Z28-6682-2

BSL

IBM®

BSL User's Guide

IBM Confidential

Fifth Edition (May, 1969)

This publication is a major revision of, and obsoletes, Z28-6682-1.

A new section on compiler-generated code is included in this edition. The information on compile-time facilites has been expanded. New and modified compiler options are covered. Significant changes in technical information (additions, deletions, alterations) can also be found under the following headings:

Compiler Operation
Summary of Compiler Options (Table 1)
Modifying Dictionary Size, Source Margins, and Control Character
Using the GENMGIN Option
Ending a Block GENERATE
Options to Print or Suppress Source Listings
Options to Format Source Listings
Options to Punch or Store Assembler Text
Options to Annotate Assembler Text
Description of DOS Files Used by the Compiler (Table 2)
Compilation Without Assembly (DOS)
Compilation, Assembly, and Execution (DOS)
Storing Text in External Libraries (DOS)
Specifying INCLUDE Option of Macro Statement (DOS)
Description of OS Data Sets Used by the Compiler (Table 4)
Cataloged Procedure BSLX (Figure 7)
Cataloged Procedure BSLASM (Figure 8)
Cataloged Procedure BSLALG (Figure 9)
Alternate DD Names (Table 5)
Labels Generated by the Compiler (Table 7)
BSL Size Restrictions (Table 10)
Initialization
String Items
Subscript Notation
Substring Notation
Avoiding Parameter References
The Compile-Time Processor and the GENERATE Statement
Program Parameterization
Compile-Time Macro Processor (Formerly Appendix B. Has been deleted)
Diagnostic Messages and Codes (Appendix E)

Changes to the text, and small changes to the illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

Specifications contained herein are subject to change from time to time. Any such change will be reported in the BSL Bulletin, an internally distributed newsletter covering various BSL topics.

Copies of BSL publications are available from Department D76, Systems Development Division, Poughkeepsie, New York. Requests for the addition of names to the BSL Bulletin distribution list should also be directed to this address.

Preface

This publication provides you (the systems programmer) with the information necessary to compile, assemble, linkage edit, and execute programs written in the Basic Systems Language (BSL).

Section 1 is an introduction to the functions of the BSL compiler. Section 2 describes the various compiler options that you can specify.

Section 3 explains how to process BSL programs with the IBM System/360 Disk Operating System (DOS); and Section 4 covers the same type of material for the IBM System/360 Operating System (OS). In each of these sections, knowledge of the particular operating system is assumed.

Section 5 discusses compiler conventions for the generation of assembler text output and the restrictions that affect the type of program you can successfully compile.

Section 6 deals with compiler-generated code. Section 7 describes programming techniques. The initial Appendixes contain examples of coding and compiler output. The final Appendix lists and explains the diagnostic messages that may occur during use of the compiler.

Additional information about BSL can be obtained from the following publications:

<u>Basic Systems Language Primer</u>, Form Z28-6678

<u>BSL Language Specification</u>, Form Z28-6642

<u>BSL Library</u>

Contents

Illustrations

Figures

Tables

Examples

Section 1: Introduction

The translation of a Basic Systems Language (BSL) source program into a System/360 object program requires a compilation and an assembly:

1. The BSL compiler translates the BSL source statements into assembler language instructions which serve as input to a System/360 assembler. Usually, a number of assembler language instructions result from each BSL source statement.

2. The System/360 assembler translates the compiler's output into an object module.

Figure 1 illustrates the flow of a BSL source program through the two steps of translation, and through subsequent steps of linkage editing and execution. As shown, the object module produced by the assembler is processed by a linkage editor to add any routines required from BSL or user libraries. The resulting load module is loaded into main storage, and control is passed to it for execution.

Compiler Operation

The BSL compiler runs as a problem program under the IBM System/360 Disk Operating System (DOS) or the IBM System/360 Operating System (OS). Under DOS, the compiler requires a machine configuration with 128K bytes of main storage. Under OS, the compiler requires 100,000 bytes of main storage in addition to the storage required by the operating system, and the compiler's dictionary (see SIZE option), as well as input and output buffers.

You use an appropriate sequence of job control statements to execute the BSL compiler as a job step under DOS or OS. Optionally, you can follow the compilation with job steps that perform assembly, linkage editing, and execution of the program. For OS users, a set of cataloged procedures facilitates the handling of these operations:

- BSLX is a cataloged procedure for the compile-only operation.

- BSLASM is a cataloged procedure for compilation and assembly.

- BSLALG is a cataloged procedure for compilation, assembly, linkage editing, and execution.

In addition to executing the compiler as a job step, OS users can invoke the compiler dynamically during execution of a problem program.

The BSL compiler offers many options which you can use to modify the compiler's input, modify the compiler's output, and test and debug your program. Most compiler options must be specified in compiler control statements that you include with the BSL source program in the compiler input. Compiler control statements are 80-byte records usually identified by a dollar sign ($) control character in the first byte. Three special options (SIZE, SORMGIN, and CONCHAR) are available only for OS compilations, and are specified in the PARM field of the EXEC statement.

Compiler-Generated Assembler Text

Compilation of a BSL external procedure produces one assembler language control section (CSECT). Separate control sections are not generated for BSL internal procedures.

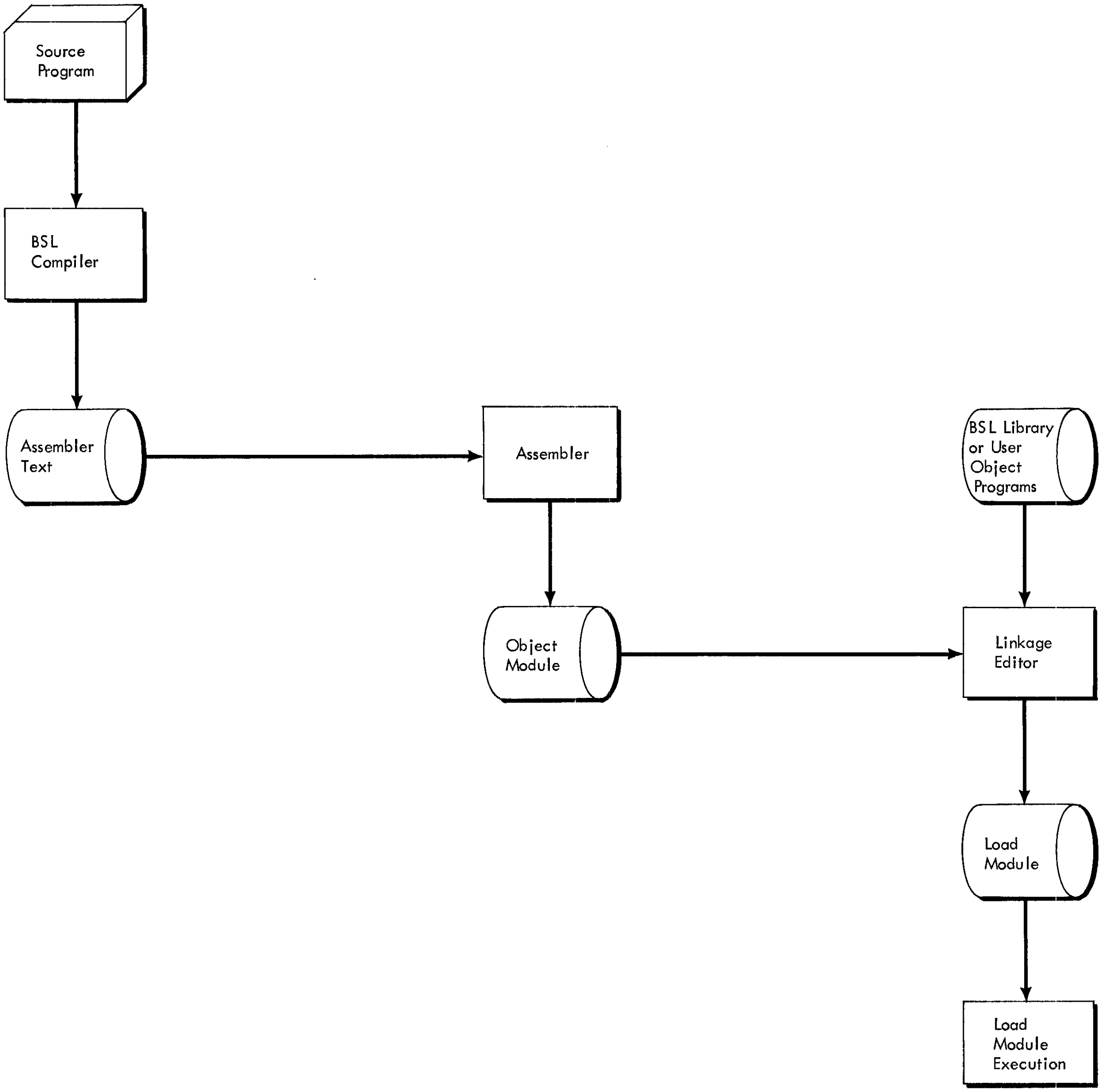


Figure 1. Basic Flowchart for Handling a BSL Program

The compiler follows certain conventions when it translates your BSL source program into assembler text. These conventions concern program format, statement labels, register usage, and program linkage. You will need this information to communicate with or modify BSL-produced programs, or to include assembler text in your BSL source program. In addition, there are certain size and language restrictions which affect the type of program you can successfully compile.

The assembler text output of the BSL compiler does not depend on any system macro instructions; therefore, it can be assembled, loaded, and executed under any System/360 programming system. (Exception: the REENTRANT procedure option can cause the compiler to generate a GETMAIN macro instruction.)

Information Listings

To assist in the development of your program, the BSL compiler produces information listings for each compilation. The listings include:

- The BSL source statements.
- An attribute and cross-reference table.
- Diagnostic messages.
- The generated assembler text.

You can use compiler options to partly suppress these listings or to change the listing format.

When an incorrect statement is encountered in your BSL source program, the compiler flags the statement and writes a diagnostic message. The messages are numbered in the form snn; where s is an alphabetic character that indicates the severity level of the error, and nn is an integer that numbers the message within its severity group. Messages from the BSL compile-time macro facility are numbered in the same manner, but are prefixed by the letter M.

Note: The examples of coding and job control statements contained in this manual are presented for your guidance. In actual use, these examples should be modified to suit your particular needs.

Section 2: Compiler Options

This section describes all options available with the BSL compiler. You use the options to:

- Modify or supplement the compiler input.
- Modify, suppress, or supplement the compiler output.
- Trace entry points and branches.

Table 1 is a summary of all compiler options. For each item, the table shows the form of specification, the placement in the input stream, the general function, and the default condition. The items are listed in the same order as they are described later in this section.

Table 1. Summary of Compiler Options (Part 1 of 3)

Specification	Placement[1]	Function	Default
SIZE=number[2]	PARM	Indicates the size of the compiler dictionary.	SIZE=30
SORMGIN=(xxx,yyy)[2]	PARM	Indicates the margins of the compiler input records.	SORMGIN=(001,072)
CONCHAR=x[2]	PARM	Specifies the character that identifies compiler control statements.	CONCHAR=$
INCLUDE {ddname(member) or sublibrary (bookname)}	$c	Includes source statements from external library.	
MACRO	$a	Invokes the compile-time macro facility.	No macro phase[3]
GENMGIN=(xxx,yyy)	$a	Specifies columns to be used as input for assembler language in a block GENERATE.	Column 1 to the right column of the the object margin.[4]
ENDGEN	$c	Indicates the end of a block GENERATE.	
$$$	$d	Control characters assigned by CONCHAR that separate BSL external procedures in batch compilation.	

(Part 1 of 3)

Table 1. Summary of Compiler Options (Part 2 of 3)

Specification	Placement[1]	Function	Default
NOLIST	$a	Suppresses all compiler listings except for control statements, time, and error messages.	Listing[3]
XREF	$a	Causes attribute and cross-reference table to be listed.	XREF. NOXREF if NOLIST is specified.
NOXREF	$a	Suppresses the attribute and cross-reference table.	Attribute and cross-reference table[3]
MSGLEVEL=$\left\{\begin{matrix}0\\1\end{matrix}\right\}$	$a	Controls the printing of warning messages.	MSGLEVEL=0
TITLE='title'	$a	Provides a heading for each page of compiler output.	No title[3]
PAGE=number	$a	Indicates the starting number for output page numbering.	PAGE=0001
NOPAGE	$a	Suppress page numbers on compiler output.	Page numbers[3]
EJECT	$b	Causes a page eject in the BSL source program listing.	No eject[3]
TIME	$a	Obtains a printing of the compilation time.	No time[3]
LINE	$a	Input line number replaces statement number in attribute and cross-referenece table and in error messages.	Uses statement numbers.
NOSNUMBER	$a	Suppresses assembler text references to the BSL statement numbers.	BSL statement number references[3]
SEQ= ('character string', left col., right col.)	$a	Indicates character string and starting sequence number to be placed between specified columns of assembler text.	SEQ= ('ƀƀƀ00000',73,80)
INCR= ('increment value,' left col., right col.)	$a	Indicates the increment value for the character string and sequence number given in SEQ option for assembler text.	INCR=('00010',76,80)

(Part 2 of 3)

Table 1. Summary of Compiler Options (Part 3 of 3)

Specification	Placement[1]	Function	Default
RESEQ= ('character string', left col., right col.)	$b	Indicates character string to replace character string in SEQ statement	No change in SEQ number
NOSEQ	$a	Suppresses assembler text sequence numbers.	Sequence numbers[3]
INTER	$a	Intermixes the generated assembler text with the BSL source statements on the source listing.	No intermix[3]
ANNOTATE=$\left\{\begin{matrix}0\\1\\2\end{matrix}\right\}$	$a	Adds comments to the generated assembler text.	No annotate[3]
PUNCH	$a	Provides punched output of the generated assembler text.	No punch[3]
ASSEM	$a	Writes assembler text on auxiliary storage for subsequent use.	ASSEM
NOASSEM	$a	No assembler text written for subsequent use.	ASSEM
TRACE [OFFSET=n]	$a	Specifies program tracing.	No trace[3]
TRACE ON	$c	Indicates the starting point for the trace operation.	
TRACE OFF	$b	Indicates the stopping point for the trace operation.	

[1] Placement Codes:
- PARM -- In the PARM field of the EXEC statement for the BSL compiler.
- $a -- In a control statement preceding the BSL source program. (Free-form)
- $b -- In a control statement within the BSL source program. (Not free-form)
- $c -- In a control statement anywhere in the compiler input. (Not free-form)
- $d -- Between BSL external procedures. (Not free-form)

[2] Options available only with OS.

[3] The default cannot be specified in a control statement.

[4] The object margins are column 1 and the last column of output before sequencing information begins.

Specifying the Options

Three special options (SIZE, SORMGIN, and CONCHAR) are available only for OS compilations, and are specified in the PARM field of the EXEC statement. All other options must be specified in compiler control statements (i.e., control cards) which you include with the BSL source program in the compiler input.

Parameters of the EXEC Statement (OS Only)

To use the SIZE, SORMGIN, or CONCHAR options, you must specify the appropriate keyword in the PARM field of the EXEC statement for the OS compiler job step. You need only specify the options you require, and they can be specified in any order.

The following example shows how to code options in the PARM field of an EXEC statement that invokes the BSL compiler:

```
//STEPA EXEC PGM=BSL,PARM='SORMGIN=(003,072),SIZE=96'
```

The following example shows how to code options in the PARM field of an EXEC statement that uses one of the BSL cataloged procedures:

```
//STEPA EXEC BSLALG,PARM.STEP1='SIZE=96,CONCHAR=@'
```

Compiler Control Statements

All compiler options except SIZE, SORMGIN, and CONCHAR are specified in compiler control statements, as follows:

- A control statement is an 80-byte record -- usually a card. The compiler scans columns 1-72 for option specifications. (Under OS, you may use the SORMGIN option to change the beginning and ending columns.) A control statement may not be continued beyond a single card, but you may use as many control statements as you need.

- A control statement is identified by a dollar sign ($) in the first column. (Under OS, the first column refers to the left-most source margin, and you can use the CONCHAR option to establish an identification character other than the dollar sign.)

- Most options can be coded free-form in the control statement. One or more of these free-form options can be included in a single control statement. The options must be separated by commas, but the use of blanks is not restricted.

- The following options are not free-form: INCLUDE, ENDGEN, TRACE ON, TRACE OFF, and EJECT. Each of these options must be coded in a separate control statement, and must immediately follow (no space) the control character.

The following example shows several ways you might code any of the free-form options:

```
$ASSEM,TIME,NOSEQ

    or

$ ASSEM,     TIME
$       NOSEQ

    or

$ASSEM
$TIME
$NOSEQ
```

The following example shows how you must code the options which are not free-form:

```
$INCLUDE ddname(member name)
$ENDGEN
$TRACE ON
$TRACE OFF
$EJECT
```

Control statements are included with the BSL source statements in the compiler input. Compiler control statements must precede the BSL source statements, except as follows:

- Control statements that specify the ENDGEN or TRACE ON options can appear before or within the BSL source statements.
- Control statements that specify the EJECT or TRACE OFF options can only appear within the BSL source statements; that is, they must not appear before the start of the source program.

Modifying Compiler Input

To modify compiler input, you can choose from a number of available options. These options are described in the following paragraphs.

Modifying Dictionary Size, Source Margins, and Control Character

If you compile under OS, you may adjust the compiler's dictionary size to provide more efficient storage utilization. You may also change the margins on the source statements or change the control character to avoid conflicts with other processors. To accomplish these modifications, you specify the appropriate options in the PARM field of the EXEC statement for the compiler job step. (If the compiler is dynamically invoked, these options may be passed as parameters -- see Section 4.) These options are:

SIZE=number
    indicates the size of the compiler dictionary. You specify a three-digit decimal number which, when multiplied by 1000, determines the number of bytes of storage to be allocated for dictionary space. You may specify a number from 001 to 128. If you do not specify this option, the compiler uses a dictionary size of 30,000 bytes.

To determine the approximate dictionary size that is required to compile an external procedure, use the following formula:

Approximate Dictionary Size (in bytes)=(37*V1)+(4*V2)+V3 where:

V1 = the total number of identifiers in the program.

V2 = the total number of references to all identifiers in the program. (This value is zero if you specify the NOXREF option.)

V3 = the total number of characters included in INITIAL attributes.

If you compile under DOS, you cannot specify the SIZE option. The compiler uses available storage for its dictionary space.

