operand of a COPY statement specified COPY text which cannot be found in the library.	12
IEU105	EOD ON SYSTEM MACRO LIBRARY	EOD before MEND card.	12
IEU106	NOT NAME OF DSECT OR DXD	Self-explanatory.	8
IEU107	INVALID OPERAND	Invalid syntax in DC operand, e.g., invalid hexadecimal character in hexadecimal DC; operand string too long for X, B, C, DC's; operand unrecognizable, contains invalid value, or incorrectly specified.	4
IEU108	PREMATURE EOD	Indicates an internal assembler error; should not occur.	16
IEU109	PRECISION LOST	Self-explanatory.	8
IEU110	EXPRESSION VALUE TOO LARGE	Value of expression greater than -16777216 to +16777215. Expressions in EQU and ORG statements are flagged if (1) they include terms previously defined as negative values, or (2) positive terms give a result of more than three bytes in magnitude. The error indication may be erroneous due to (1) the treatment of negative values as three-byte positive values, or (2) the effect of large positive values on the location counter if a control section begins with a START statement having an operand greater than zero, or a control section is divided into subsections.	8
IEU111	SYSGO DD CARD MISSING NOLOAD OPTION USED	Self-explanatory.	16
IEU112	SYSPUNCH DD CARD MISSING NODECK OPTION USED	Self-explanatory.	16
IEU997	SYSPRINT DD CARD MISSING NOLIST OPTION USED	Self-explanatory. Printed on console typewriter.	0
IEU998	ASSEMBLY TERMINATED. MISSING DATA SET FOR (ddname)	Self-explanatory. Printed on console typewriter.	20
IEU999	ASSEMBLY TERMINATED. PERMANENT I/O ERROR ON (ddname)	Self-explanatory. Printed on console typewriter.	20

TEXT (TXT) CARD FORMAT

The format of the TXT cards is as follows:

Columns	Contents
1	12-2-9 punch
2-4	TXT
5	Blank
6-8	Relative address of first instruction on card
9-10	Blank
11-12	Byte count -- number of bytes in information field (cc 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 (cc 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.

For example, if the RLD Entries 1, 2, and 3 of the program listing (Appendix C) contain the following information:

	Pos. ESDID	Rel. ESDID	Flag	Address
Entry 1	02	04	0C	000100
Entry 2	02	04	0C	000104
Entry 3	03	01	0C	000800

Columns 17-36 of the RLD card would appear as follows:
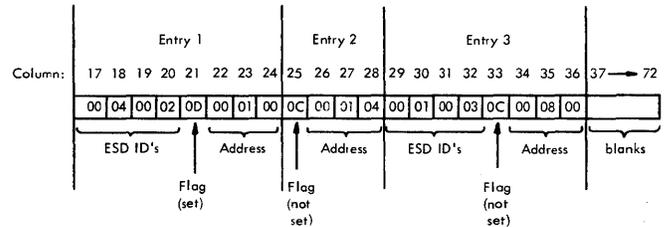
ESD CARD FORMAT

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 (cc 17-64)
13-14	Blank
15-16	ESDID of first SD, XD, CM, PC, or ER 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 -- Alignment if XD; otherwise blank; 3 bytes -- Length, LDID, or blank
65-72	Blank
73-76	Deck ID (from first TITLE card)
77-80	Card sequence number

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

TESTRAN (SYM) CARD FORMAT

If requested by the user, the assembler punches out symbolic information for TESTRAN concerning the assembled program. This output appears ahead of all loader text. The format of the card images for TESTRAN output is as follows:

Columns	Contents
1	12-2-9 punch
2-4	SYM
5-10	Blank
11-12	Variable field count -- number of bytes of text in variable field (cc 17-72)
13-16	Blank
17-72	Variable field (see below)
73-76	Deck ID (from first TITLE card)
77-80	Card sequence number

The variable field (columns 17-72) contains up to 56 bytes of TESTRAN text. The items making the text are packed together, consequently only the last card may contain less than 56 bytes of text in the variable field. The formats of a text card and an individual text item are shown in Figure 6. The contents of the fields within an individual entry are as follows:

1. Organization (1 byte)
   Bit 0:
      0 = non-data type
      1 = data type
   Bits 1-3 (if non-data type):
      000 = space
      001 = control section
      010 = dummy control section
      011 = common
      100 = instruction
      101 = CCW

   Bit 1 (if data type):
      0 = no multiplicity
      1 = multiplicity (indicates presence of M field)
   Bit 2 (if data type):
      0 = independent (not a packed or zoned decimal constant)
      1 = cluster (packed or zoned decimal constant)
   Bit 3 (if data type):
      0 = no scaling
      1 = scaling (indicates presence of S field)
   Bit 4:
      0 = name present
      1 = name not present
   Bits 5-7:
      Length of name minus one
2. Address (3 bytes) - displacement from base of control section
3. Symbol Name (0-8 bytes) - symbolic name of particular item

NOTE: The following fields are only present for data-type items.

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

      00 = character
      04 = hexadecimal
      08 = binary
      10 = fixed point, full
      14 = fixed point, half
      18 = floating point, short
      1C = floating point, long
      20 = A-type or Q-type data
      24 = Y-type data
      28 = S-type data
      2C = V-type data
      30 = packed decimal
      34 = zoned decimal
5. Length (2 bytes for character, hexadecimal, or binary items; 1 byte for other types) - length of data item minus 1
6. Multiplicity - M field (3 bytes) - equals 1 if not present
7. Scale - signed integer - S field (2 bytes) - present only for F, H, E, D, P and Z type data, and only if scale is non-zero.

1	2	4 5	10 11 12 13	16 17	72 73	76 77	80
12 2 9	SYM	blank	No. of bytes of text	blank	TESTRAN text – packed entries	Deck ID	Sequence Number
1	3	6	2	4	56	4	4

Entry (complete or end portion)	N complete entries N ≥ 1	Entry (complete or head portion)
	Variable size entries	

Org.	Address	Symbol Name	Data type	Length	Mult. factor	Scale	Org.	Symbol Name
1	3	0-8	1	1-2	3	2		
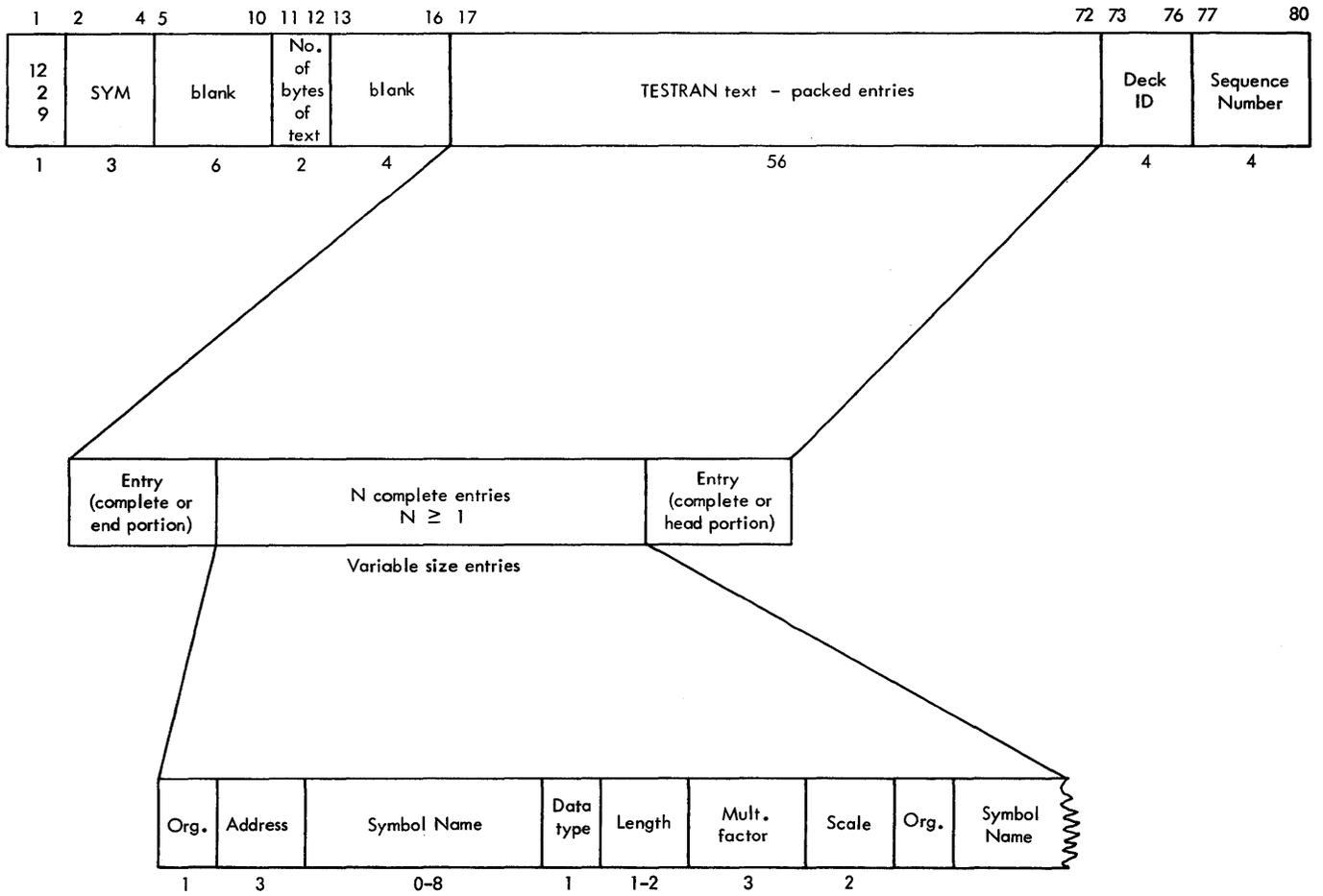
Figure 6.   TESTRAN SYM Card Format