SORMGIN=(xxx,yyy)
indicates the margins of the source and control statements to be read by the compiler. You must specify three-digit decimal numbers for xxx and yyy, where:

xxx is the left margin. The compiler begins scanning each input statement at the column designated by xxx. The number can be from 001 to 065.

yyy is the right margin. It is the last column scanned by the compiler for possible input. The number can be from 015 to 080.

The number xxx must be less than the number yyy, and the difference between the numbers must provide an input record of at least 15 columns. If you do not specify the source-margin option, the compiler uses columns 001 through 072.

The source-margin option does not control the margins of assembler text within a block GENERATE statement. The margins of the assembly text are controlled by the GENMGIN option.

If you compile under DOS, you cannot specify the SORMGIN option. The compiler always uses columns 1-72 for the source margins.

CONCHAR=x
indicates the character that appears in the first column (leftmost margin) of all compiler control statements. You may specify any EBCDIC character except the single quote ('). If you do not use the CONCHAR option, the compiler identifies control statements by the appearance of a dollar sign ($) in the first column.

If you compile under DOS , you cannot specify the CONCHAR option. The compiler always uses the dollar sign ($) for the control character.

When you compile under OS, there are two special cases that require use of the SORMGIN or CONCHAR options. These are:

1. When you want to use the output of the PL/I macro processor as input to the BSL compiler, you must specify SORMGIN=(002,072) for your BSL compilation. This is necessary because the output margins of the PL/I macro processor are always 2-72.

2. When you want to maintain BSL source programs as libraries under the CLEAR system, then the dollar sign ($) must not appear in columns one or two of the BSL source statements. The CLEAR system has a reserved meaning for the dollar sign in these positions. To avoid conflict with CLEAR, you can use the SORMGIN option to establish column three as the left source margin, or you can use the CONCHAR option to change the dollar sign identification. (See "Using the Clear Facilities," Section 4.)

Including BSL Source Statements From a Library

The INCLUDE option allows you to keep certain standard declarations or sections of source code in external libraries. As needed, these items can be obtained from the libraries during compilation, and included in your programs.

The INCLUDE option is specified in a compiler control statement. For a complete description of the option and the libraries, refer to Section 4 which describes OS compilation or Section 3 which describes DOS compilation.

Using the Compile-Time MACRO Facility

The MACRO option invokes the compiler's macro phase (a compile-time modification of the source text). The macro phase starts immediately following the compiler control statement that contains the MACRO option, and processes all input from that point to the end of file or $$$. You should note that:

- The macro phase and the compile phase use the same source margins (see SORMGIN option).

- The control character in macro statements is always %, and is not affected by the CONCHAR option.

The macro phase writes the modified source text on intermediate storage for subsequent use in the compile phase. Under DOS, the intermediate text is directed to the file with the symbolic name SYS002. Under OS, the intermediate text is directed to the data set with the ddname SYSUT3.

Using the GENMGIN Option

The GENMGIN option specifies the columns that will be used as input for the assembly language in a block GENERATE. The format of the option is GENMGIN=(xxx,yyy).

xxx indicates the left input column. This column will be put in column 1 of the generated output.

yyy indicates the right input column. This column will be put in the continuation column of the output.

If GENMGIN specifies columns that are the same as the object margin, then the code to be generated remains in the same columns. (If no object margins are specified, the default values are columns 1 and 72.) If, for example, the specification is:

```
GENMGIN=(001,072)
```

the code to be generated remains in the same columns.

If GENMGIN specifies fewer input columns than there are output columns between the object margins, blanks are generated for the difference. The rightmost column of input will be placed in the continuation column. If, for example, the specification is:

```
GENMGIN=(005,071)
```

the BSL compiler will put the code to be generated into columns 1 through 67. Blanks will be generated to column 72. The character that is placed in column 67 will also be placed in column 72.

If GENMGIN specifies more input columns than there are output columns between the object margins, truncation will occur. The right most column of input will be placed in the continuation column. If, for example, the specification is:

```
GENMGIN=(001,079)
```

the BSL compiler will put columns 1 through 71 of the code to be generated into columns 1 through 71. Characters in columns 72 through 78 will be truncated. The character in column 79 will be placed in column 72.

Ending a Block GENERATE

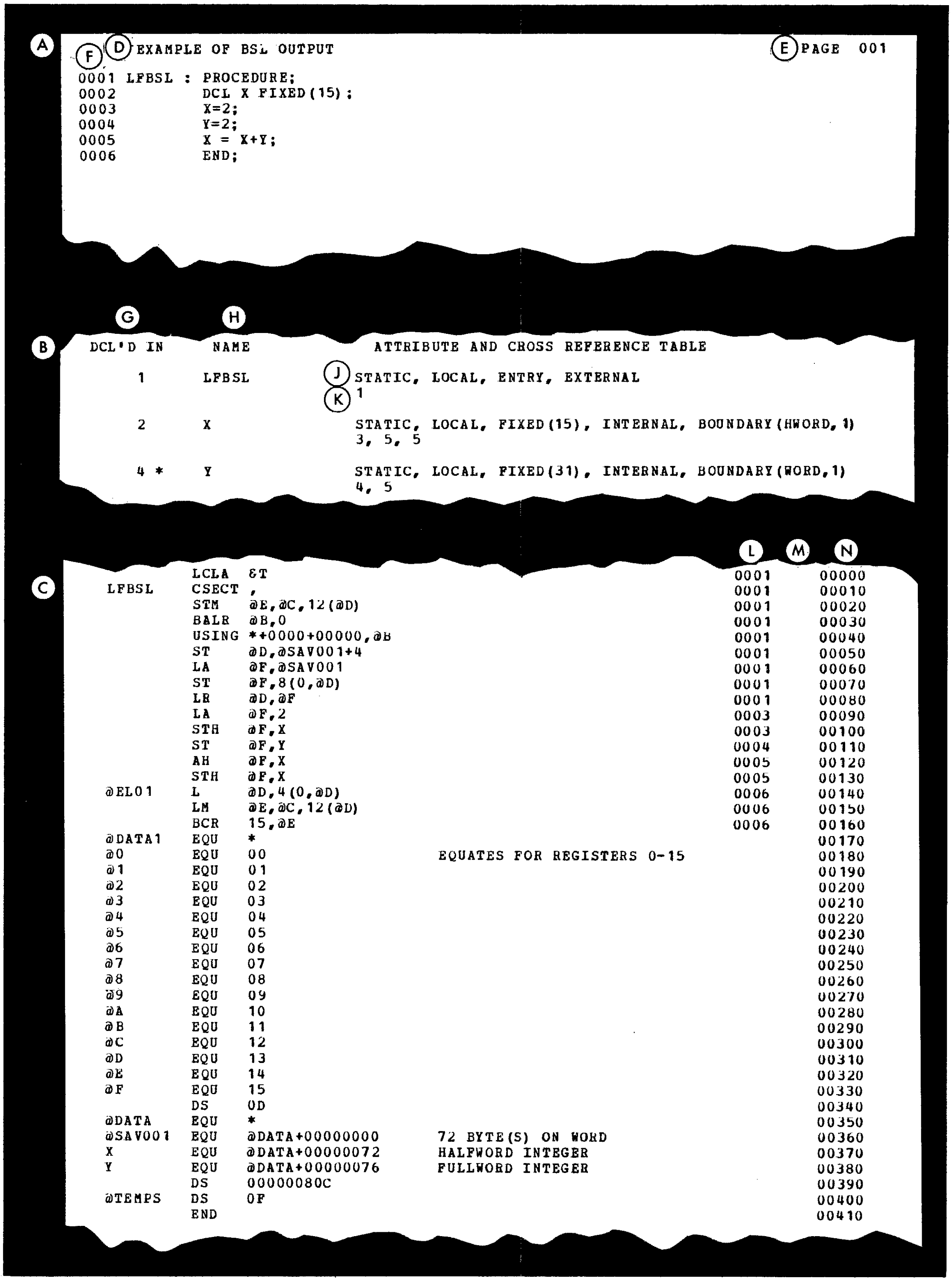
The compiler recognizes the end of a block GENERATE when it encounters a compiler control statement with the word ENDGEN. However, the GENERATE statement (at the beginning of a block GENERATE) is not a compiler control statement. Thus, the format of a block GENERATE is:

```
GENERATE [DATA];
     .
     .
     .
$ENDGEN
```

Delimiter for Batch Compilation

When you use the batch compilation process, you separate the external procedures with a compiler control statement consisting of three dollar signs ($$$). The $$$ statement must follow each external procedure in the batch -- except the last.

The three dollar signs must appear in the first three columns (at the left source margin) of the compiler control statement. If you use the CONCHAR option, all three dollar signs must be replaced by the appropriate control character.


A
F
D
E
B
G
H
J
K
C
L
M
N

```
EXAMPLE OF BSL OUTPUT                                                    PAGE  001
0001 LFBSL : PROCEDURE;
0002         DCL X FIXED(15);
0003         X=2;
0004         Y=2;
0005         X = X+Y;
0006         END;
```

```
DCL'D IN      NAME            ATTRIBUTE AND CROSS REFERENCE TABLE

    1         LFBSL          STATIC, LOCAL, ENTRY, EXTERNAL
                             1

    2         X              STATIC, LOCAL, FIXED(15), INTERNAL, BOUNDARY(HWORD,1)
                             3, 5, 5

    4 *       Y              STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                             4, 5
```

```
              LCLA  &T                                                 0001      00000
    LFBSL     CSECT ,                                                  0001      00010
              STM   @E,@C,12(@D)                                       0001      00020
              BALR  @B,0                                               0001      00030
              USING *+0000+00000,@B                                    0001      00040
              ST    @D,@SAV001+4                                       0001      00050
              LA    @F,@SAV001                                         0001      00060
              ST    @F,8(0,@D)                                         0001      00070
              LR    @D,@F                                              0001      00080
              LA    @F,2                                               0003      00090
              STH   @F,X                                               0003      00100
              ST    @F,Y                                               0004      00110
              AH    @F,X                                               0005      00120
              STH   @F,X                                               0005      00130
    @EL01     L     @D,4(0,@D)                                         0006      00140
              LM    @E,@C,12(@D)                                       0006      00150
              BCR   15,@E                                              0006      00160
    @DATA1    EQU   *                                                            00170
    @0        EQU   00                  EQUATES FOR REGISTERS 0-15               00180
    @1        EQU   01                                                           00190
    @2        EQU   02                                                           00200
    @3        EQU   03                                                           00210
    @4        EQU   04                                                           00220
    @5        EQU   05                                                           00230
    @6        EQU   06                                                           00240
    @7        EQU   07                                                           00250
    @8        EQU   08                                                           00260
    @9        EQU   09                                                           00270
    @A        EQU   10                                                           00280
    @B        EQU   11                                                           00290
    @C        EQU   12                                                           00300
    @D        EQU   13                                                           00310
    @E        EQU   14                                                           00320
    @F        EQU   15                                                           00330
              DS    0D                                                           00340
    @DATA     EQU   *                                                            00350
    @SAV001   EQU   @DATA+00000000      72 BYTE(S) ON WORD                       00360
    X         EQU   @DATA+00000072      HALFWORD INTEGER                         00370
    Y         EQU   @DATA+00000076      FULLWORD INTEGER                         00380
              DS    00000080C                                                    00390
    @TEMPS    DS    0F                                                           00400
              END                                                                00410
```

Figure 2. Example of Compiler Output

Modifying Compiler Output

As its basic output, the compiler lists the BSL source statements, an attribute and cross-reference table, the generated assembler text, and diagnostic messages. Figure 2 shows an example of the output. In the figure, the circled letters point out the various output items, as explained below:

- Item A is a listing of the BSL source statements. Page headings (Item D) are obtained only when you specify the TITLE option. The page numbers (Item E) can be changed or suppressed by the use of options. The BSL statement numbers (Item F) are produced during compilation.

- Item B is a listing of the attribute and cross-reference table. The table contains reference information for each variable in the BSL program. There are three parts to the table:

  1. The first column (Item G) shows the number of the BSL source statement in which the variable was declared. An asterisk following the statement number indicates that the attributes were assigned by default.

  2. The second column (Item H) lists the identifiers in collating sequence. Any referenced but undefined labels are denoted by the letter U which appears to the right of the identifier.

  3. The third column shows the list of attributes (Item J) assigned to the variable. The PARAMETER attribute is given for items which are parameters to procedures or procedure entry points. The third column also shows the number of the statement (Item K) in which the variable is referenced.

- Item C is a listing of the generated assembler text. The reference to BSL statement numbers (Item L) shows which assembler text was generated for each BSL statement. A column is provided (Item M) for an identification name that you can specify with the SEQ option. The assembler text sequence numbers (Item N) can be changed or suppressed by the use of options.

To suppress, supplement, or format the basic compiler output, you can choose from a number of available options. These options which must be specified in compiler control statements, are described in the following paragraphs.

Options to Print or Suppress Source Listings

You can use options to suppress the printing of all or part of the compiler listings. These options are:

NOLIST
suppresses the printing of all compiler listings, except for listings of control statements, time, and error messages.

XREF
causes an attribute and cross-reference table to be listed even if the NOLIST option has been used.

NOXREF
suppresses printing of the attribute and cross-reference table.

MSGLEVEL=n
controls the printing of warning messages. MSGLEVEL=0 is the default option and allows printing of the warning messages. MSGLEVEL=1 suppresses printing of the warning messages.

Options to Format Source Listings

You can use options for heading and paging the listings, and to obtain a printed compilation time. These options are:

TITLE='Title'
: provides a heading for each page of the output listing created by the compiler. If the TITLE option comes before the MACRO option it will also title the macro source listing. The title is printed with the first character printed in the same print positjon as column 1 of the source statement. You specify a character string enclosed in single quotes. With the normal source margins of 1-72, the character string can contain a maximum of 63 characters (including blanks). The other nine columns are required for the statement format $TITLE=".

PAGE=number
: indicates the initial page number to be used on the printed output of the compiler. You specify a number up to four decimal digits, with or without leading zeros. If you do not use this option, the compiler numbers the pages starting from number one.

NOPAGE
: eliminates the page numbers from the output listing created by the compiler.

EJECT
: causes a page eject whenever this option is encountered within the BSL source program or the macro source listing. The word EJECT is not printed on the source listing. This option has no effect on the page format of the generated assembler text listing.

TIME
: indicates that the time required for the BSL compilation is to be printed on the output listing. The compilation time is shown in nimutes and seconds, and represents the total elapsed time from the beginning of compiler execution to its termination.

LINE
: causes the input line number to be used in place of the statement number in the attribute and cross-reference table listing and in the error messages.

Options to Punch or Store Assembler Text

The compiler produces a printed listing of the generated assembler text. You can use options to punch or store this assembler text for use in subsequent operations. These options are:

PUNCH
: indicates that the assembler text is to be punched. If you are compiling under DOS, the assembler text output is directed to the file with the symbolic unit name SYSPCH. If you are compiling under OS, the assembler text output is directed to the data set with the ddname SYSPUNCH. With this option, you get the printed listing of assembler text as well as the punched output.

ASSEM
: indicates that the assembler text is to be written on auxiliary storage (provides input to the assembly operation.) If you are compiling under DOS, the assembler text output is directed to the

file with the symbolic unit name SYS001. If you are compiling under OS, the assembler text output is directed to the data set with the ddname SYSUT2.

NOASSEM
indicates that no assembler output for subsequent use will be generated.

Options to Annotate Assembler Text

The compiler provides a link between the BSL source program and the generated assembler text by means of the BSL statement numbers. Each line of the assembler text contains, as a remark, the number of the BSL statement that caused the code to be generated. You may refer back to Figure 2 (Items F and L) for an example of this relationship.

There are options you can use to add other types of annotation to the assembler text, and to change or suppress the existing annotation. These options are:

NOSNUMBER
suppresses the references to BSL statement numbers in the assembler text.

SEQ=('character string',left col.,right col.)
indicates a character string to be placed in the assembler text beginning at the column number specified in left column and continuing through the column number specified in right column. You may specify up to 19 alpha characters, blanks, and digits in the character string. Any part of the character string may be the starting sequence number. Left column may not be less than 62. The number of columns between left column and right column must equal the length of the character string. SEQ must be followed by an INCR option, on a separate card. An ICTL assembler statement is generated as a result of SEQ. The continuation column becomes the column before the SEQ character string begins.

INCR=('increment value', left col.,right col.)
indicates the increment value for the assembly language sequence number in a SEQ character string. You may specify up to 19 digits. The number of columns between left column and right column must equal the length of the number. The increment value will be added to the number in the SEQ character string starting at left column. Therefore this portion of the SEQ character string must have been a number.

The following example illustrates the SEQ and the INCR options.

```
$SEQ=('APAR01 0000',70,80)
$INCR=('0010',77,80)
```

The character string APAR01 0000 is less than 19 characters long. It will be generated in columns 70 through 80 of the assembler text. In this example the starting sequence number is 0000. It may appear anywhere in the string. When the INCR option is coded, right and left column indicators must reflect the position of the starting sequence number in the string. In this case it would be columns 77 through 80. The INCR option would add 0010 to the sequence number each time a line is printed, making each subsequent line 0010,0020,0030, etc.

RESEQ=('character string',left col.,right col.)

indicates the character string that replaces the character string originally specified in the SEQ option. You may specify up to 19 alpha characters, blanks, and digits in character string. The left column must be equal to or greater than the SEQ left column. The right column must be equal to or less than the SEQ right column. If the RESEQ character string replaces the incremented field with alpha characters, these will be set to zeros before the increment is applied.

The following example illustrates the RESEQ option.

```
$RESEQ=('23',74,75)
```

This option would cause the data in columns 74 and 75 to be replaced by the number 23. If applied to the example for SEQ and INCR the result would be APAR23 in columns 70 through 75 of the assembler text.

NOSEQ

suppresses the generation of assembler text sequence numbers.

INTER

causes the listing of the BSL source program to be merged with the listing of the assembler text. Each BSL source statement is followed by the assembler text that it generated. This combined listing replaces both the BSL source statement listing and the assembler text listing that are usually produced by the compiler. (An alternate method of obtaining an intermixed listing is explained under ANNOTATE.)

ANNOTATE=x

allows you to add comments to the generated assembler text. The comments are derived from the BSL source program. The codes you can specify are:

ANNOTATE=0 -- the portion of the BSL input record between the established source margins is included as a comment in the generated assembler text.