APPENDIX C. ASSEMBLER F PROGRAM LISTING

The Assembler F listing shown in this appendix results from assembling the source program documented in an appendix to the Assembler Language publication. For easy reference to the explanations that appear in the section The Assembler Listing, the headings on the listing are numbered.

Since there were no errors in the assembly, a diagnostic list was not produced. Each of the following pages represents one printer-produced. listing page.

```
EXAM     ②  ③  ④   ⑤   ⑥        EXTERNAL SYMBOL DICTIONARY                    PAGE    1
SYMBOL   TYPE ID  ADDR  LENGTH LD ID                                          00.16  4/11/66
①
SAMPLR   SD  01 000000 000388
```

```
                        1 **      THIS IS THE EXECUTABLE SAMPLE PROGRAM SHOWN IN THE SRL -      *
                        2 **      ASSEMBLER LANGUAGE MANUAL.                                -      *
```

```
                                    4              PRINT DATA                            ⑰ SAMPL002
                                    5  *                                                   SAMPL003
                                    6  *           THIS IS THE MACRO DEFINITION            SAMPL004
                                    7  *                                                   SAMPL005
                                    8              MACRO                                   SAMPL006
                                    9              MOVE   &TO,&FROM                        SAMPL007
                                   10  .*                                                  SAMPL008
                                   11  .*          DEFINE SETC SYMBOL                      SAMPL009
                                   12  .*                                                  SAMPL010
                                   13              LCLC   &TYPE                            SAMPL011
                                   14  .*                                                  SAMPL012
                                   15  .*          CHECK NUMBER OF OPERANDS                SAMPL013
                                   16  .*                                                  SAMPL014
                                   17              AIF    (N'&SYSLIST NE 2).ERROR1         SAMPL015
                                   18  .*                                                  SAMPL016
                                   19  .*          CHECK TYPE ATTRIBUTES OF OPERANDS       SAMPL017
                                   20  .*                                                  SAMPL018
                                   21              AIF    (T'&TO NE T'&FROM).ERROR2        SAMPL019
                                   22              AIF    (T'&TO EQ 'C' OR T'&TO EQ 'G' OR T'&TO EQ 'K').TYPECGK   SAMPL020
                                   23              AIF    (T'&TO EQ 'D' OR T'&TO EQ 'E' OR T'&TO EQ 'H').TYPEDEH   SAMPL021
                                   24              AIF    (T'&TO EQ 'F').MOVE              SAMPL022
                                   25              AGO    .ERROR3                          SAMPL023
                                   26 .TYPEDEH ANOP                                        SAMPL024
                                   27  .*                                                  SAMPL025
                                   28  .*          ASSIGN TYPE ATTRIBUTE TO SETC SYMBOL    SAMPL026
                                   29  .*                                                  SAMPL027
                                   30 &TYPE        SETC   T'&TO                            SAMPL028
                                   31 .MOVE        ANOP                                    SAMPL029
                                   32  *           NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO   SAMPL030
                                   33              L&TYPE   2,&FROM                        SAMPL031
                                   34              ST&TYPE  2,&TO                          SAMPL032
                                   35              MEXIT                                   SAMPL033
                                   36  .*                                                  SAMPL034
                                   37  .*          CHECK LENGTH ATTRIBUTES OF OPERANDS     SAMPL035
                                   38  .*                                                  SAMPL036
                                   39 .TYPECGK AIF    (L'&TO NE L'&FROM OR L'&TO GT 256).ERROR4   SAMPL037
                                   40  *           NEXT STATEMENT GENERATED FOR MOVE MACRO  SAMPL038
                                   41              MVC    &TO,&FROM                        SAMPL039
                                   42              MEXIT                                   SAMPL040
                                   43  .*                                                  SAMPL041
                                   44  .*          ERROR MESSAGES FOR INVALID MOVE MACRO INSTRUCTIONS   SAMPL042
                                   45  .*                                                  SAMPL043
                                   46 .ERROR1  MNOTE 1,'IMPROPER NUMBER OF OPERANDS, NO STATEMENTS GENERATED'   SAMPL044
                                   47              MEXIT                                   SAMPL045
                                   48 .ERROR2  MNOTE 1,'OPERAND TYPES DIFFERENT, NO STATEMENTS GENERATED'   SAMPL046
                                   49              MEXIT                                   SAMPL047
                                   50 .ERROR3  MNOTE 1,'IMPROPER OPERAND TYPES, NO STATEMENTS GENERATED'   SAMPL048
                                   51              MEXIT                                   SAMPL049
                                   52 .ERROR4  MNOTE 1,'IMPROPER OPERAND LENGTHS, NO STATEMENTS GENERATED'   SAMPL050
                                   53              MEND                                    SAMPL051
                                   54  *                                                   SAMPL052
                                   55  *           MAIN ROUTINE                            SAMPL053
                                   56  *                                                   SAMPL054
000000                             57 SAMPLR   CSECT                                       SAMPL055
                                   58 BEGIN        SAVE   (14,12),,*                       SAMPL056
```

```
000000 47F0 F00A              0000A    59+BEGIN    B      10(0,15) BRANCH AROUND ID
000004 05                              60+         DC     AL1(5)
000005 C2C5C7C9D5                      61+         DC     CL5'BEGIN' IDENTIFIER                      ⑰
00000A 90EC D00C              0000C    62+         STM    14,12,12(13) SAVE REGISTERS
00000E 05C0                            63          BALR   R12,0        ESTABLISH ADDRESSABILITY OF PROGRAM    SAMPL057
000010                                 64          USING  *,R12        AND TELL THE ASSEMBLER WHAT BASE TO USE SAMPL058
000010 50D0 C0B8              000C8    65          ST     13,SAVE13                                  SAMPL059
000014 9857 C390              003A0    66          LM     R5,R7,=A(LISTAREA,16,LISTEND)  LOAD LIST AREA PARAMETERS SAMPL060
000000                                 67          USING  LIST,R5      REGISTER 5 POINTS TO THE LIST SAMPL061
000018 45E0 C0BE              000CE    68 MORE     BAL    R14,SEARCH   FIND LIST ENTRY IN TABLE      SAMPL062
00001C 9180 C0BC              000CC    69          TM     SWITCH,NONE  CHECK TO SEE IF NAME WAS FOUND SAMPL063
000020 4710 C0B0              000C0    70          BO     NOTTHERE     BRANCH IF NOT                 SAMPL064
000000                                 71          USING  TABLE,R1     REGISTER 1 NOW POINTS TO TABLE ENTRY SAMPL065
                                       72          MOVE   TSWITCH,LSWITCH          MOVE FUNCTIONS    SAMPL066
                                       73+*         NEXT STATEMENT GENERATED FOR MOVE MACRO
000024 D200 1003 5008 00003 00008      74+         MVC    TSWITCH,LSWITCH
                                       75          MOVE   TNUMBER,LNUMBER              FROM LIST ENTRY SAMPL067
                                       76+*         NEXT STATEMENT GENERATED FOR MOVE MACRO
00002A D202 1000 5009 00000 00009      77+         MVC    TNUMBER,LNUMBER
                                       78          MOVE   TADDRESS,LADDRESS            TO TABLE ENTRY SAMPL068
                                       79+*         NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO
000030 5820 500C              0000C    80+         L      2,LADDRESS
000034 5020 1004              00004    81+         ST     2,TADDRESS
000038 8756 C008              00018    82 LISTLOOP BXLE   R5,R6,MORE   LOOP THROUGH THE LIST         SAMPL069
00003C D5EF C240 C0F0 00250 00100      83          CLC    TESTTABL(240),TABLAREA                     SAMPL070
000042 4770 C07C              0008C    84          BNE    NOTRIGHT                                    SAMPL071
000046 D55F C330 C1E0 00340 001F0      85          CLC    TESTLIST(96),LISTAREA                      SAMPL072
00004C 4770 C07C              0008C    86          BNE    NOTRIGHT                                    SAMPL073
                                       87          WTO    'ASSEMBLER SAMPLE PROGRAM SUCCESSFUL'      SAMPL074
000050                                 88+         CNOP   0,4
000050 4510 C06C              0007C    89+         BAL    1,IHB0005A BRANCH AROUND MESSAGE
000054 0027                            90+         DC     AL2(IHB0005-*) MESSAGE LENGTH
000056 0000                            91+         DC     AL2(0)
000058 C1E2E2C5D4C2D3C5                92+         DC     C'ASSEMBLER SAMPLE PROGRAM SUCCESSFUL' MESSAGE
000060 D940E2C1D4D7D3C5
000068 40D7D9D6C7D9C1D4
000070 40E2E4C3C3C5E2E2
000078 C6E4D3
00007B                                 93+IHB0005  EQU    *
00007C                                 94+IHB0005A DS     0H
00007C 0A23                            95+         SVC    35 ISSUE SVC
00007E 58D0 C0B8              000C8    96 EXIT     L      R13,SAVE13                                  SAMPL075
                                       97          RETURN (14,12),RC=0                               SAMPL076
000082 98EC D00C              0000C    98+         LM     14,12,12(13) RESTORE THE REGISTERS
000086 41F0 0000              00000    99+         LA     15,0(0,0) LOAD RETURN CODE
00008A 07FE                            100+        BR     14 RETURN
                                       101 *                                                         SAMPL077
                                       102 NOTRIGHT WTO   'ASSEMBLER SAMPLE PROGRAM UNSUCCESSFUL'    SAMPL078
00008C                                 103+        CNOP   0,4
00008C 4510 C0AA              000BA    104+NOTRIGHT BAL   1,IHB0007A BRANCH AROUND MESSAGE
000090 0029                            105+        DC     AL2(IHB0007-*) MESSAGE LENGTH
000092 0000                            106+        DC     AL2(0)
000094 C1E2E2C5D4C2D3C5                107+        DC     C'ASSEMBLER SAMPLE PROGRAM UNSUCCESSFUL' MESSAGE
00009C D940E2C1D4D7D3C5
0000A4 40D7D9D6C7D9C1D4
```