ANNOTATE=1 -- the entire BSL input record is included as a comment in the generated assembler text. Source margins specified by the SORMGIN option are ignored.

ANNOTATE=2 -- only the <u>comments</u> from the BSL input records are included as comments in the generated assembler text. These comments are placed in the assembler text listing in the same position as they appeared in the BSL source program. (Actually, the comments are moved one column to the right to allow for the assembler text asterisk.) If the source comment appeared within a BSL source statement (preceding the semicolon), it is placed before the assembler text generated for that statement. If the source comment appeared after a BSL source statement (following the semicolon), it is placed after the assembler text generated for that statement.

<u>Note</u>: The contents of columns 71-80 of the BSL input records are never reproduced in the assembler text by the ANNOTATE option.

Using the Object Program Tracing Facility

The compiler's TRACE option is designed to help you debug your BSL program. The TRACE option generates extra assembler code at pertinent points in your program so that entries and branches can be traced. The program trace is listed at execution time. An example of this output is illustrated in Appendix B.

Trace Control Statements

There are three compiler control statements associated with the tracing facility. The TRACE statement appears first and establishes use of the option. The TRACE ON and TRACE OFF statements indicate starting and stopping points for the trace operation. Specifically, the requirements for the control statements are:

TRACE [OFFSET=number]
indicates use of the option for the compilation. This statement must precede the BSL source program. For the OFFSET value (optional), you specify a number up to 4095 which indicates a displacement (in bytes) from the save area address contained in register 13. The default offset value is 2.

The TRACE option always assumes that register 13 contains the address of a save area. At this location, plus the offset, the tracing operation saves the statement number of each statement before it is executed. If you specify an offset greater than 71 bytes, a larger save area is generated to meet the requirement. If the procedure option NOSAVEAREA is specified, tracing is still performed using the contents of register 13. (The main BSL procedure should not specify NOSAVEAREA if the TRACE option is to be used. The results are unpredictable.)

The compiler options TRACE and NOSNUMBER are mutually exclusive. If both are specified, the compiler uses only the first one it encounters.

TRACE ON
indicates a point at which program tracing is to start. TRACE ON statements can appear anywhere in the input records provided that the TRACE statement has been specified previously.

TRACE OFF
is used in conjunction with the TRACE ON statement, and indicates the point at which program tracing is to stop. TRACE OFF statements can appear anywhere within the BSL source program provided that the TRACE statement has been specified previously.

TRACE Assembler Code

The extra assembler code generated by the compiler sets up linkage to the object-time tracing routines, and defines the constants that are needed for tracing. The tracing code precedes the regular code generated for the corresponding statements. After testing and debugging is completed, you eliminate the extra code by recompiling your program without TRACE control statements. (You may need to provide additional addressability to handle the extra code generated for tracing purposes. This addressability may be removed when the TRACE option is removed.)

When the TRACE option is in effect, the extra code generated for each point in the program is as follows:

Entry Points: If TRACE ON is not specified, the tracing code generated at entry points is:

```
L     15,@BSLTRCI
BALR  14,15
DC    H'offset value'
```

If TRACE ON is specified, the tracing code generated at entry points is:

```
L     15,@BSLTRCE
BALR  14,15
DC    CL8'entry name'
DC    H'statement number'
DC    H'offset value'
```

If CODEREG(0) is specified, you are responsible for generating the trace initialization code at the entry points. Follow the appropriate example shown above, depending on whether or not TRACE ON is in effect.

Labeled Statements: If TRACE ON is specified, the tracing code generated at labeled statements is:

```
BAL   14,@BSLTRCB
DC    H'statement number'
```

Unlabeled Statements: If TRACE ON is specified, the tracing code generated at unlabeled statements is:

```
MVI   offset(13),first part of statement number
MVI   offset+1(13),second part of statement number
```

If there is no change to the first part of the statement number, the first MVI instruction is omitted.

CALL Statements: If TRACE ON is specified, the tracing code generated at CALL statement is:

```
L     @F,@BSLTRCC
BALR  @E,@F
DC    H'statement number'
DC    CL8'entry name'
```

RETURN Statements: If TRACE ON is specified, the tracing code generated at RETURN statement is:

```
BAL   14,@BSLTRCB
DC    H'statement number'
L     15,@BSLTRCR
BALR  14,15
DC    CL8'return point'
```

If a return code must be passed when TRACE ON is specified, the return code must be placed in the register 15 field of the save area rather than in register 15.

Data Area: Whether or not TRACE ON is specified, the tracing code generated in the program's data area is:

```
@BSLTRCC  DC   V(IKETRCC)
@BSLTRCR  DC   V(IKETRCR)
@BSLTRCI  DC   V(IKETRCI)
@BSLTRCE  DC   V(IKETRCE)
@BSLTRCL  DC   V(IKETRCL)
@BSLTRCB  L    15,@BSLTRCL
          BCR  15,15
```

TRACE Routines and Output

The tracing routines are included in the BSL object-time library. These routines initialize and perform tracing operations when your program is executed. If you are executing under DOS, the tracing output is directed to the file with the symbolic unit name SYSLST. If you are executing under OS, the tracing output is directed to the data set with the ddname BSLOUT.

The five tracing routines and their related output are:

1. The IKETRCE routine initializes tracing and produces the following output:

   ENTERING entry name, SAVE AREA AT savearea address.

2. When TRACE ON is not specified, the IKETRCI routine performs tracing initialization at entry to a BSL procedure.

3. The IKETRCL routine produces the following output when the program executes a branch to a labeled statement:

   TO statement number FROM old statement number.

4. The IKETRCC routine produces the following output when a CALL statement is encountered:

   CALLING entry name FROM STATEMENT statement number.

5. The IKETRCR routine produces the following output when a RETURN TO statement is encountered:

   RETURNING TO return name FROM statement number. RETURN CODE IS register 15 value.

   and the following output when a RETURN statement is encountered:

   RETURNING TO -CALLER- FROM statement number. RETURN CODE IS register 15 value.

Your program can dynamically suppress the tracing output by setting a switch in the tracing routine. This switch is a variable named IKETRCOF which is declared in the tracing routine as FIXED(31) LOCAL EXTERNAL INIT(0). To use the switch, your program must declare the same variable as NONLOCAL EXTERNAL.

The IKETRCOF switch is initially set to 0, which allows normal tracing output. If your program sets the switch to 1, no tracing information is printed until the switch is reset to 0. For example, if you only want to trace the last time through a loop, your program might specify:

```
DCL   IKETRCOF EXTERNAL;
 .
 .
 .
IKETRCOF = 1;
DO I = 1 TO 100;
IF I = 100 THEN IKETRCOF = 0;
 .
 .
 .
END;
```

Section 3: BSL Under the Disk Operating System (DOS)

This section contains the information you need to compile, assemble, linkage edit, and execute your BSL programs under the IBM System/360 Disk Operating System (DOS).

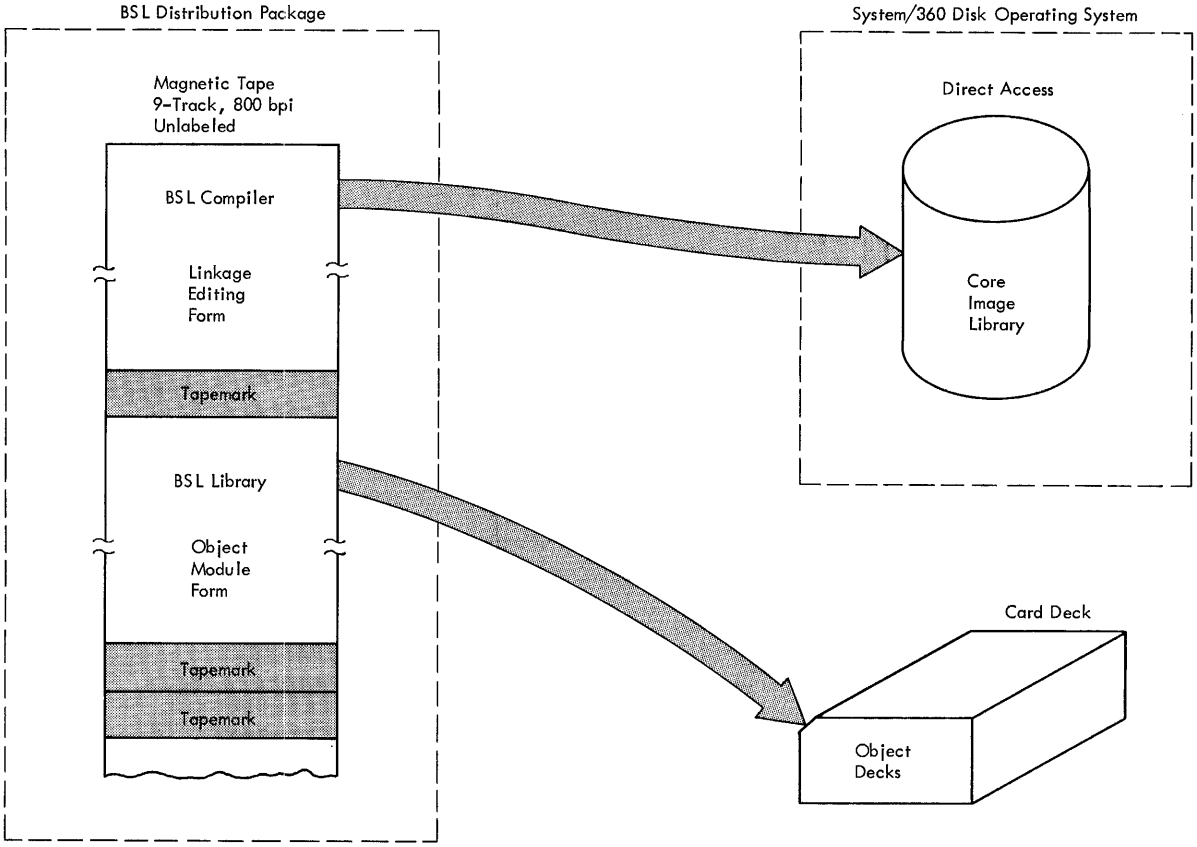
Adding BSL to the System

The BSL compiler runs as a problem program under DOS. It requires a machine configuration with 128K bytes of main storage.

Figure 3 shows the BSL distribution package and how it is incorporated into the system. For DOS users, the BSL distribution package is provided on a magnetic tape volume which is 9-track, 800 bpi, and unlabeled. The tape contains two files:

File 1 -- The BSL compiler in a form suitable for linkage editing.

File 2 -- The BSL library containing object-time program in object module form.



Notes:

- To put the BSL compiler into the core image library, use the CATAL linkage editor option.
- To punch the BSL library, use the TPCD utility program.

Figure 3. Adding BSL to the Disk Operating System

To linkage edit the BSL compiler and add it to the core image library, you may use the job control statements shown in the following example. The example assumes that the distribution tape is mounted on unit 284. (The operator should type 'ignore' when intervention is required on unit 284.) The job control statements are:

```
//   JOB LNKBSL
//   ASSGN SYSIPT,X'284'
//   OPTION LINK,CATAL
//   UPSI 10000000
     PHASE BSL,*
     INCLUDE
     ENTRY BSLCTL
//   EXEC LNKEDT
/&
```

To have the programs of the BSL library available at execution time, you should punch them as object decks. An example of the job control statements for the punch operation is given below. The example assumes that the distribution tape is mounted on unit 284. Specify:

```
//   JOB PNCHBSL
//   ASSGN SYS006,X'00D'
//   ASSGN SYS004,X'284'
//   MTC FSF,SYS004,01
//   UPSI 10100000
//   EXEC TPCD
//   UTC TR,FF,A=(80,1600),B=(80,80),IN,S1,O1,R1
//   END
/&
```

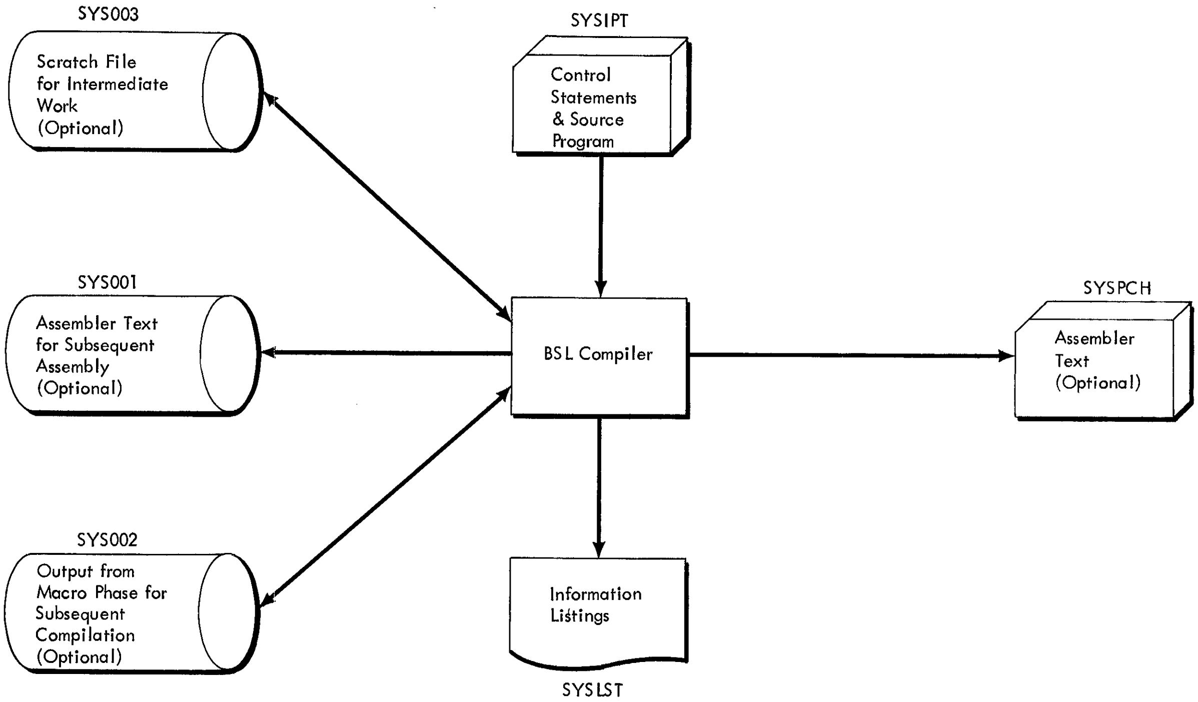
Using the BSL Compiler

The smallest self-contained unit of input to the compiler is a BSL external procedure. The output of the compiler is assembler text. The compiler output must subsequently be assembled before your program can be loaded and executed.

The assembler text output of the compiler does not depend on any system macro instructions; therefore, it can be assembled, loaded, and executed under any System/360 programming system. However, if you include system macro instructions in the BSL program (by means of the GENERATE statement or the REENTRANT attribute), you must use the related system's assembler to process the compiler's output.

Files Required for Compilation

The files used in the compilation process are illustrated in Figure 4, and described in Table 2. The Compiler's input/output processing is independent of the device type used, except for the scratch file (SYS003) and the intermediate files (SYS001 and SYS002). The scratch and intermediate files require magnetic tape devices.


SYS003
Scratch File for Intermediate Work (Optional)
SYSIPT
Control Statements & Source Program
SYS001
Assembler Text for Subsequent Assembly (Optional)
BSL Compiler
SYSPCH
Assembler Text (Optional)
SYS002
Output from Macro Phase for Subsequent Compilation (Optional)
Information Listings
SYSLST

• Figure 4. Flowchart of DOS Files Used by the Compiler

• Table 2. Descripion of DOS Files Used By the Compiler

Symbolic Unit Names	Function	Device Types Permitted	Blocking
SYSIPT	Provides input to the compiler; i.e., control statements and source program.	Card Reader Direct Access Tape	Unblocked
SYSLST	Used to write out information listings of source program, assembler text, diagnostic messages, etc.	Printer Direct Access Tape	Unblocked
SYS003	Scratch file for intermediate work. Used only to save data created by GENERATE DATA statement.	Tape Direct Access	Unblocked
SYS001	Used to write out generated assembler text for subsequent assembly. If the NOASSEM option is specified, this file is not required.	Tape Direct Access	Unblocked
SYS002	The macro phase writes the modified source text on this file for subsequent use in the compile phase. If the MACRO option is not specified, this file is not required.	Tape Direct Access	Unblocked
SYSPCH	The PUNCH option provides this output of the generated assembler text. If the PUNCH option is not specified, this file is not required.	Card Punch Direct Access Tape	Unblocked

Compilation Without Assembly

The compile-only operation allows you to compile a batch (one or more) of BSL external procedures without the production of an output file of assembler text. You should not specify the ASSEM option for compile-only operations. If you specify the PUNCH option, the compiler produces a punched deck containing the generated assembler text. This punched output may be subsequently used as input to an assembler.

To compile your source programs, you execute the BSL compiler in a DOS job step. An example of the job control statements needed to compile a single BSL external procedure is:

```
//  JOB BSL
//  ASSGN SYS003,X'282'   (only if program uses GENERATE DATA)
//  ASSGN SYS002,X'284'   (only for MACRO option)
//  EXEC BSL
  Compiler Control Statements (include the NOASSEM option)
        .
        .
  BSL External Procedure
        .
        .
        .
/*
//  RESET SYS003   (only if program uses GENERATE DATA)
//  RESET SYS002   (only for MACRO option)
/&
```

For batch compilation, you use the $$$ delimiter to separate the BSL external procedures in the compiler input. The last BSL external procedure is followed by the /* delimiter. An example of the job control statements needed to compile a batch of BSL external procedures is:

```
//  JOB BSL
//  ASSGN SYS003,X'282'   (only if program uses GENERATE DATA)
//  ASSGN SYS002,X'284'   (only for MACRO option)
//  EXEC BSL
  Compiler Control Statements  (include the NOASSEM option)
        .
  First BSL External Procedure
        .
        .
$$$
  Compiler Control Statements  (include the NOASSEM option)
        .
  Next BSL External Procedure
        .
        .
$$$
  Compiler Control Statements  (include the NOASSEM option)
        .
  Last BSL External Procedure
        .
        .
/*
//  RESET SYS003   (only if program uses GENERATE DATA)
//  RESET SYS002   (only for MACRO option)
/&
```

Compilation and Assembly

If you are going to assemble the compiler's output with a DOS assembler, you can accomplish both compilation and subsequent assembly by executing two DOS job steps. This method permits compilation and assembly of one BSL external procedure. Batched compilation with subsequent assembly is not possible because the assembler cannot handle the multiple END statements generated by the compiler.

An example of the job control statements needed to compile and assemble one BSL external procedure is:

```
// JOB BSL
// ASSGN SYS001,X'283'
// ASSGN SYS002,X'284'    (only for MACRO option)
// EXEC BSL
  Compiler Control Statements
        .
        .
  BSL External Procedure
        .
        .
        .
/*
// RESET SYS001
// RESET SYS002   (only for MACRO option)
// ASSGN SYSIPT,X'283'
// EXEC ASSEMBLY
// RESET SYSIPT
/&
```

You can compile and assemble several BSL external procedures by repeating the job steps shown above.

Compilation, Assembly, and Execution

Since BSL is a language for the writing of systems programs, the compiler makes no assumptions about the environment in which the resulting object program will be executed. The environment you select for program execution depends on the type of program that you write. You can run the program in a stand-alone mode or under some operating system. You can use any execution-time facilities that are available in the environment, such as control program services or special machine instructions.

If you are going to execute the program under DOS, the compilation, assembly, linkage edit, and execution may each be a separate step of a single job. In this job, you can compile and assemble a number of BSL external procedures, and then linkage edit and execute them as one load module. Additional object decks can be included in the linkage edit step. An example of the job control statements needed for this type of job is:

```
// JOB RUNBSL
// OPTION LINK
// ASSGN SYS003,X'281'   (only if program uses GENERATE DATA.)
// ASSGN SYS001,X'283'
// ASSGN SYS002,X'284'   (only for MACRO option)
// EXEC BSL
  Compiler Control Statements
     .
  First BSL External Procedure
     .
     .
/*
// RESET SYS003   (only if program uses GENERATE DATA.)
// RESET SYS001
// RESET SYS002   (only for MACRO option)
// ASSGN SYSIPT,X'283'
// EXEC ASSEMBLY
// RESET SYSIPT                                   )
// ASSGN SYS003,X'281' (only if program uses      |
                       GENERATE DATA.)            |
// ASSGN SYS001,X'283'                            |
// ASSGN SYS002,X'284' (only for MACRO            |
                       option)                    |
// EXEC BSL                                       | Repeat these statements
  Compiler Control Statements                     | for each additional BSL
     .                                            } procedure, or delete to
  Next BSL External Procedure                     | compile only one BSL
     .                                            | procedure.
     .                                            |
/*                                                |
// RESET SYS003   (only if program uses           |
                  GENERATE DATA.)                 |
// RESET SYS001                                   |
// RESET SYS002   (only for MACRO option)         |
// ASSGN SYSIPT,X'283'                            )
// EXEC ASSEMBLY
// RESET SYSIPT
   INCLUDE                                        } Delete if object decks
  Object Decks                                    } are not required.
/*
   ENTRY Name of procedure to be entered
// EXEC LNKEDT
// EXEC
  Program Input (if any)
/&
```

BSL source programs written for execution under DOS should comply with the following special requirements:

- If you specify the REENTRANT procedure option, and DATAREG is not zero, the compiler generates the OS GETMAIN macro instruction. If you want reentrant code for execution under DOS, you must specify DATAREG(0) and provide the storage and addressability for the AUTOMATIC data.

- The BSL program that will receive control from DOS should use the DONTSAVE procedure option.

- To return control to DOS, the EOJ macro instruction should be specified in a GENERATE statement at the logical end of the BSL program.

The following example illustrates all of the above requirements:

```
DOSMAIN:  PROC OPTIONS (REENTRANT,DATAREG(0),DONTSAVE);
             .
             .
          GEN (EOJ);
          END;
```

Using Control Program Services and Special Machine Instructions

The GENERATE statement allows you to use any control program service or special machine instruction in your BSL program. In the case of control program services or other pre-defined macro instructions, you are responsible for ensuring that the required macro expansions are available at assembly time.

The BSL compiler never inspects the contents of a GENERATE statement. If you use the GENERATE statement, you must ensure that the contents are legal and properly related to the compiler-generated assembler text.

Other considerations for using GENERATE are explained under "BSL Programming Techniques" in Section 7 of this manual. The form of the statement is described in the *BSL Language Specification* manual.

Using the BSL Library Routines

When executing a BSL program under DOS, you may use the service routines provided in the BSL library. These routines perform certain basic functions that are useful for testing your programs. To call a library routine, you must specify the entry name in a BSL CALL statement and then include the object deck in the linkage editing process.

Complete descriptions and examples of the available routines can be found in the BSL Library publication. For your reference, a summary of the functions is listed in Table 3. (The BSL library also contains the TRACE routines which are described in Section 2.)

The facilities provided by the library routines should not be regarded as part of the BSL language. The output of the BSL compiler is independent of any operating environment, but the library routines function only under DOS. (Similar routines with identical entry names are provided in the compiler distribution package for OS.)

Table 3. Functions of the BSL Library Routines (DOS)

Routine Names	Entry Points	Functions
EDIT	GET GETS PUT PUTS TABSET	Performs reading, writing, and conversion of data in a manner similar to PL/I EDIT DIRECTED I/O.
PDUMP	PDUMP	Performs snapshot dumps of specific data areas. The dumps are provided in hexadecimal, character, or bit string formats.
SUBSTR	SUBSTR	Performs character substring assignment while checking substring ranges.
BSL I/O	READ PRINT PUNCH CLOSE	Employs the DOS sequential access method to provide basic input/output services at object time. These routines can read an 80-byte record from SYSRDR, write a 121-byte record on SYSLST, write an 80-byte record on SYSPCH, and close SYSLST.
Interrupt Handler	IKETRCII IKETRCID	Provides statement number and save area trace when a program check occurs.

Including Text From a Library

When you compile under DOS, you can use the compiler's INCLUDE features:

- The INCLUDE macro statement (%INCLUDE) incorporates text during compile-time macro processing.
- The INCLUDE option ($INCLUDE) incorporates text after macro activity and does not submit the text to scanning for macro substitution.

These features allow you to keep certain standard declarations or sections of source code in external libraries. As needed, these items can be obtained from the libraries during compilation, and included in your programs.

Storing Text in External Libraries

The section of code that is to be kept for inclusion in source programs must be a book in a source statement library. The source statement library may be a private library if SYSSLB is assigned.

You can use the source statement library maintenance and service programs to add, delete, rename, condense or update the books. The following example adds a book to a private library. If SYSSLB is not defined, the book will be added to the system source statement library. Example:

```
//            JOB     jobname
//            OPTION  PARSTD
//            ASSGN   SYSSLB,X'cnn'
//            DLBL    IJSYSSL,'user identification of private        X
                      library', date,code
//            EXTENT  SYSSLB,extent information
//            EXEC    MAINT
              CATALS  sublib.bookname[,v.m[,c]]
              BKEND
              (BSL source statements)
              BKEND
/*
/&
```

Specifying INCLUDE Option of Macro Statement

The INCLUDE option or macro statement may appear at any point (and any number of times) in the compiler input. When INCLUDE is specified, the contents of the book in the source statement library is read as input to the compiler. This included text may not contain an INCLUDE.

You specify INCLUDE in the following format:

```
$INCLUDE    sublibrary(bookname)
        or
%INCLUDE    sublibrary(bookname)
```

where:

{$INCLUDE
%INCLUDE}
: specifies the type of INCLUDE. The dollar sign ($) identifies the INCLUDE option, and must appear in the first column (leftmost source margin). The word INCLUDE must immediately follow (no space) the dollar sign, and cannot be combined with other options in the same compiler control statement.

The percent sign (%) identifies the INCLUDE macro statement, as described in the <u>BSL Language Specification</u> manual.

sublibrary
: indentifies the sublibrary to which the book belongs. This can be an alpha character (0-9,A-Z,#,$, and @).

(bookname)
: represents the name of the book in the sublibrary. The name is one to eight alphanumeric characters, the first of which must be alphabetic. If the named book is not in the sublibrary, the compilation is terminated after printing a D13 diagnostic message.

The following example shows the use of the INCLUDE option. In the example, text is obtained from book DCL01 in sublibrary B of a private source statement library. Example:

```
//           JOB     BSLCOMP
//           OPTION  PARSTD
//           ASSGN   SYSSLB,X'191'
//           DLBL    IJYSSLB,'library identification',date,code
//           EXTENT  SYSSLB,extent information
//           EXEC    BSL
$TIME
PROGM1:  PROC;
         DCL TABLE(256)CHAR(10);
$INCLUDE B(DCL01)
             CALL X;
X:  PROC;
         DO I=1 BY 1 TO 256;
         TABLE(I,1:4)=CODE(I);
         END;
         END X;
         END PROGM1;
/*
/&
```

Section 4: BSL Under the Operating System (OS)

This section contains the information you need to compile, assemble, linkage edit, and execute your BSL programs under the IBM System/360 Operating System (OS).

Adding BSL to the System

The BSL compiler runs as a problem program under OS. It requires 100,000 bytes of main storage in addition to the storage required by the operating system, and the compiler's dictionary (see SIZE option) as well as the input and output buffers.

Figure 5 shows how BSL is incorporated into the system. For OS users, the BSL distribution package is provided on a magnetic tape volume (9-track, 800 bpi) with standard labels. Three suggested job control procedures are provided later in the section.

Creating a Private Library

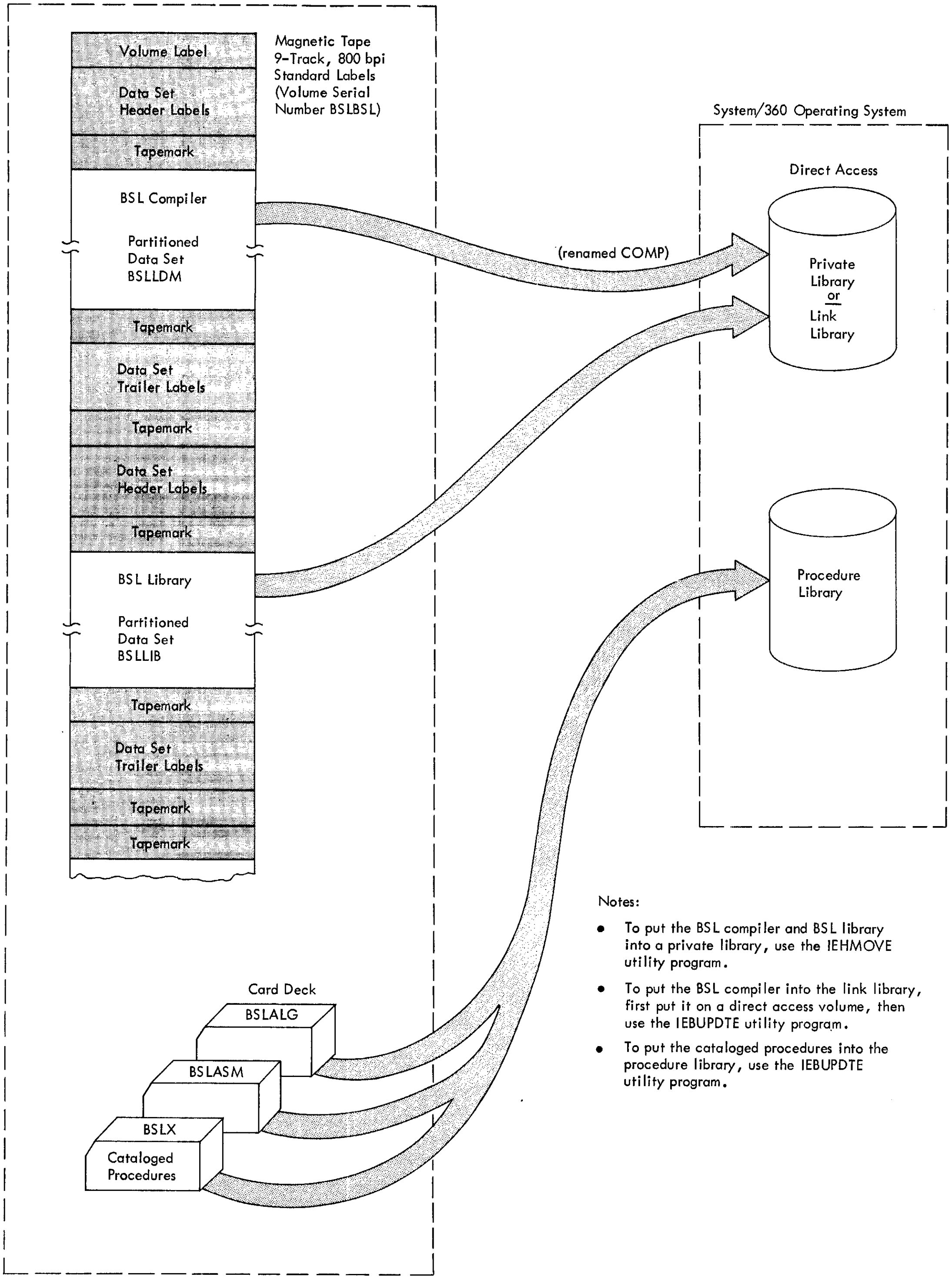
The volume serial number of the distribution tape is BSLBSL. The tape contains two partitioned data sets:

BSLLDM
is the BSL compiler in the form of a partitioned data set. The data set name is BSLLDM. The data set has only one member, and the member name is BSL.

BSLLIB
is the BSL library in the form of a partitioned data set. The data set name is BSLLIB. Its members are a number of object-time service programs.

The examples of job control statements in this section of the manual are based on the assumption that you will create private libraries that contain the BSL compiler and BSL library. At compilation time, the private library that contains the BSL compiler is concatenated to the link library (SYS1.LINKLIB) by means of a JOBLIB DD statement. Optionally, you can add the BSL compiler to the link library, and omit the JOBLIB DD statement at compilation time.

You can use the IEHMOVE system utility program to move the partitioned data sets from the distribution tape to the private library volume (direct access). If these partitioned data sets already exist on your volume, they must be scratched first.



Notes:

- To put the BSL compiler and BSL library into a private library, use the IEHMOVE utility program.
- To put the BSL compiler into the link library, first put it on a direct access volume, then use the IEBUPDTE utility program.
- To put the cataloged procedures into the procedure library, use the IEBUPDTE utility program.

• Figure 5. Adding BSL to the Operating System

An example of the job control statements needed to create the private libraries is shown below. The example assumes that the private libraries are placed on a 2311 disk with the volume serial number 111111. Also, the data set BSLLDM (the BSL compiler) is renamed COMP because that is the name usually specified in the JOBLIB DD statement. The job control statements are:

```
//A           JOB
//B           EXEC  PGM=IEHMOVE
//SYSPRINT    DD    SYSOUT=A
//DD1         DD    VOLUME=SER=BSLBSL,UNIT=2400,DISP=OLD
//DD2         DD    DSNAME=COMP,VOLUME=SER=111111,UNIT=2311,          X
//                  SPACE=(TRK,(30,10,9)),DISP=(NEW,KEEP)
//SYSUT1      DD    VOLUME=SER=111111,UNIT=2311                       X
//                  SPACE=(TRK,(30,10)),DISP=(NEW,DELETE)
//SYSIN       DD    *
              COPY  PDS=BSLLDM,TO=2311=111111,                        X
                    FROM=2400=(BSLBSL,1),RENAME=COMP
/*
//BB          EXEC  PGM=IEHMOVE
//SYSPRINT    DD    SYSOUT=A
//DD1         DD    VOLUME=SER=BSLBSL,UNIT=2400,DISP=OLD
//DD2         DD    DSNAME=BSLLIB,VOLUME=SER=111111,UNIT=2311,        X
//                  SPACE=(CYL,(4,2,8)),DISP=(NEW,KEEP)
//SYSUT1      DD    VOLUME=SER=111111,UNIT=2311,                      X
//                  SPACE=(TRK,(30,10)),DISP=(NEW,DELETE)
//SYSIN       DD    *
              COPY  PDS=BSLLIB,TO=2311=111111,FROM=2400=(BSLBSL,2)
/*
```

Cataloging the Job Control Procedures

A set of job control procedures is provided in this section. The set contains three procedures (BSLX, BSLASM, and BSLALG) which can be used as cataloged procedures for BSL operations under OS.

If you intend to use cataloged procedures to run the BSL operations, you must add the suggested procedures (or your own procedures) to the procedure library (SYS1.PROCLIB).

An example of the job control statements you need to catalog the procedures is shown below. The example uses the IEBUPDTE utility program, and assumes that the procedure library is on a 2311 disk with the volume serial number 111111. The job control statements are:

```
//            JOB
//            EXEC  PGM=IEBUPDTE,PARM=MOD
//SYSPRINT    DD    SYSOUT=A
//SYSUT1      DD    DSNAME=SYS1.PROCLIB,VOLUME=SER=111111,            X
//                  UNIT=2311,DISP=(OLD,KEEP),                        X
//                  DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2      DD    DSNAME=SYS1.PROCLIB,VOLUME=SER=111111,            X
//                  UNIT=2311,DISP=(OLD,KEEP),
//                  DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSIN       DD    DATA
./            ADD   LIST=ALL,NAME=BSLALG,LEVEL=01,SOURCE=0
     Card Deck (containing BSLALG procedure)
         .
         .
./            ADD LIST=ALL,NAME=BSLASM,LEVEL=01,SOURCE=0
     Card Deck (containing BSLASM procedure)
         .
         .
./            ADD LIST=ALL,NAME=BSLX,LEVEL=01,SOURCE=0
     Card Deck (containing BSLX procedure)
         .
/*
```

Using the BSL Compiler

The smallest self-contained unit of input to the compiler is a BSL external procedure. The output of the compiler is assembler text. The compiler output must subsequently be assembled before your program can be loaded and executed.

The assembler text output of the compiler does not depend on any system macro instructions; therefore, it can be assembled, loaded, and executed under any IBM System/360 programming system. However, if you include system macro instructions in the BSL source program (by means of the GENERATE statement or the REENTRANT attribute), you must use the related system's assembler to process the compiler's output.

Data Sets Required for Compilation

The data sets used in the compilation process are illustrated in Figure 6, and described in Table 4. As shown in the table, the compiler's input/output processing is independent of the device type used.