Appendix C. Assembler F Program Listing   35

⑩   ⑪   ⑫   ⑬   ⑭   ⑮   ⑯
LOC   OBJECT CODE   ADDR1 ADDR2 STMT   SOURCE STATEMENT   F 14FEB66   4/11/66

```
0000AC  40E4D5E2E4C3C3C5
0000B4  E2E2C6E4D3
0000B9                                 108+IHB0007  EQU    *
0000BA                                 109+IHB0007A DS     OH                                                    ⑰
0000BA  0A23                           110+         SVC    35 ISSUE SVC
00008C  47F0 C06E              0007E   111          B      EXIT                                             SAMPL079
0000C0  9680 5008         00008        112 NOTTHERE OI     LSWITCH,NONE TURN ON SWITCH IN LIST ENTRY        SAMPL080
0000C4  47F0 C028              00038   113          B      LISTLOOP    GO BACK AND LOOP                     SAMPL081
0000C8  00000000                       114 SAVE13   DC     F'0'                                             SAMPL082
0000CC  00                             115 SWITCH   DC     X'00'                                            SAMPL083
000080                                 116 NONE     EQU    X'80'                                            SAMPL084
                                       117 *                                                               SAMPL085
                                       118 *        BINARY SEARCH ROUTINE                                  SAMPL086
                                       119 *                                                               SAMPL087
0000CD  00
0000CE  947F C08C         000CC        120 SEARCH   NI     SWITCH,255-NONE TURN OFF NOT FOUND SWITCH       SAMPL088
0000D2  9813 C39C              003AC   121          LM     R1,R3,=F'128,4,128' LOAD TABLE PARAMETERS       SAMPL089
0000D6  4111 C0E0              000F0   122          LA     R1,TABLAREA-16(R1)  GET ADDRESS OF MIDDLE ENTRY SAMPL090
0000DA  8830 0001              00001   123 LOOP     SRL    R3,1                DIVIDE INCREMENT BY 2        SAMPL091
0000DE  D507 5000 1008 00000 00008     124          CLC    LNAME,TNAME         COMPARE LIST ENTRY WITH TABLE ENTRY SAMPL092
0000E4  4720 C0E4              000F4   125          BH     HIGHER              BRANCH IF SHOULD BE HIGHER IN TABLE SAMPL093
0000E8  078E                           126          BCR    8,R14               EXIT IF FOUND               SAMPL094
                                       127          SR     R1,R3               OTHERWISE IT IS LOWER IN THE TABLE XSAMPL095
                                                                               SO SUBTRACT INCREMENT        SAMPL096
0000EA  1B13
0000EC  4620 C0CA              000DA   128          BCT    R2,LOOP             LOOP 4 TIMES                 SAMPL097
0000F0  47F0 C0EA              000FA   129          B      NOTFOUND            ARGUMENT IS NOT IN THE TABLE SAMPL098
0000F4  1A13                           130 HIGHER   AR     R1,R3               ADD INCREMENT                SAMPL099
0000F6  4620 C0CA              000DA   131          BCT    R2,LOOP             LOOP 4 TIMES                 SAMPL100
0000FA  9680 C08C         000CC        132 NOTFOUND OI     SWITCH,NONE         TURN ON NOT FOUND SWITCH     SAMPL101
0000FE  07FE                           133          BR     R14                 EXIT                         SAMPL102
                                       134 *                                                               SAMPL103
                                       135 *        THIS IS THE TABLE                                      SAMPL104
                                       136 *                                                               SAMPL105
000100                                 137          DS     0D                                               SAMPL106
000100  0000000000000000               138 TABLAREA DC     XL8'0',CL8'ALPHA'                                SAMPL107
000108  C1D3D7C8C1404040
000110  0000000000000000               139          DC     XL8'0',CL8'BETA'                                 SAMPL108
000118  C2C5E3C140404040
000120  8000000000000000               140          DC     XL8'0',CL8'DELTA'                                SAMPL109
000128  C4C5D3E3C1404040
000130  0000000000000000               141          DC     XL8'0',CL8'EPSILON'                              SAMPL110
000138  C5D7E2C9D3D6D540
000140  0000000000000000               142          DC     XL8'0',CL8'ETA'                                  SAMPL111
000148  C5E3C14040404040
000150  0000000000000000               143          DC     XL8'0',CL8'GAMMA'                                SAMPL112