A column in Table 4 describes the DCB attributes for each of the compiler's data sets, and shows which attributes are subject to change. The values shown for the changeable attributes are those that the compiler assigns by default. To establish different values, you specify the desired attributes in the DCB parameter of the applicable DD statement. For example, if the SYSIN data set is on tape with blocked records, the DD statement could be specified as:

```
//SYSIN    DD    DSNAME=SOURCE1,UNIT=183,DISP=OLD,VOLUME=SER=A11111,     X
                 LABEL=(,NL),DCB=(RECFM=FB,BLKSIZE=640)
```

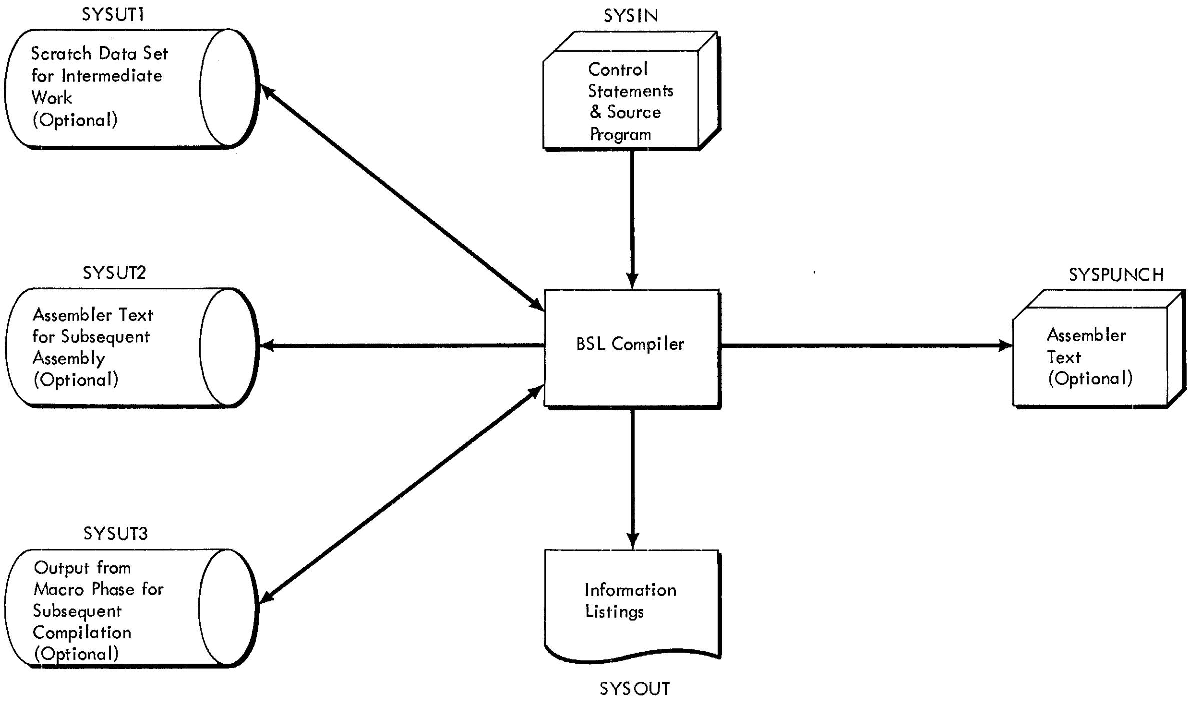

SYSUT1
Scratch Data Set for Intermediate Work (Optional)
SYSIN
Control Statements & Source Program
SYSUT2
Assembler Text for Subsequent Assembly (Optional)
BSL Compiler
SYSPUNCH
Assembler Text (Optional)
SYSUT3
Output from Macro Phase for Subsequent Compilation (Optional)
Information Listings
SYSOUT

Figure 6. Flowchart of OS Data Sets Used by the Compiler

Table 4. Description of OS Data Sets Used By the Compiler

ddname	Function	Device Types Permitted	DCB Specifications Used By Compiler

SYSIN	Provides input to the compiler; i.e., control statements and source program.	Card Reader[1]
		Direct Access
		Tape
			RECFM=F[2]
			LRECL=80
			BLKSIZE=80[2]
			BUFNO=3[2]

SYSOUT	Used to write out information listings of source program, assembler text, diagnostic messages, etc.	Printer[1]
		Direct Access
		Tape
			RECFM=FA[2]
			LRECL=121
			BLKSIZE=121[2]
			BUFNO=3[2]

SYSUT1	Scratch data set for intermediate work. Used only to save data created by GENERATE DATA statement.	Direct Access[1]
		Tape
			RECFM=FB[2]
			LRECL=80
			BLKSIZE=3520[2]
			BUFNO=2[2]

SYSUT2	Used to write out generated assembler text for subsequent assembly. If the NOASSEM option is specified, this data set is not required.	Direct Access[1]
		Tape
			RECFM=FB[2]
			LRECL=80
			BLKSIZE=800[2]
			BUFNO=2[2]

SYSUT3	The macro phase writes the modified source text in this data set, for subsequent use in the compile phase. If the MACRO option is not specified, this data set is not required.	Direct Access[1]
		Tape
			RECFM=FB[2]
			LRECL=80
			BLKSIZE=3520[2]
			BUFNO=2

SYSPUNCH	The PUNCH option provides this output of the generated assembler text. If the PUNCH option is not specified, this data set is not required.	Card Punch[1]
		Direct Access
		Tape
			RECFM=F[2]
			LRECL=80
			BLKSIZE=80[2]
			BUFNO=2[2]

[1] These are the device types specified in the cataloged procedures.

[2] You can specify other values for these DCB parameters.

BSL Cataloged Procedures

Three cataloged procedures are provided for BSL operations under OS. You can:

- Use the cataloged procedures suggested in this section.
- Write and use your own cataloged procedures.
- Execute BSL without cataloged procedures, by providing the full set of job control statements at compilation time.

If you create your own procedures, you can use the suggested procedures as examples of format and statement requirements. Your procedures must conform to the data set requirements described previously.

Three suggested cataloged procedures are listed in Figures 7, 8, and 9. The procedure names and functions are:

BSLX
    contains the job control statements for the compile-only operation.

BSLASM
    contains the job control statements for compilation and assembly.

BSLALG
    contains the job control statements for compilation, assembly, linkage editing, and execution.

```
//STEP1       EXEC   PGM=BSL
//SYSOUT      DD     SYSOUT=A
//SYSUT1      DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(2,1))
//SYSUT2      DD     DSNAME=&BSLGO,UNIT=SYSDA,DISP=(NEW,DELETE),         X
//                   SPACE=(CYL,(3,1))
//SYSUT3      DD     DSNAME=&UT3,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(3,1))
//SYSPUNCH    DD     UNIT=SYSCP
```

• Figure 7. Cataloged Procedure BSLX

```
//STEP1       EXEC   PGM=BSL
//SYSOUT      DD     SYSOUT=A
//SYSUT1      DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(2,1))
//SYSUT2      DD     DSNAME=&BSLGO,UNIT=SYSDA,DISP=(NEW,PASS),           X
//                   SPACE=(CYL,(3,1))
//SYSUT3      DD     DSNAME=&UT3,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(3,1))
//SYSPUNCH    DD     UNIT=SYSCP


//STEP2       EXEC   PGM=ASMBLR,COND=(9,LT,STEP1),PARM=LOAD
//SYSGO       DD     DSNAME=&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),         X
//                   SPACE=(80,(200,50))
//SYSPUNCH    DD     UNIT=SYSCP
//SYSPRINT    DD     SYSOUT=A
//SYSUT1      DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(3,1))
//SYSUT2      DD     DSNAME=&UT2,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(3,1))
//SYSUT3      DD     DSNAME=&UT3,UNIT=SYSDA,DISP=(NEW,DELETE),           X
//                   SPACE=(CYL,(3,1))
//SYSIN       DD     DSNAME=*.STEP1.SYSUT2,UNIT=SYSDA,                   X
//                   DISP=(OLD,DELETE)
```

• Figure 8. Cataloged Procedure BSLASM

```
//STEP1        EXEC   PGM=BSL
//SYSOUT       DD     SYSOUT=A
//SYSUT1       DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(2,1))
//SYSUT2       DD     DSNAME=&BSLGO,UNIT=SYSDA,DISP=(NEW,PASS),             X
//                    SPACE=(CYL,(3,1))
//SYSUT3       DD     DSNAME=&UT3,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(3,1))
//SYSPUNCH     DD     UNIT=SYSCP


//STEP2        EXEC   PGM=ASMBLR,COND=(9,LT,STEP1),PARM=LOAD
//SYSGO        DD     DSNAME=&LOADSET,UNIT=SYSDA,DISP=(MOD,PASS),           X
//                    SPACE=(80,(200,50))
//SYSPUNCH     DD     UNIT=SYSCP
//SYSPRINT     DD     SYSOUT=A
//SYSUT1       DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(3,1))
//SYSUT2       DD     DSNAME=&UT2,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(3,1))
//SYSUT3       DD     DSNAME=&UT3,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(3,1))
//SYSIN        DD     DSNAME=*.STEP1.SYSUT2,UNIT=SYSDA,                     X
//                    DISP=(OLD,DELETE)


//LKED         EXEC   PGM=LINKEDIT,COND=(4,LT,STEP2),                       X
//                    PARM='XREF,LIST,LET'
[1]//SYSLIB    DD     DSNAME=BSLLIB,UNIT=SYSDA,DISP=(OLD,KEEP),             X
//                    VOLUME=SER=111111
//SYSLIN       DD     DSNAME=&LOADSET,DISP=(OLD,DELETE)
//             DD     DDNAME=SYSIN
//SYSUT1       DD     DSNAME=&UT1,UNIT=SYSDA,DISP=(NEW,DELETE),             X
//                    SPACE=(CYL,(2,1))
//SYSLMOD      DD     DSNAME=&GOSET(BSLGO),UNIT=SYSDA,                      X
//                    DISP=(MOD,PASS),SPACE=(1024,(50,20,1))
//SYSABEND     DD     SYSOUT=A
//SYSPRINT     DD     SYSOUT=A


//GO           EXEC   PGM=*.LKED.SYSLMOD,COND=(4,LT,LKED)
//SYSABEND     DD     SYSOUT=A
//BSLOUT       DD     SYSOUT=A
//BSLPUNCH     DD     UNIT=SYSCP
```

[1] Used by the operating system to locate the private library that contains the BSL library. If the library is resident in the system link library (SYS1.LINKLIB), the SYSLIB statement should be omitted.

• Figure 9. Cataloged Procedure BSLALG

Compilation Without Assembly

The compile-only operation allows you to compile a batch (one or more) of BSL external procedures without the production of an output data set of assembler text. You should specify the NOASSEM option for compile-only operations. If you specify the PUNCH option, the compiler produces a punched deck containing the generated assembler text. This punched output may subsequently be used as input to an assembler.

The BSLX cataloged procedure executes the BSL compiler as a single OS job step. This job step performs the compile-only operation. Figure 10 describes the input stream that you must provide for the BSLX cataloged procedure.

```
[1]//AA        JOB   JOHNDOE,MSGLEVEL=1
[2]//JOBLIB    DD    DSNAME=COMP,UNIT=2311,DISP=(OLD,PASS),               X
   //                VOLUME=SER=111111
[3]//STEPA     EXEC  BSLX
[4]//STEP1.SYSIN     DD *
[5] Compiler Control Statements
         .
    First BSL External Procedure
         .
         .
[6]$$$
[5] Compiler Control Statements
         .
    Next BSL External Procedure
         .
         .
[6]$$$
[5] Compiler Control Statements
         .
    Last BSL External Procedure
         .
         .
[6]/*
```

[1] The JOB statement is required. The jobname and the parameters in the operand field are optional.

[2] Used by the operating system to locate the private library that contains the BSL compiler. If the compiler is resident in the system link library (SYS1.LINKLIB), the JOBLIB statement should be omitted.

[3] Identifies the BSLX cataloged procedure. If you want to add the SORMGIN, SIZE, or CONCHAR options, specify PARM.STEP1='options' in the operand field of this statement.

[4] Describes the data set containing the compiler input.

[5] Should include the NOASSEM option.

[6] The $$$ delimiter is used only between BSL external procedures. The last (or only) BSL external procedure is followed by the /* delimiter.

• Figure 10. Sample Use of BSLX

Compilation and Assembly

If you are going to assemble the compiler's output with an OS assembler, you can accomplish both compilation and subsequent assembly by executing two OS job steps. This method permits compilation and assembly of one external procedure. Batched compilation with subsequent assembly is not possible because the assembler cannot handle the multiple END statements generated by the compiler.

The BSLASM cataloged procedure executes the job steps required for compilation and subsequent assembly (Assembler F). Figure 11 describes the input stream that you must provide for the BSLASM cataloged procedure.

```
[1]//BB        JOB    JOHNDOE,MSGLEVEL=1
[2]//JOBLIB    DD     DSNAME=COMP,UNIT=2311,DISP=(OLD,PASS),           X
   //                 VOLUME=SER=111111
[3]//STEPA     EXEC   BSLASM
[4]//STEP1.SYSIN      DD *
     Compiler Control Statements
         .
     BSL External Procedure
         .
         .
[5]/*
[6]//STEP2.SYSLIB     DD DSNAME=SYS1.MACLIB,DISP=(OLD,KEEP)
[7]//                 DD DSNAME=MYLIB,DISP=(OLD,KEEP)
```

[1] The JOB statement is required. The jobname and the parameters in the operand field are optional.

[2] Used by the operating system to locate the private library that contains the BSL compiler. If the compiler is resident in the system link library (SYS1.LINKLIB), the JOBLIB statement should be omitted.

[3] Identifies the BSLASM cataloged procedure. If you want to add the SORMGIN, SIZE, or CONCHAR options, specify PARM.STEP1='Options' in the operand field of this statement.

[4] Describes the data set containing the compiler input.

[5] Separates the compiler input data set.

[6] Required only if OS macro instructions are used in the BSL program, or generated for the REENTRANT attribute. This statement describes the data set containing the OS macro definitions.

[7] Required only if user-defined macro instructions are used in the BSL program. This statement describes the data set containing your macro definitions.

• Figure 11. Sample Use of BSLASM

Compilation, Assembly, and Execution

Since BSL is a language for the writing of systems programs, the compiler makes no assumptions about the environment in which the resulting object program will be executed. The environment you select for program execution depends on the type of program that you write. You can run the program in a stand-alone mode or under some operating system. You can use any execution-time facilities that are available in the environment, such as control program services or special machine instructions.

Note: If you specify the REENTRANT procedure option, and DATAREG is not zero, the compiler generates the OS GETMAIN macro instruction. If you want reentrant code, but do not intend to execute the program under OS, You must specify DATAREG(0) and provide the storage and addressability for the AUTOMATIC data.

If you are going to execute the program under OS, compilation, assembly, linkage edit, and execution may each be separate steps of a single job.

The BSLALG cataloged procedure executes the job steps required to compile, assemble, linkage edit, and execute a single BSL external procedure. Additional object modules can be included in the linkage edit step. Figure 12 describes the input stream that you must provide for the BSLALG cataloged procedure.

```
[1]//CC           JOB    JOHNDOE,MSGLEVEL=1
[2]//JOBLIB       DD     DSNAME=COMP,UNIT=2311,DISP=(OLD,PASS),           X
   //                    VOLUME=SER=111111
[3]//STEPA        EXEC   BSLALG
[4]//STEP1.SYSIN         DD *
     Compiler Control Statements
         .
     BSL External Procedure
         .
         .
[5]/*
[6]//STEP2.SYSLIB DD     DSNAME=SYS1.MACLIB,DISP=(OLD,KEEP)
[7]//             DD     DSNAME=MYLIB,DISP=(OLD,KEEP)
[8]//LKED.SYSIN   DD     *
     Object Decks
         .
         .
[9]/*
[10]//GO.ddname   DD     (parameters)
```

[1] The JOB statement is required. The jobname and the parameters in the operand field are optional.

[2] Used by the operating system to locate the private library that contains the BSL compiler. If the compiler is resident in the system link library (SYS1.LINKLIB), the JOBLIB statement should be omitted.

[3] Identifies the BSLALG cataloged procedure. If you want to add the SORMGIN, SIZE, or CONCHAR options, specify PARM.STEP1='options' in the operand field of this statement.

[4] Describes the data set containing the compiler input.

[5] Separates the compiler input data set.

[6] Required only if OS macro instructions are used in the BSL program, or generated for the REENTRANT attribute. This statement describes the data set containing the OS macro definitions.

[7] Required only if user-defined macro instructions are used in the BSL program. This statement describes the data set containing your macro definitions.

[8] Required only if additional object modules are to be included in the linkage editing step. This statement describes the data set containing the object modules.

[9] Indicates the end of the object module data set.

[10] Required only to define additional data sets that may be needed for the execution of the BSL program.

• Figure 12. Sample Use of BSLALG

The cataloged procedures can also be used to compile, assemble, linkage edit, and execute several BSL external procedures in a single job. To accomplish this, you use and repeat the BSLASM cataloged procedure to compile and assemble each BSL external procedure except the last.

For the final BSL external procedure to be compiled and assembled, you use the BSLALG cataloged procedure in the normal manner. This sequence of steps causes all the assembler output to be placed in the &LOADSET data set, from where it is linkage edited and executed as a single load module.

Dynamic Invocation of the Compiler

In addition to being executed as a job step, the BSL compiler can be invoked during the execution of a problem program. Your program can pass control to the compiler through the use of the CALL, LINK, ATTACH, or XCTL macro instructions; or you can set up your own calling sequence using branching instructions. The compiler receives and returns control according to standard linkage conventions.

When the compiler receives control, it assumes that register 1 contains the address of a parameter list which is made up of contiguous fullword addresses on fullword boundaries. These address parameters are positional:

1. The first address parameter points to a list of compiler options. This address must always be provided. If there are no options to be specified, this address must point to a halfword of binary zeros.

2. The second address parameter points to a list of alternate ddnames. If this address is omitted, the high-order bit of the first address must be set to 1.

An example of dynamic invocation is provided in Appendix D.

Form of Invocation

If you use the CALL, LINK, or ATTACH macro instructions to invoke the compiler, the macro expansion builds the parameter list and loads the pointer into register 1. If you use XCTL or your own calling sequence, your program must build the parameter list and load the pointer into register 1. The various methods of invoking the compiler are shown below:

Operation	Operands
LINK ATTACH	EP=BSL,PARAM=(optionlist[,ddnamelist]),VL=1
CALL	BSL,(optionlist[,ddnamelist]),VL
XCTL	EP=BSL
LA L BALR	1,parlist 15,bsladdr 14,15

optionlist
: specifies the address of a variable length list containing the options usually specified in the PARM field of the EXEC statement. If there are no options to be specified, this address must point to a halfword of binary zeros.

ddnamelist
: specifies the address of a variable length list containing alternate ddnames for the data sets used by the compiler. If alternate ddnames are not required, this operand may be omitted.

parlist
: specifies the address of a parameter list which is built by your program. If your parameter list contains only one address (optionlist), its high-order bit must be set to 1. If the high-order bit of the first address is not set to 1, the parameter list must contain a second address (ddname list).

bsladdr
: specifies the entry point of the load module containing the BSL compiler. You should use a V-type address constant for BSL, or issue a LOAD macro instruction for BSL and use the address that is returned in register 0.

Option List

The first address parameter in the parameter list points to a variable length list of compiler options. This option list must be provided in the form:

counter	option1,option2,option3

where counter is a halfword on a halfword boundary. This halfword counter contains the binary value of the number of characters in the string of options. The maximum value is 35. If the counter is set to binary zeros, no options are specified.

The string of options contains the same compiler options that may be specified in the PARM field of the EXEC statement. These three options (SIZE, SORMGIN, and CONCHAR) can be specified in any order, and any or all may be omitted.

List of Alternate DD Names

The second address in the parameter list points to a variable length list containing alternate ddnames for the data sets used by the BSL compiler. This ddname list must be provided in the form:

counter	string of ddnames

where counter is a halfword on a halfword boundary. The halfword counter contains the binary value of the number of characters in the string of ddnames. The maximum value is 88. If the counter value is less than or equal to 32, no alternate ddnames are specified.

The ddnames in the list must be specified in the same positions as the standard order of I/O ddnames. Each ddname must be an 8-byte element of the string. If the last ddname is less than eight characters, the low-order bytes will be padded with blanks. If a ddname is omitted within the list, the 8-byte entry must contain binary zeros. If necessary, ddnames can be omitted from the end of the list by adjusting the counter value.

Table 5 shows the standard list of ddnames, the seven ddnames used by the BSL compiler, the positions for specifying the seven alternate ddnames, and the required counter values. Whenever you do not specify an alternate ddname, the compiler ddname is used. The compiler ddnames are restored after every invocation.

• Table 5. Alternate DD Names

Dec. Displ.	Standard DD Names	Compiler DD Names	Alternate DD Names	Counter (binary)
+0	SYSLINƀƀ	not used	---	---
+8	member	not used	---	---
+16	SYSLMODƀ	not used	---	---
+24	SYSLIBƀƀ	not used	---	---
+32	SYSINƀƀƀ	SYSINƀƀƀ	ddname1	40
+40	SYSPRINT	SYSOUTƀƀ	ddname2	48
+48	SYSPUNCH	SYSPUNCH	ddname3	56
+56	SYSUT1ƀƀ	SYSUT1ƀƀ	ddname4	64
+64	SYSUT2ƀƀ	SYSUT2ƀƀ	ddname5	72
+72	SYSUT3ƀƀ	SYSUT3ƀƀ	ddname6	80
+80	SYSTERMƀ	SYSTERMƀ	ddname7	88

Using Control Program Services and Special Machine Instructions

The GENERATE statement allows you to use any control program service or special machine instruction in your BSL program. In the case of control program services or other pre-defined macro instructions, you are responsible for ensuring that the required macro expansions are available at assembly time.

The BSL compiler never inspects the contents of a GENERATE statement. If you use the GENERATE statement, you must ensure that the contents are legal and properly related to the compiler-generated assembler text.

Other considerations for using the GENERATE statement are explained under "BSL Programming Techniques" in Section 7 of this manual. Detailed examples are listed in Appendix A. The form of the GENERATE statement is described in the BSL Language Specification manual.

Using the BSL Library Routines

When executing a BSL program under OS, you may use the service routines provided in the BSL library. These routines perform certain basic functions that are useful for testing your programs. To call a library routine, you specify the entry name in a BSL CALL statement, and identify the BSLLIB data set in the linkage-editing operation:

- The BSLALG cataloged procedure provides for use of the BSL library. In the linkage-editing job step, the SYSLIB DD statement describes the BSLLIB data set. No additional DD statements are required.

- If you do not use the cataloged procedure, your linkage-editing job step must include a SYSLIB DD statement that describes the BSLLIB data set, and you may not specify the linkage-editor option NCAL.

Complete descriptions and examples of the available routines can be found in the BSL Library publication. For your reference, a summary of the functions is listed in Table 6. (The BSL library also contains the TRACE routines which are described in Section 2.)

The facilities provided by the library routines should not be regarded as part of the BSL language. The output of the BSL compiler is independent of any operating environment, but the library routines function only under OS. (Except for ERRINT, similar library routines with identical entry names are provided in the compiler distribution package for DOS.)

Table 6. Functions of the BSL Library Routines (OS)

Routine Names	Entry Points	Functions
EDIT	GET GETS PUT PUTS TABSET	Performs reading, writing, and conversion of data in a manner similar to PL/I EDIT DIRECTED I/O.
ERRINT	ERRINT ERRCL	Gives the user a facility like PL/I ON-UNITS for handling program interruptions.
PDUMP	PDUMP	Performs snapshot dumps of specific data areas. The dumps are provided in hexadecimal, character, or bit string formats.
SUBSTR	SUBSTR	Performs character substring assignment while checking substring ranges.
BSL I/O	READ PRINT PUNCH CLOSE	Employs the queued sequential access method (QSAM) to provide basic input/output services at object time. These routines can read a logical record, write a logical record, punch a logical record, and close the system output data set.
Interrupt Handler	IKETRCII IKETRCID	Provides statement number and save area trace when a program check occurs.

Including Text From a Library

When you compile under OS, you can use the compiler's INCLUDE features:

- The INCLUDE macro statement (%INCLUDE) incorporates text during compile-time macro processing.

- The INCLUDE option ($INCLUDE) incorporates text after macro activity and does not submit the text to scanning for macro substitution.

These features allow you to keep certain standard declarations or sections of source code in external libraries. As needed, these items can be obtained from the libraries during compilation, and included in your programs.

Storing Text in External Libraries

The section of code that is to be kept for inclusion in source programs must be a member of a partitioned data set. The data set requirements are:

- The partitioned data set must contain fixed format records; the DCB parameter RECFM must not specify U (undefined) or V (variable).
- The partitioned data set must contain 80-byte logical records; the DCB parameter LRECL must specify 80.
- The partitioned data set may contain blocked records up to a block size of 3520 bytes. The compiler obtains storage for two input blocks, and this storage must be taken into account when determining the total storage size required for BSL compilation (see SIZE option).

If you attempt to include text from a data set that does not conform to the above requirements, the compiler issues an error message and compilation is terminated.

You can use the IEBUPDTE utility program to create or add to your library of text. The following example shows how to do it. In the example, the SYSUT2 DD statement describes the partitioned data set that will contain the text, and the sections of text to be added as members are named MEMBR1 and MEMBR2. The example is:

```
//JOB1        JOB
//STEP1       EXEC  PGM=IEBUPDTE,PARM=NEW
//SYSPRINT    DD    SYSOUT=A
//SYSUT2      DD    DSNAME=TEXTLIB,UNIT=2311,DISP=(NEW,KEEP),          X
//                  VOLUME=SER=111111,SPACE=(80,(25,25,1))
//SYSIN       DD    *
./            ADD   NAME=MEMBR1,LEVEL=00,SOURCE=0,LIST=ALL
   Card Deck (containing section of text to be kept in the library)
        .
        .
        .
./            ADD   NAME=MEMBR2,LEVEL=00,SOURCE=0,LIST=ALL
   Card Deck (containing section of text to be kept in the library)
        .
        .
        .
./            ENDUP
/*
```

Specifying INCLUDE Option or Macro Statement

The INCLUDE option or macro statement may appear at any point (and any number of times) in the compiler input. When INCLUDE is employed, the contents of the named member of the partitioned data set is used as input to the compiler. This included text may not contain an INCLUDE.

You specify INCLUDE in the following format:

```
 $INCLUDE   ddname(member name)

            or

 %INCLUDE   ddname(member name);
```

where:

{$INCLUDE
%INCLUDE}

specifies the type of INCLUDE. The dollar sign ($) identifies the INCLUDE option, and must appear in the first column (leftmost source margin). The dollar sign is the default control character -- a different character can be specified by means of the CONCHAR compiler option. The word INCLUDE must immediately follow (no space) the dollar sign, and cannot be combined with other options in the same compiler control statement.

The percent sign (%) identifies the INCLUDE macro statement, as described in the _BSL Language Specification_ manual.

ddname

identifies the DD statement that describes the partitioned data set from which the text is to be obtained. The appropriate DD statement must be included in the job control statements for the compilation.

(member name)

identifies the member of the partitioned data set. The contents of this member are included in the compiler input at the point where INCLUDE is specified. If the named member does not exist in the defined data set, compilation is terminated.

The following example shows the use of the INCLUDE option. In the example, text is obtained from the MEMBR1 and MEMBR2 members of the partitioned data set named TEXTLIB. The TEXTLIB data set is described by the LIB1 DD statement. The LIB1 DD statement, as well as the member names, are identified in the INCLUDE option. The example is:

```
//JOB2          JOB
//JOBLIB        DD     DSNAME=COMP,UNIT=2311,DISP=(OLD,PASS),            X
//                     VOLUME=SER=111111
//STEPA         EXEC   BSLASM
//STEP1.LIB1    DD     DSNAME=TEXTLIB,UNIT=2311,DISP=OLD,                X
//                     VOLUME=SER=111111
//STEP1.SYSIN   DD     *
$ASSEM,TIME
  R1: PROC;
      DCL M PTR(31), (Q,X) ENTRY INTERNAL;
      CALL X;
      CALL Q;
$INCLUDE LIB1(MEMBR1)
   Q: PROC;
$INCLUDE LIB1(MEMBR2)
      J=J+1;
   END;
  END;
/*
```

Using the TESTRAN Facilities

If you test your BSL program under OS, you can take advantage of the debugging facility called TESTRAN. TESTRAN is a feature of OS; it is not part of the BSL compiler.

TESTRAN assists in debugging BSL programs in the same manner as with assembler language programs. The scope of this manual does not include a detailed description of TESTRAN processing (see the TESTRAN publication, Form C28-6648); however, items of special significance to BSL programmers are explained in the following paragraphs. In addition, examples illustrating the use of TESTRAN with BSL are provided in Appendix C.

BSL Requirements for the TESTRAN Control Section

You should place all TESTRAN macro instructions to be used in a separate control section. This TESTRAN control section should execute the TEST OPEN macro instruction, and call the BSL program. The entry name of the BSL external procedure that is to be the primary entry point of your program must be identified by an EXTRN instruction in the TESTRAN control section.

For the DUMP DATA statement of TESTRAN, you may refer to the BSL data in two ways:

1. If the data is STATIC in BSL, refer to it by name.
2. If the data is AUTOMATIC in BSL, you should use a DSECT instruction in the TESTRAN control section, and point the appropriate register at the item in the BSL program.

Job Control Statements for TESTRAN

When you run your BSL program with TESTRAN, certain items must be added to the job control statements. You must:

- Specify the entry name of the TESTRAN control section. This name must be specified by the linkage editor ENTRY statement in the linkage editing step.
- Specify the linkage editor TEST option in the linkage editing step.
- Define a SYSTEST data set for the execution step.
- Execute an additional step after execution to perform the TESTRAN EDIT.

If you specify the TEST parameter for the assembly job step, names in BSL programs can be referred to without special attributes. The TEST parameter causes the assembler to produce symbol tables in the object module. The following example shows how to specify the TEST parameter when you use the BSLALG cataloged procedure:

```
//STEPA   EXEC   BSLALG,PARM.STEP2='TEST'
```

Using the CLEAR Facilities

If you develop or maintain your BSL program under OS, you can use the control facility called CLEAR. CLEAR is a separate facility; it is not part of the BSL compiler. To use CLEAR with BSL, your system must have CLEAR Version 4.0 or a later version.

CLEAR allows you to use the verbs ALTER, EDIT, INTEG, MODIFY, and SCREATE, by specifying the keyword parameter CMPLR=BSL. If your CLEAR system has been modified with PTF (referencing APAR) number CL4017, you can also specify a number of optional keyword parameters which invoke the appropriate compiler options described in section 2 of this manual. The optional keyword parameters are:

BCOL=beginning card column
: specifies the card column in which the BSL source statements begin. The compiler uses this value as the leftmost source margin. You must specify a three-digit number, using leading zeros as necessary. The default is BCOL=003. (Using CLEAR Version 5.1 the default is BCOL=002.)

ECOL=ending card column
: specifies the card column in which the BSL source statements end. The compiler uses this value as the rightmost source margin. You must specify a three-digit number, using leading zeros as necessary. The default is ECOL=072.

CONCHAR=control character
: specifies a single character that will be used as the identifier of the compiler control statements in the compiler input. The default is CONCHAR=$. (Using CLEAR Version 5.1 the default is CONCHAR=@.)

SIZE=dictionary size in multiples of 1000 bytes
: specifies the size of main storage (in multiples of 1000 bytes) that the BSL compiler may use for dictionary space. The default is SIZE=48. (Using CLEAR Version 5.1 the default is SIZE=30.)

BSLRGN=region size in multiples of K
: specifies a region size for the BSL compiler job step. You specify a number up to four digits, followed by the character K (K=1024 bytes). The default is BSLRGN=175K. (Using CLEAR Version 5.1 the default is BSLRGN=160K.)

INCLUDE=data set name
: causes the generation of a DD statement named SYSLIB which defines a DSNAME equal to the name shown in this keyword. The named data set must be cataloged. This parameter allows you to use the BSL INCLUDE option in your source program, provided the INCLUDE statement specifies SYSLIB in the ddname field.

PVTMACS=data set name
PVTMAC2=data set name
PVTMAC3=data set name
: provides access to private macro libraries during the assembler job step of the BSL job. The requirements for using these keyword parameters are exactly the same as described in the <u>CLEAR Programmer's Guide</u> publication (Form Z28-6636) under the ALTER, EDIT, INTEG, MODIFY, and SCREATE verbs.

If your system has not been modified with PTF number CL4017, you cannot use the optional keyword parameters described above. The default values are assigned for the BCOL, ECOL, CONCHAR, SIZE, and BSLRGN keywords; and the options for private libraries (INCLUDE, PVTMACS, PVTMAC2, and PVTMAC3) are not available.

Section 5: Compiler Conventions and Restrictions

This section describes:

- The conventions that the BSL compiler follows for the generation of its assembler text output. These conventions concern the general form of the output program, the types of compiler-generated statement labels, the use of registers, and the linkage to called and calling programs. You will need this information to communicate with or modify BSL-produced programs, or to include assembler text in your BSL source program.
- The restrictions that affect the type of program you can successfully compile. Some of the restrictions are necessitated by the compiler's table sizes; others are due to BSL language implementation.

General Form of Compilation

Compilation of a BSL external procedure produces one assembler language control section (CSECT). Separate control sections are not generated for BSL internal procedures. The control section produced by the compiler contains:

- All the executable instructions generated by the compilation.
- Compiler-generated constants and temporaries.
- Data declared as STATIC LOCAL in the BSL source program.
- An assembler language ENTRY statement for data declared as LOCAL EXTERNAL in the BSL program.
- An assembler language EXTRN statement and an A-type address constant for items (except branch points) declared as NONLOCAL EXTERNAL in the BSL program.
- An assembler language V-type address constant for all branch points external to the BSL program.
- All text specified in BSL GENERATE statements.

If you specify the REENTRANT option, the compiler also generates a dummy control section (DSECT) to describe the AUTOMATIC storage that is required.

Compiler-Generated Labels

The labels (statement identifiers) that you use in the BSL source program are reproduced in the compiler-generated assembler text. However, the compiler must generate additional labels to identify areas, values, and statements created by expansion of the BSL program into assembler text. To help you identify various items in the assembler text, the conventions for compiler-generated labels are listed in Table 7.

As shown in Table 7, almost all of the compiler-generated labels begin with either the character @ or the character &. The label types that may be needed more than once in a control section are followed by an integer that is incremented sequentially. For example, if four separate character constants are required, they may be labeled @C1, @C2, @C3, and @C4.

• Table 7. Labels Generated by the Compiler (Part 1 of 2)

Label	Function
@AD01	Identifies the address of @DATA1 in a non-reentrent program when the DATAREG option is specified.
@DATA1	Identifies the compiler constant area.
@DATA	Identifies the start of the static data area, and the end of the generated code.
@DATD	Identifies the DSECT that describes the dynamic storage area.
@DATEND	Identifies the end of the DSECT that describes the dynamic storage, or the end of the data area in a non reentrant program.
@SIZ001	Identifies a value that represents the size of the dynamic storage area.
&SPN	Subpool number of dynamic storage for reentrant procedure.
@TEMPS	Identifies an area that contains space for temporaries.
@CTEMPinteger	Identifies a string temporary.
@TEMPinteger	Identifies an arithmetic temporary which has high-order zeros.
@SAVinteger	Identifies a procedure save area.
@IFinteger	Identifies a value that represents the length of a temporary area to be cleared for string expressions (variable length substrings).
@L	Identifies a value that represents the length of a temporary area needed in reentrant procedures.
&T	Used to initialize interleaved arrays.
@PLinteger	Identifies an argument list for reentrant procedures.
@ELinteger	Identifies the epilogue of a procedure.
@Ainteger	Identifies an A-type address constant.
@Vinteger	Identifies a V-type address constant.
@Binteger	Identifies a bit constant.
@Cinteger	Identifies a character constant.

(Part 1 of 2)

Table 7. Labels Generated by the Compiler (Part 2 of 2)

Label	Function
@Dinteger	Identifies an arithmetic constant.
@Tinteger	Identifies a temporary location used for evaluating an arithmetic expression.
@Xinteger	Identifies a hexadecimal constant.
@DOinteger	Identifies statements in the generation of a DO loop.
@CLinteger	Used in non-reentrant procedures to identify and branch around argument lists.
@integer	Used for IF branches and to branch around ELSE statements.
@CLC @MVC @NC @OC @XC	Identifies instructions which must be executed by means of an EX instruction.
@0,@1,...@F	Symbolic names for the general registers.
.@001	Label of an ANOP following a LCLA assembler instruction.
Ainteger	Name generated for items declared without a name (*).
@PSTART	Label for first instruction following the BALR that establishes the primary base register.