000158  C7C1D4D4C1404040
000160  0000000000000000               144          DC     XL8'0',CL8'IOTA'                                 SAMPL113
000168  C9D6E3C140404040
000170  0000000000000000               145          DC     XL8'0',CL8'KAPPA'                                SAMPL114
000178  D2C1D7D7C1404040
000180  0000000000000000               146          DC     XL8'0',CL8'LAMBDA'                               SAMPL115
000188  D3C1D4C2C4C14040
000190  0000000000000000               147          DC     XL8'0',CL8'MU'                                   SAMPL116
000198  D4E4404040404040
0001A0  0000000000000000               148          DC     XL8'0',CL8'NU'                                   SAMPL117
```

```
                                                                                                    ⑰
0001A8 D5E4404040404040
0001B0 0000000000000000        149         DC    XL8'0',CL8'OMICRON'                          SAMPL118
0001B8 D6D4C9C3D9D6D540
0001C0 0000000000000000        150         DC    XL8'0',CL8'PHI'                              SAMPL119
0001C8 D7C8C94040404040
0001D0 0000000000000000        151         DC    XL8'0',CL8'SIGMA'                            SAMPL120
0001D8 E2C9C7D4C1404040
0001E0 0000000000000000        152         DC    XL8'0',CL8'ZETA'                             SAMPL121
0001E8 E9C5E3C140404040
                               153 *                                                          SAMPL122
                               154 *           THIS IS THE LIST                               SAMPL123
                               155 *                                                          SAMPL124
0001F0 D3C1D4C2C4C14040        156 LISTAREA DC  CL8'LAMBDA',X'0A',FL3'29',A(BEGIN)            SAMPL125
0001F8 0A00001D00000000
000200 E9C5E3C140404040        157         DC    CL8'ZETA',X'05',FL3'5',A(LOOP)               SAMPL126
000208 05000005000000DA
000210 E3C8C5E3C1404040        158         DC    CL8'THETA',X'02',FL3'45',A(BEGIN)            SAMPL127
000218 0200002D00000000
000220 E3C1E44040404040        159         DC    CL8'TAU',X'00',FL3'0',A(1)                   SAMPL128
000228 0C0000000000001
000230 D3C9E2E340404040        160         DC    CL8'LIST',X'1F',FL3'465',A(0)                SAMPL129
000238 1F00C1D100000000
000240 C1D3D7C8C1404040        161 LISTEND  DC  CL8'ALPHA',X'00',FL3'1',A(123)                SAMPL130
000248 000000010000007B
                               162 *                                                          SAMPL131
                               163 *           THIS IS THE CONTROL TABLE                       SAMPL132
                               164 *                                                          SAMPL133
000250                         165         DS    0D                                           SAMPL134
000250 000001000000007B        166 TESTTABL DC  FL3'1',X'00',A(123),CL8'ALPHA'                SAMPL135
000258 C1D3D7C8C1404040
000260 0000000000000000        167         DC    XL8'0',CL8'BETA'                             SAMPL136
000268 C2C5E3C140404040
000270 0000000000000000        168         DC    XL8'0',CL8'DELTA'                            SAMPL137
000278 C4C5D3E3C1404040
000280 0000000000000000        169         DC    XL8'0',CL8'EPSILON'                          SAMPL138
000288 C5D7E2C9D3D6D540
000290 0000000000000000        170         DC    XL8'0',CL8'ETA'                              SAMPL139
000298 C5E3C14040404040
0002A0 0000000000000000        171         DC    XL8'0',CL8'GAMMA'                            SAMPL140
0002A8 C7C1D4D4C1404040
0002B0 0000000000000000        172         DC    XL8'0',CL8'IOTA'                             SAMPL141
0002B8 C9D6E3C140404040
0002C0 0000000000000000        173         DC    XL8'0',CL8'KAPPA'                            SAMPL142
0002C8 D2C1D7D7C1404040
0002D0 00001D0A00000000        174         DC    FL3'29',X'0A',A(BEGIN),CL8'LAMBDA'           SAMPL143
0002D8 D3C1D4C2C4C14040
0002E0 0000000000000000        175         DC    XL8'0',CL8'MU'                               SAMPL144
0002E8 D4E4404040404040
0002F0 0000000000000000        176         DC    XL8'0',CL8'NU'                               SAMPL145
0002F8 D5E4404040404040
000300 0000000000000000        177         DC    XL8'0',CL8'OMICRON'                          SAMPL146
000308 D6D4C9C3D9D6D540
000310 0000000000000000        178         DC    XL8'0',CL8'PHI'                              SAMPL147
000318 D7C8C94040404040
000320 0000000000000000        179         DC    XL8'0',CL8'SIGMA'                            SAMPL148