Register Conventions and Options

Although your BSL source program need not be concerned with the use of registers, the BSL compiler must use registers in the generated assembler text. If you do not specify the BSL procedure options CODEREG, DATAREG, or REENTRANT, the compiler follows the basic register conventions listed in Table 8.

• Table 8. Register Use Table

Register Number	Function
0	Alternate area for holding partial results of expression evaluations.
1	Used as pointer to parameter and argument lists.
11	Used as base register for addressing data and code.
12	Used as base register for addressing the dynamic storage area in reentrant procedures.
13	Used as pointer to the current procedure's save area.
14	Used as linkage for calls.
15	Used as branch register for external calls.
2 through 9, and 12	Used to hold pointer values, index calculations, and to evaluate arithmetic expressions.
10 and 14	Used for subscript and substring computations.
14 and 15	Used to hold the results of expression evaluations.

The following procedure options affect the register assignment for the compilation:

CODEREG (register numbers)
: allows you to designate the base registers used for addressing the generated code. The registers you specify replace register 11 as the base register, and are unavailable for any other purposes in the entire compilation. Registers 1, 10, 13, 14, and 15 cannot be specified in this option.

If you specify zero as the register in this option, the compiler does not generate any addressability. In this case, the compiler assumes that you provided your own addressing by means of a GENERATE statement, or that your program resides in the lower 4K of main storage.

DATAREG (register numbers)
: allows you to designate the base registers used for addressing the data. The registers you specify are unavailable for any other purposes in the entire compilation. Registers 1, 10, 13, 14, and 15 cannot be specified in this option.

If you specify zero as the register in this option, then the same base register is used for the data and the executable code, and the compiler generates no storage or addressability for the AUTOMATIC data.

REENTRANT

requires the existence of separate base registers for addressing data and code. You can designate specific registers by using the DATAREG and CODEREG options described above, or you can let the compiler assign the base registers by the following conventions:

Register 12 -- Base register for addressing the dynamic storage area.

Register 11 -- Base register for addressing the generated code and STATIC data.

Linkage Conventions and Options

The BSL compiler generates code to handle the linkage between called and calling programs. The code generated for BSL external procedures follows the established OS type 1 linkage conventions; that is, specific registers are assigned to handle the various linkage functions, and a save area in storage is used to preserve the contents of all registers. You can use certain BSL procedure options to override the established linkage conventions.

Use of Linkage Registers

The OS type 1 conventions for the use of linkage registers are summarized below:

- Register 15 contains the address of the entry point to the called program. The called program can use register 15 as its initial base register for addressability. Register 15 can contain a return code when control is returned to the calling program.

- Register 14 contains the address of the next sequential instruction in the calling program. The called program can use register 14 to return to the calling program.

- Register 13 contains the address of the calling program's save area.

- If parameters are passed to the called program, register 1 contains the address of a list of fullword quantities that are the addresses of the parameters.

Use of Save Areas

The OS type 1 conventions for the use of save areas are summarized below:

- Every calling program provides a 72-byte save area for the preservation of its registers, and places the address of this save area in register 13. (If your program does not call any other programs, you need not provide a save area.)

- The called program saves the contents of registers 14 through 12 in the calling program's save area, in accordance with the format shown in Table 9.

- If the called program has no save area of its own, then the address of the calling program's save area is kept in register 13.

- If the called program has its own save area, then it places the address of the calling program's save area (from register 13) into the chain-back field of its own save area; and places the address of its own save area into register 13. Optionally, the called program also may place the address of its own save area into the chain-forward field of the calling program's save area.

- Before returning to the calling program, the called program restores the contents of all registers. First the called program restores register 13 from its own save area (if necessary), and then uses this pointer to restore registers 14 through 12 from the calling program's save area.

Table 9. Format of Save Area

Word	Contents
1	Not used
2	Chain back (address of calling program's save area)
3	Chain forward (address of called program's save area)
4	Register 14 (return address)
5	Register 15 (entry point address and return code)
6	Register 0
7	Register 1
8	Register 2
9	Register 3
10	Register 4
11	Register 5
12	Register 6
13	Register 7
14	Register 8
15	Register 9
16	Register 10
17	Register 11
18	Register 12

Effect of Procedure Options

There are three BSL procedure options (SAVE, DONTSAVE, and NOSAVEAREA) that modify the linkage conventions that are followed by the compiler. These options allow you to define your own linkage conventions, or to reduce the program housekeeping associated with internal procedures. The effect of these options is:

SAVE (register numbers)
specifies the registers to be saved and restored. Only the registers you specify are saved -- you can use the other registers to propagate register changes back to the calling program. If you specify this option with no registers, all the registers are saved.

DONTSAVE (register numbers)
specifies the registers not to be saved. The registers you specify are not saved, and may be used to propagate register changes back to the calling program. If you specify this option with no registers, none of the registers are saved. If register 13 is not saved, the chain forward field in the old save area is not updated.

The SAVE and DONTSAVE options are mutually exclusive; that is, you cannot use both options in the same PROCEDURE statement.

NOSAVEAREA
eliminates the generation of a save area. This option is useful with a procedure that does not call any other procedures. When you use this option, your program:

1. Does not establish a save area.

2. Does not update the chain-forward field in the calling program's save area.

3. Does not restore the contents of register 13 before returning control to the calling program.

Note: You can suppress all of the BSL compiler's initial procedure housekeeping by specifying DONTSAVE, NOSAVEAREA, CODEREG(0), and DATAREG(0).

Conventions for Reentrant Procedures

If you specify the BSL procedure option REENTRANT, the compiler generates reentrant code in accordance with the conventions described in the following paragraphs. You should note that the generated code is reentrant only for the BSL external procedure together as a unit with its internal procedures. In other words, the internal procedures by themselves are not reentrant.

Obtaining Dynamic Storage

In the prologue of the external procedure, the BSL compiler generates an OS GETMAIN macro instruction to obtain an area for dynamic storage. The storage is obtained from subpool 0. A different subpool can be specified by setting &SPN to the desired value. For example:

```
   GEN;
&SPN   SETA   1
$ENDGEN
```

This will set the subpool number to 1. If you do not intend to execute a reentrant program under OS, you specify the procedure option DATAREG(0), in which case the compiler generates no storage or addressability for the AUTOMATIC data (you must obtain your own dynamic storage.)

The code expansion of the GETMAIN macro instruction requires the use of register 1. If your procedure has one or more parameters, the compiler generates code to restore register 1 after execution of GETMAIN. If your procedure has no parameters, register 1 is not restored. If you need to preserve the contents of register 1, you can specify a dummy parameter for your procedure.

The size of the dynamic storage area required by the procedure is always stored in an area labeled @SIZ001, which is one fullword in length and located on a word boundary in the static area. You can use this size specification to obtain your own dynamic storage.

Using the Dynamic Storage Area

The compiler maps the dynamic storage area in a dummy control section (DSECT) labeled @DATD. A separate base register is assigned to address this area -- you can specify a particular register with the procedure option DATAREG.

Data generated by GENERATE DATA statements will be placed in the dynamic storage area. Remember to use only define storage and equate statements, not define constants.

Data items that are not initialized, as well as data items that are declared AUTOMATIC, are placed in the dynamic storage area by default. The compiler will not allow you to to specify the INITIAL attribute for data declared as AUTOMATIC.

Save areas are located in dynamic storage.

When non-arithmetic items or off-boundary arithmetic items are used in arithmetic expressions, the compiler requires a temporary storage space with high-order zeros. In a reentrant environment, the compiler generates code to clear an area for this temporary storage -- the area is labeled @TEMPS and its length is equated to the label @L. If you specify the DATAREG(0) option to suppress the GETMAIN macro instruction, then you assume responsibility for clearing this temporary area. This can be accomplished with the following instruction:

```
GEN (XC @TEMPS(@L),@TEMPS);
```

BSL Size Restrictions

The maximum size of some items in your BSL procedure are limited by the size of the compiler's work areas. These size restrictions are listed, by category, in Table 10.

• Table 10. BSL Size Restrictions (Part 1 of 2)

Item	Limitation
Internal procedures	The maximum number of internal procedures in a single compilation is 14.
Secondary entry points	The maximum displacement between a secondary entry point and the beginning address of the procedure is 4095 bytes.
Variables	Generally, the number of variables in a single compilation should not exceed 1000 (with SIZE=96).
Constants	The maximum number of unique FIXED constants in a single compilation is 75. The maximum number of unique address constants in a single compilation is 75. The maximum length of a string constant in an INITIAL attribute is 256 characters. The maximum length of a string constant in an assignment statement is 53 characters. A constant such as '010'B counts as three characters. The maximum total length of all string constants in a single compilation is 1400 characters. A constant such as '010'B counts as three characters. Duplicates do not count. Constants used as initial values in DECLARE statements do not count.
Structures and arrays	The maximum number of levels in a structure is 255. This maximum applies to the actual number of levels. The maximum number of components in any one level of a structure is 255. The maximum offset of any element in a structure from the beginning of its major structure is 32,767 bytes. The maximum dimension of an array is 32,767 bytes. An array can have only one dimension.
DO loops	The maximum level of nesting for DO loops is 8.

(Part 1 of 2)

Table 10. BSL Size Restrictions (Part 2 of 2)

Item	Limitation
IF statements	The maximum level of nesting for IF statements is 14. The maximum number of true/false branches in an IF statement is 24. The maximum number of true/false branches in a nest of IF statements is 50. The number of true/false branches is equal to the number of IF statements plus the number of logical connectives (| or &).
Pointers	The maximum number of explicit pointer qualifications applied to a variable is 7. The maximum number of implicit pointer qualifications is 7.
Factored attributes	The maximum number of left parentheses used to factor attributes in a DECLARE statement is 20. The maximum number of identifiers that can have factored attributes in a single declaration is 49.
CALL statements	The maximum number of arguments in a CALL statement is 25.
Macro statements	The maximum number of macro variables and labels is 500. The maximum length of a macro string variable is 1000 bytes. The total length of all macro string variables should not exceed 45,000 bytes at any one time. The range of a macro FIXED variable is from -9999999 to 99999999.

BSL Language Restrictions

Some uses of the BSL language features are not supported by the current compiler. These language restrictions are described, by general category, in the following paragraphs.

Reserved Words

Some words have special meanings to the BSL compiler, and therefore must not be used as identifiers in your BSL source program. These reserved words are:

ABS
ADDR
BY
CALL
DCL
DECLARE
DO
ELSE
END
ENTRY
GEN
GENERATE
GO
GOTO
IF
PROC
PROCEDURE
RELEASE
RESTRICT
RETURN
THEN
TO

The following words are reserved only in the compiler's macro phase:

ACT
ACTIVATE
DEACT
DEACTIVATE
INCLUDE

Procedure Format

An internal procedure must be placed immediately preceding the END statement of the external procedure. If more than one procedure is internal to the same external procedure, they must immediately follow one another and the group must be immediately followed by the END statement of the external procedure. An example of an illegal format is:

```
A:  PROC;
    .
    .
    .
    B:  PROC;
    .
    .
    .
    END  B;
    X = 1;     /* THE POSITION OF THIS STATEMENT IS ILLEGAL */
    END  A;
```

Procedure Parameters

Parameters common to more than one entry point must be in the same parameter position. You can use dummy parameters to obtain proper positioning. An example of this restriction is:

```
R1:  PROC(Q1,Q2);
R2:  ENTRY(Q1,Q2,Q3);  /* LEGAL */
R3:  ENTRY(Q2,Q1,Q3);  /* ILLEGAL PARAMETER POSITIONS */
R4:  ENTRY(Q3,Q1,Q2);  /* ILLEGAL PARAMETER POSITIONS */
```

If the main entry point has no parameters, secondary entry points cannot have parameters. An example of an illegal parameter at a secondary entry point is:

```
A:  PROC;
      .
      .
B:  ENTRY(P1); /* ILLEGAL PARAMETER */
```

Internal procedures cannot reference parameters of the containing procedure. An example of illegal reference is:

```
P1:   PROC(A);
      DECLARE B FIXED;
      P2:  PROC;
           A = 5;  /* ILLEGAL REFERENCE TO A */
           B = 5;  /* LEGAL REFERENCE TO B */
           END P2;
      END P1;
```

CALL Statements

You cannot use an explicit pointer to qualify the entry name in a CALL STATEMENT. An example of illegal qualification is:

```
CALL P-> NAME; /* ILLEGAL POINTER QUALIFIER */
```

Arguments in a CALL statement must not be:

- REGISTER variables.
- Expressions involving 'and', 'or', or 'exclusive or' operations.
- Bit strings that are not on byte boundaries.

Addresses

If the constant displacement from the starting address of EXTERNAL, parameter, or BASED data items is greater than 4095 bytes, an addressing error occurs. The constant displacement is equal to the sum of the following:

- For an element in a structure, the displacement of the element from the beginning of the major structure.
- For an array element with a constant subscript, the product of the constant subscript, minus one, and the element length.
- For an element with a constant substring, the displacement of the initial character of the substring from the beginning of the base string.

An example of an illegal displacement is:

```
DCL 1 X BASED(P),
      2 Y CHAR(5000),
      2 Z CHAR(1);

Z = 'A':  /* ADDRESSING ERROR OCCURS BECAUSE ELEMENT Z IS 5000 BYTES
             FROM BEGINNING OF STRUCTURE */
```

If an absolute address is used in the declaration of a BASED attribute, that address cannot be greater than 65,535. An example is:

```
DCL A BASED(70000); /* ILLEGAL ADDRESS */
```

Register Attribute

Data declared with the REGISTER attribute must be either FIXED(31), PTR(31), or PTR(24). No boundary attribute can be specified.

Initialization

Arithmetic items must not be initialized with BIT or CHARACTER constants, or with addresses.

String items must not be initialized with a character string that is longer than the declared length of the string item.

String items must not be initialized with binary or decimal constants, or with addresses.

If hexadecimal or bit string constants do not fill a byte multiple, the low-order bits are padded with zeros.

If bit string items are initialized as components of a structure, or are elements of a bit array, bits are not packed; that is, each bit string starts on a byte boundary. For example:

```
DCL 1 X,
     2 R BIT(1),
     2 Y BIT(3)INIT('111'B);
  /* R AND Y ARE NOT PACKED INTO THE SAME BYTE */
     2 Z(5) BIT(1);
  /* EACH ELEMENT OF Z IS IN SEPARATE BYTE */
```

String Items

String items used in arithmetic expressions cannot be longer than 4 bytes. String items used in assignment statements or comparison expressions cannot be longer than 256 bytes.

String Assignment Statements

If a bit variable is assigned to a bit variable, the variables must be on byte boundaries and must be multiples of 8 bits. An example of legal and illegal assignments is:

```
DCL A BIT(16), B BIT(8), C BIT(3);
 A = B;     /* LEGAL ASSIGNMENT */
 A = C;     /* ILLEGAL ASSIGNMENT */
 C = A;     /* ILLEGAL ASSIGNMENT */
```

If a bit constant is assigned to a bit variable, and the bit variable is not on a byte boundary, then the assigned bit string must not cross two byte boundaries (it may cross one byte boundary.) If the bit variable is aligned on a byte boundary, the length of the bit string is not restricted. Examples of legal and illegal assignments are:

```
DCL 1 R,
      2 X BIT(3)
      2 M BIT(13),
      2 Q BIT(3);
 M = '1111110000001'B; /* LEGAL BECAUSE IT CROSSES ONE BYTE
                          BOUNDARY */
DCL 1 R,
      2 X BIT(3),
      2 M BIT(14),
      2 Q BIT(2);
 M = '11111110000000'B; /* ILLEGAL BECAUSE IT CROSSES TWO BYTE
                           BOUNDARIES */
```

Comparison Expressions

If the left side of a string comparison has constant length substring, the right side cannot have a variable length substring. For example:

```
DCL (A,B) CHAR(10);
 IF A(I:J) = B(K:L) THEN...  /* LEGAL */
 IF A = B(K:L) THEN...       /* ILLEGAL */
```

The length of the left side of a string comparison should be equal to the length of the right side. If the right side is longer, the comparison is still made by ignoring the extra length. However, if the left side is longer, no comparison is made and an error message is issued. For example:

```
DCL (A,B) CHAR(10), (C,D) CHAR(4);
 IF A = C THEN GOTO L1;          /* ILLEGAL */
 IF (A&B)>(C&D) THEN GOTO L1;    /* ILLEGAL */
```

If a string data item appears on the left side of a comparison, then only string data items can appear on that side of the comparison. An example of illegal combinations is:

```
DCL C4 CHAR(4), F31 FIXED(31), P31 PTR(31);
 IF (C4&F31) = 0 THEN GOTO L1;     /* ILLEGAL */
 IF (P31 && C4) > 0 THEN GOTO L1; /* ILLEGAL */
```

Expressions involving comparison operators, as well as comparisons connected by 'and' or 'or' operators, must not be enclosed in parentheses. However, when 'and', 'or', or 'exclusive or' is used as a bit operator in an IF statement, the containing expression must be enclosed in parentheses. Use of the parentheses is shown in the following example:

```
IF A>B|C>D THEN...  /* LEGAL */
IF (A>B)|(C>D) THEN...  /* ILLEGAL */
IF A>B&C>D|E = F THEN...  /* LEGAL */
IF (A>B&C>D)|E = F THEN...  /* ILLEGAL */
IF (A|B) = (C&D) THEN...  /* LEGAL */
IF A|B = C&D THEN...  /* ILLEGAL */
IF (A&B)>C THEN...  /* LEGAL */
IF A&B>C THEN...  /* ILLEGAL */
```

If a comparison involves a bit variable that is not on a byte boundary, a bit variable not a multiple of eight bits on a byte boundary, or a bit constant that is less than eight bits, then the bit variable or constant must be:

- Entirely contained within the boundaries of one byte.
- On the left side of the comparison.
- Compared to a bit constant which is either all ones or all zeros.
- Compared by the = or ¬= operator.