```

```
000328 E2C9C7D4C1404040
000330 00000505000000DA       180           DC    FL3'5',X'05',A(LOOP),CL8'ZETA'            SAMPL149
000338 E9C5E3C140404040
                              181  *                                                        SAMPL150
                              182  *         THIS IS THE CONTROL LIST                        SAMPL151
                              183  *                                                         SAMPL152
000340 D3C1D4C2C4C14040       184  TESTLIST DC    CL8'LAMBDA',X'0A',FL3'29',A(BEGIN)         SAMPL153
000348 0A000001D00000000
000350 E9C5E3C140404040       185           DC    CL8'ZETA',X'05',FL3'5',A(LOOP)            SAMPL154
000358 050000050000000DA
000360 E3C8C5E3C1404040       186           DC    CL8'THETA',X'82',FL3'45',A(BEGIN)         SAMPL155
000368 8200002D00000000
000370 E3C1E4404C404040       187           DC    CL8'TAU',X'80',FL3'0',A(1)                SAMPL156
000378 8000000000000001
000380 D3C9E2E340404040       188           DC    CL8'LIST',X'9F',FL3'465',A(0)             SAMPL157
000388 9F0001D100000000
000390 C1D3D7C8C1404040       189           DC    CL8'ALPHA',X'00',FL3'1',A(123)            SAMPL158
000398 000000010000007B
                              190  *                                                        SAMPL159
                              191  *         THESE ARE THE SYMBOLIC REGISTERS               SAMPL160
                              192  *                                                         SAMPL161
000000                        193  R0       EQU   0                                         SAMPL162
000001                        194  R1       EQU   1                                         SAMPL163
000002                        195  R2       EQU   2                                         SAMPL164
000003                        196  R3       EQU   3                                         SAMPL165
000005                        197  R5       EQU   5                                         SAMPL166
000006                        198  R6       EQU   6                                         SAMPL167
000007                        199  R7       EQU   7                                         SAMPL168
00000C                        200  R12      EQU   12                                        SAMPL169
00000D                        201  R13      EQU   13                                        SAMPL170
00000E                        202  R14      EQU   14                                        SAMPL171
00000F                        203  R15      EQU   15                                        SAMPL172
                              204  *                                                        SAMPL173
                              205  *         THIS IS THE FORMAT DEFINITION OF LIST ENTRIES  SAMPL174
                              206  *                                                         SAMPL175
000000                        207  LIST     DSECT                                           SAMPL176
000000                        208  LNAME    DS    CL8                                       SAMPL177
000008                        209  LSWITCH  DS    C                                         SAMPL178
000009                        210  LNUMBER  DS    FL3                                       SAMPL179
000000                        211  LADDRESS DS    F                                         SAMPL180
                              212  *                                                        SAMPL181
                              213  *         THIS IS THE FORMAT DEFINITION OF TABLE ENTRYS  SAMPL182
                              214  *                                                         SAMPL183
000000                        215  TABLE    DSECT                                           SAMPL184
000000                        216  TNUMBER  DS    FL3                                       SAMPL185
000003                        217  TSWITCH  DS    C                                         SAMPL186
000004                        218  TADDRESS DS    F                                         SAMPL187
000008                        219  TNAME    DS    CL8                                       SAMPL188
000000                        220           END   BEGIN                                     SAMPL189
000000
0003A0
0003A0
0003A0 000001F0               221                 =A(LISTAREA,16,LISTEND)
0003A4 0000008000000004       222                 =F'128,4,128'
0003AC 00000080
```

38

⑦
EXAM

⑱   ⑲   ⑳   ㉑   ⑯
POS.ID   REL.ID   FLAGS   ADDRESS   4/11/66

        01   01   0C   0001FC
        01   01   0C   00020C
        01   01   0C   00021C
        01   01   0C   0002D4
        01   01   0C   000334
        01   01   0C   00034C
        01   01   0C   00035C
        01   01   0C   00036C
        01   01   0C   0003A0

㉒ SYMBOL	㉓ LEN	㉔ VALUE	㉕ DEFN	㉖ REFERENCES					
BEGIN	00004	000000	0059	0156	0158	0174	0184	0186	0220
EXIT	00004	00007E	0096	0111					
HIGHER	00002	0000F4	0130	0125					
IHB0005	00001	00007B	0093	0090					
IHB0005A	00002	00007C	0094	0089					
IHB0007	00001	0000B9	0108	0105					
IHB0007A	00002	0000BA	0109	0104					
LADDRESS	00004	00000C	0211	0080					
LIST	00001	000000	0207	0067					
LISTAREA	00008	0001F0	0156	0066	0085	0221			
LISTEND	00008	000240	0161	0066	0221				
LISTLOOP	00004	000038	0082	0113					
LNAME	00008	000000	0208	0124					
LNUMBER	00003	000009	0210	0077					
LOOP	00004	0000DA	0123	0128	0131	0157	0180	0185	
LSWITCH	00001	000008	0209	0074	0112				
MORE	00004	000018	0068	0082					
NONE	00001	000080	0116	0069	0112	0120	0132		
NOTFOUND	00004	0000FA	0132	0129					
NOTRIGHT	00004	00008C	0104	0084	0086				
NOTTHERE	00004	0000C0	0112	0070					
R0	00001	000000	0193						
R1	00001	000001	0194	0071	0121	0122	0122	0127	0130
R12	00001	00000C	0200	0063	0064				
R13	00001	00000D	0201	0096					
R14	00001	00000E	0202	0068	0126	0133			
R15	00001	00000F	0203						
R2	00001	000002	0195	0128	0131				
R3	00001	000003	0196	0121	0123	0127	0130		
R5	00001	000005	0197	0066	0067	0082			
R6	00001	000006	0198	0082					
R7	00001	000007	0199	0066					
SAMPLR	00001	000000	0057	0220					
SAVE13	00004	0000C8	0114	0065	0096				
SEARCH	00004	0000CE	0120	0068					
SWITCH	00001	0000CC	0115	0069	0120	0132			
TABLAREA	00008	000100	0138	0083	0122				
TABLE	00001	000000	0215	0071					
TADDRESS	00004	000004	0218	0081					
TESTLIST	00008	000340	0184	0085					
TESTTABL	00003	000250	0166	0083					
TNAME	00008	000008	0219	0124					
TNUMBER	00003	000000	0216	0077					
TSWITCH	00001	000003	0217	0074					

NO STATEMENTS FLAGGED IN THIS ASSEMBLY
  351 PRINTED LINES

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

If the Assembler is invoked by CALL, LINK, or ATTACH, the user may supply:

1) The Assembler options
2) The ddnames of the data sets to be used during processing

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

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

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

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

The option list must begin on a half-word boundary. The first two bytes contain a count of the number of bytes in the remainder of the list. If no options are specified, the count must be zero. The option list is free form with each field separated by a comma. No blanks or zeros should appear in the list.

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

The ddname list must begin on a half-word boundary. The first two bytes contain a count of the number of bytes in the remainder of the list. Each name of less than eight bytes must be left-justified and padded with blanks. If an alternate ddname is omitted, the standard name will be assumed. If the name is omitted within the list, the 8-byte entry must contain binary zeros. Names can be omitted from the end merely by shortening the list. The sequence of the 8-byte entries in the ddname list is as follows:

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

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

C26-3756-2

IBM

IBM System/360 Operating System Assembler (F) Programmer's Guide

The attached pages bring the above publication up to date. Changes
are indicated by a vertical line at the left of affected text, a
bullet (•) at the left of the title of a changed illustration, and
a bullet beside the page number of a page that should be reviewed
in its entirety. Pages that contain changes are coded in the upper
outside corner.

Replace the following pages:

iii and iv
1 and 2
3 and 4
5 and 6
7 and 8
9 and 10
29 and 30
41 and 42

Add the following pages:

9.1 and 10
43 and 44

The changes documented by the attached replacement pages include,

● Change the default entry to specify LINECNT=55 (Release 14).

● Delete a SYSLIB restriction (Release 13).

● Add Dynamic Invocation of the Assembler (Release 14).

● Revise Table 1.

● Add two examples for overriding cataloged procedures.

● Miscellaneous format, editorial and technical corrections.

File this cover letter at the back of the publication. It will then
serve as a record of the changes received and incorporated.