For example:

```
DCL 1 R,
      2 Y BIT(3),
      2 Z BIT(5);
 IF Z(1:3) = '111'B THEN GOTO L1;   /* LEGAL */
 IF Z(1:3) = '101'B THEN GOTO L1;  /* ILLEGAL BECAUSE BIT CONSTANT IS
                                      MIXED ONES AND ZEROS */
 DCL B8 BIT(8);
 IF B8='11'B THEN GOTO L1;  /* LEGAL */
 IF B8='10'B THEN GOTO L1; /* ILLEGAL BECAUSE BIT CONSTANT IS
                                      MIXED ONES AND ZEROS */
```

Subscript Notation

Subscript expressions must contain either decimal constants or unsubscripted, unsubstringed variables. The subscript expression can contain 'exclusive or', 'or', 'and', plus, minus, or multiplication operators (&&,|,&,+,-,*) -- but not the division operator. Operators must appear in order of binding strength (from low to high). A subscript expression cannot contain a subset of parentheses. Examples of subscript form are:

```
DCL A(10) FIXED;
A(3) = 0;            /* LEGAL */
A(I) = 0;            /* LEGAL */
A(3+J) = 0;          /* LEGAL */
A(10-2*I) = 0;       /* LEGAL */
A(I&J&10*K)          /* LEGAL */
A(K+L+M+N*Q) = 0;    /* LEGAL */
A(K+L+M*N*Q) = 0;    /* LEGAL */
A(B->C) = 0;         /* LEGAL */
A(A(1)) = 0;         /* ILLEGAL BECAUSE SUBSCRIPT IS SUBSCRIPTED */
A(K/L) = 0;          /* ILLEGAL BECAUSE DIVISION OPERATOR IS USED */
A(2*I+1) = 0;        /* ILLEGAL BECAUSE OPERATORS ARE NOT IN ORDER OF
                                BINDING STRENGTH */
A(10-(2*I)) = 0;     /* ILLEGAL BECAUSE PARENTHESIS IS NOT ALLOWED IN
                                SUBSCRIPT */
```

A bit string item that is not on a byte boundary or is not a multiple of eight bits cannot be used as a subscript.

The control variable of a DO statement cannot be subscripted. An example is:

```
DCL A(10) FIXED;
 DO A(2) = 1 TO 10;   /* ILLEGAL CONTROL VARIABLE */
```

In a declaration, a locating expression that is associated with a BASED item cannot be subscripted or substringed. In a reference statement, a pointer that is associated with a BASED item cannot be subscripted. Examples are:

```
DCL X CHAR(3), P PTR;
DCL CAT CHAR(1) BASED(P);           /* LEGAL */
DCL FAT CHAR(2) BASED(P+1);         /* LEGAL */
DCL RAT CHAR(4) BASED(72);          /* LEGAL */
DCL BAT CHAR(3) BASED(ADDR(FAT));   /* LEGAL */
DCL HAT CHAR(3) BASED(ADDR(X)+1);   /* LEGAL */
DCL MAT CHAR(3) BASED(ADDR(X(1)));  /* ILLEGAL */
DCL P(10) PTR, R BASED(P(1));       /* ILLEGAL */
P(1)->R = 3;                        /* ILLEGAL */
```

Substring Notation

Substring expressions must contain either constants or unsubscripted, unsubstringed variables. The substring expression can contain 'exclusive or','or','and', plus, minus, or multiplication operators(&&,,&,+,-, *)-- but not the division operator. Operators must appear in order of binding strength (from low to high). A substring expression cannot contain a subset of parentheses. Examples of substring form are:

```
DCL CHAR30   CHAR(30);
DCL A CHAR(10);
A(3) = '0';            /* LEGAL */
A(I) = '0';            /* LEGAL */
A(3+J) = '0';          /* LEGAL */
A(10-2*I) = '0';       /* LEGAL */
A(I&J&10*K) = '0';     /* LEGAL */
A(K+L+M+N*Q) = '0';    /* LEGAL */
A(K+L+M*N*Q) = '0';    /* LEGAL */
A(B-C) = '0';          /* LEGAL */
A(I:I+4)='xy';         /* LEGAL */
A(I+1:I+K)=CHAR30;     /* LEGAL */
A(A(1)) = '0';         /* ILLEGAL BECAUSE SUBSTRING IS SUBSTRINGED */
A(K/L) = '0';          /* ILLEGAL BECAUSE DIVISION OPERATOR IS USED */
A(2*I+1) = '0';        /* ILLEGAL BECAUSE OPERATORS ARE NOT IN ORDER
                          OF BINDING STRENGTH */
A(10-(2*I)) = '0';     /* ILLEGAL BECAUSE PARENTHESES ARE NOT ALLOWED
                          IN SUBSTRING */
```

A bit string item that is not on a byte boundary or is not a multiple of eight bits cannot be used as a substring.

A bit string cannot be substringed with a variable that specifies a single bit. For example:

```
DCL B Bit(32);
 B(I)= '1'B         /* ILLEGAL */
```

An arithmetic or pointer item cannot be substringed. An example is:

```
DCL (A,B) FIXED;
 A = B(2);          /* ILLEGAL SUBSTRING */
```

A string constant of one byte cannot be assigned to a substring with a variable range. For example:

```
DCL A CHAR(6);
 A (I:J) = 'B';     /* ILLEGAL */
```

<u>Note</u>: If a bit variable has a variable substring range the compiler assumes, but does not check to assure, that the lower bound specifies the first bit of a byte and the upper bound specifies the last bit of a byte.

The compiler assumes that the range of the substring notation does not exceed the range of its associated string item. However, there is no enforced restriction to this effect, and the compiler does not provide code to check the range. In the following example:

```
DCL A CHAR(6);
 A (I:J) = X;
```

the compiler does <u>not</u> provide code to check that:

- I is equal to, or greater than one.
- J is equal to, or less than six.
- I is equal to, or less than J.
- J minus I, plus one, is not greater than the length of X.

Section 6: Compiler-Generated Code

This section contains examples of the code generated by the current BSL compiler to manipulate various kinds of data. (Future compilers may not generate the same code.)

It is provided for two reasons:

- With this information the user can obtain insight to produce better and more efficient code.

- The user will have closer control of the hardware and software interfaces.

This section is organized to provide easy access to examples of different data. It illustrates how changing the attributes of data affects the way it is accessed. Included in the section is information on arithmetic and string expressions.

Arithmetic Expressions

Unless otherwise noted the following conventions are used in the example of arithmetic expressions:

- A, B, C, etc. represent FIXED(31) variables.
- Fn represents a fixed item with precision n.
- Pn represents a pointer with precision n.
- @r and @r1 represent any available registers.

Register Usage

The BSL compiler always uses register 15 for evaluating arithmetic expressions. Register 14 will be used with register 15 when an even/odd pair of registers is required. For example:

```
A=B*C+D;
    L       @F,C
    M       @E,B
    A       @F,D
    ST      @F,A
```

For expressions containing a subscripted, substringed, based or non-local variable, another register will be used to compute the address of the variable. The variable will then be used as if no address calculation was necessary. For example:

```
DCL P PTR, B BASED (P);
DCL C(10) FIXED;
A=B + C(I);
    L       @r,I             COMPUTING THE ADDRESS
    SLA     @r,2               OF C(I)
    L       @F,C-4(@r)       LOAD C(I)
    L       @r1,P            ADDRESS OF B
    A       @F,0(0,@r1)      USE OF B
    ST      @F,A
```

If a register is available, it will be used for evaluating parts of an arithmetic expression when register 15 already has a partial result. When no register is available, register 15 will be stored in a compiler generated temporary and evaluation will continue in register 15. For example:

```
A=B+C & D+E;
   L        @F,C
   A        @F,B
   L        @r,E
   A        @r,D
   NR       @F,@r
   ST       @F,A
```

Compiler Generated Temporaries

Temporaries are generated to hold partial results of arithmetic expressions when no register is available to continue the evaluation. The code shown above would compile as follows if no register were available:

```
A=B+C & D+E;
   L        @F,C
   A        @F,B
   ST       @F,@T1
   L        @F,E
   A        @F,D
   N        @F,@T1
   ST       @F,A
```

Unsigned data (CHAR, BIT, or PTR) that occupies two or three bytes will be moved to a compiler-generated temporary with high order zeros before it is used. This is to eliminate the sign bit from the operation. For example:

```
DCL P16 PTR(16);
A=B + P16;
   MVC      @TEMP2+2(2), P16
   L        @F,@TEMP2
   A        @F,B
   ST       @F,A
```

A temporary will be used to align data that is not on an appropriate boundary. For example:

```
DCL F31 FIXED(31) BDY(WORD,2);
A=B + F31;
   MVC      @T4(4),F31
   L        @F,@T4
   A        @F,B
   ST       @F,A
```

Fixed Data

FIXED(31) and FIXED(15) data items used in arithmetic statements are normally manipulated with fullword or halfword RX instructions. For example:

```
F31=F31 + F15;
   LH       @F,F15
   A        @F,F31
   ST       @F,F31
```

FIXED items that are not on appropriate boundaries are moved to compiler-generated temporaries that are on appropriate boundaries, and the temporaries are manipulated with RX instructions. For example:

```
DCL F31 FIXED(31) BDY (WORD,2);
A=B + F31:
   MVC      @T4(4),F31
   L        @F,@T4
   A        @F,B
   ST       @F,A
```

FIXED items with register storage class are manipulated with RR instructions. For example:

```
DCL R2 REG(2);
A=B + R2;
   LR       @F,@2
   A        @F,B
   ST       @F,A
```

Pointer data

The way a pointer is accessed depends on its precision, boundary, and storage class. The following examples show the code that would be generated to access pointers of various precisions with default boundaries, when used as pointers and as arithmetic items.

```
 1. A=P8-> 7B;
      SR      @r,@r
      IC      @r,P8
      L       @F,0(0,@r)
      ST      @F,A
 2. A=P8;
      SR      @F,@F
      IC      @F,P8
      ST      @F,A
 3. A=P15-> B;
      LH      @r,P15
      L       @F,0(0,@r)
      ST      F,A
 4. A=P15;
      LH      @F,P15
      ST      @F,A
 5. A=P16-> B;
      MVC     @TEMP2+2(2),P16
      L       @r,@TEMP2
      L       @F,0(0,@r)
      ST      @F,A
 6. A=P16;
      MVC     @TEMP2+2(2),P16
      L       @F,@TEMP2
      ST      @F,A
 7. A=P24-> B;
      L       @r,P24-1
      L       @F,0(0,@r)
      ST      @F,A
 8. A=P24;
      MVC     @TEMP1+1(3),P24
      L       @F,TEMP1
      ST      @F,A
 9. A=P31-> B;      /* THE SAME CODE IS GENERATED FOR P32 */
      L       @r,P31
      L       @F,0(0,@r)
      ST      @F,A
10. A=P31;          /* THE SAME CODE IS GENERATED FOR P32 */
      L       @F,P31
      ST      @F,A
```

A pointer with other than a default boundary will be moved to a temporary with the appropriate boundary, and the temporary will be accessed with the same sequence of code generated for precisions 31 and 32. For example:

```
DCL P15 PTR(15) BDY (WORD,2);
A=P15-> B;
   MVC      @TEMP2+2(2),P15
   L        @r,@TEMP2
   L        @F,0(0,@r)
   ST       @F,A
```

A pointer with register storage class will generate the same sequence of code without loading the pointer into a register. For example:

```
DCL P24 PTR REG(5);
A=P24 -> B;
    L         @F,0(0,@5)
    ST        @F,A
          or
A=P24;
    ST        @5,A
```

A pointer that is a parameter, a nonlocal pointer, or a based pointer will generate the same sequence of code with additional code to locate the pointer. For example:

```
DCL P15 PTR(15);
DCL P8 PTR(8) BASED (P15);
A=P8;
    LH        @r,P15            ADDRESS OF P8
    SR        @F,@F
    IC        @F,0(0,@r)        P8
    ST        @F,A
```

String Data

String data is unsigned. Whenever possible the BSL compiler generates code to prevent the high order bit from being treated as a sign.

The sign is eliminated from one byte items by using the SR-IC sequence. For example:

```
A=C1;
    SR        @F,@F
    IC        @F,C1
    ST        @F,A
```

For two and three byte items a compiler generated temporary with high order zeros is used. For example:

```
A=B+C2;
    MVC       @TEMP2+2(2),C2
    L         @F,C2
    A         @F,B
    ST        @F,A
```

Logical instructions are used whenever possible for four byte items. For example:

```
A=C4+B;
    L         @F,B
    AL        @F,C4
    ST        @F,A
```

String items longer than four bytes are not allowed in arithmetic expressions.

String Expressions

No registers are involved in the evaluation of a string expression, (except for calculating the addresses of operands that are subscripted, substringed, based, or nonlocal). The expression will either be evaluated in a temporary or in the receiver of an assignment statement.

A temporary will be used for string expressions in IF statements, and for assignment expressions that include the variable that will have the result of the expression assigned to it. For example:

```
DCL (A,B,C,D) CHAR(4);
IF(A&B)=(C&D) THEN...
   MVC      @CTEMP1(4),B
   NC       @CTEMP1(4),A
   MVC      @CTEMP2(4),D
   NC       @CTEMP2(4),C
   CLC      @CTEMP1(4),@CTEMP2
        and
A=B&A&C;
   MVC      @CTEMP1(4),C
   NC       @CTEMP1(4),A
   NC       @CTEMP1(4),B
   MVC      A(4),@CTEMP1
```

The receiver of an assignment statement will be used when it does not appear in the assignment expression, or it appears in the expression in such a way that it can be used first when:

```
A=B&C;          /* RECEIVER DOES NOT APPEAR */
   MVC      A(4),C
   NC       A(4),B
        or
A=B&A;          /* CAN BE USED FIRST */
   NC       A(4),B
```

Expressions with variable length strings are evaluated using the same criteria for deciding whether to use the assignment receiver or a temporary. Whatever is used, it will be zeroed (or blanked) before it is used to insure that shorter operands will be properly extended. Registers 10,14, and two other registers are used in all operations with variable-length strings. For example:

```
A=B&C(I:J);
   L        @r,J
   MVI      A,C'b'              BLANK A IN CASE C(I:J)
   MVI      A+1(3),A            IS SHORTER THAN A
   L        @r1,I
   LA       @E,C-1(@r1)
   SR       @r,@rl
   BCTR     @r,0
   LA       @A,A
   EX       @r,@MVC
   NC       A(4),B
```

Section 7: BSL Programming Techniques

This section describes a variety of programming techniques that you can use to:

- Format your source program.
- Handle parameters and argument lists.
- Improve the compiler-generated code.
- Manipulate data.

Formatting the Source Program

The modular arrangement of your program affects the efficiency of its operation. The appearance of your program -- the way your source statements are set up -- is an important debugging tool. Both the arrangement and the appearance are factors in making future changes and analyses.

Use of External Procedures

Invoking a procedure generates more instructions than branching to a label; therefore, you should not write a separate procedure for an operation that can conveniently be placed in line.

You may find it convenient to modularize a programming task by breaking it up into a number of separately compiled procedures. However, this convenience should be balanced against the resulting call, prologue, and epilogue overhead. If you modularize your program, keep down the overhead by carefully planning the breaks between procedures.

Organization of Source Statements

The following guide lines[1] may help you produce a source program listing that is easy to read, easy to understand, and easy to use:

- Avoid placing many BSL statements on a single card. This can cause difficulty when it is necessary to change one statement on the card.
- Adopt specific conventions about the placement of comments in your source statements.
- Group your DECLARE statements at the beginning of the procedure.
- Indent each DO statement in a nest of DO statements, and line up each END statement under its associated DO statement. This ensures that you properly close each DO group. Also, indent and align all statements applicable to each DO statement.
- When you write nested IF statements, adopt a specific convention to associate each IF statement with its ELSE clause. A suggested method is to indent each IF statement in the nest, and line up its ELSE clause under it.

----

[1] Additional information on guide lines is available in BSL Bulletin Number Four or in the Design Summary Book Section 3.50.17.

Handling Argument Lists and Parameters

Argument lists and parameters are the means of communication between procedures. The following paragraphs discuss a few techniques for handling this communication.

Building an Argument List

You can let the compiler generate code for building the necessary argument lists, or you can build your own. There are two conditions under which you may want to build your own argument lists:

1. If a particular argument list is used many times, you can save space by building it yourself. Otherwise, the BSL compiler builds a separate argument list for each CALL statement in your program.

2. You may need an argument list that is different from the one generated by the compiler.

The following example shows how to build your own argument list for calling the PDUMP routine:

```
/* THIS BUILDS AN ARGUMENT LIST FOR PDUMP */
DCL L1 CHAR(1) INIT('A');
DCL A CHAR(50);
DCL L2 CHAR(1) INIT('*');
DCL LEN FIXED INIT(50);
DCL 1 ARG,
      2 TYPE PTR(31) INIT(ADDR(L1)),
      2 ADCON PTR(31) INIT(ADDR(A)),
      2 LENGTH PTR(31) INIT(ADDR(LEN)),
      2 STOP PTR(31) INIT(ADDR(L2));
/* ARGUMENT LIST DEFINED AS A STRUCTURE */
 RESTRICT(1);
 DCL R1 REG(1) PTR(31);
     R1 =ADDR(ARG); /* POINT R1 TO ARG LIST */
 CALL PDUMP;
    .
    .
    .
 CALL PDUMP;
```

TECH

Passing an Argument List in a Register

To pass an argument in a register, you must set the register in the calling procedure, and preserve its value on entry to the called procedure. This example shows how:

```
X:  PROC;
    DCL TAB CHAR(80), R3 REG(3) PTR(31);
    RESTRICT(3);
    R3 = ADDR(TAB);  /* SET R3 TO POINT TO TAB */
    CALL IN;         /* INVOKE PROCEDURE */
      .
      .
      .
IN: PROC;
    DCL FAB CHAR(80) BASED(R3); /* FAB BASED ON R3 */
    RESTRICT(3);
    IF FAB(1) = '$' THEN DO;
      .
      .
      .
    END IN;
END X;
```

How to Avoid Passing Arguments

For performance reasons, it is sometimes desirable to avoid passing arguments to procedures. There are two ways to do this:

1. An internal procedure can directly access all the data declared in the external procedure.

2. If an external procedure is called and the data to be communicated is static, declare the data LOCAL EXTERNAL in the calling procedure and NONLOCAL EXTERNAL in the called procedure.

Variable Parameter Lists

The BSL language does not support variable length parameter lists, but you can handle them with the following method:

1. The calling procedure declares the procedure having the variable parameter list with OPTIONS(VLIST) and calls it with the standard CALL statement.

2. The called procedure defines its own parameter list and searches for the end of the list.

This method is illustrated in the following example:

```
CALLPROC: PROC;
  .
  .
DCL VPLIST ENTRY OPTIONS(VLIST);
CALL VPLIST (A1,A2);
END CALLPROC;

VPLIST:  PROC;  /* NO FORMAL PARAMETERS */
   DCL R1 REG(1)PTR; /* R1 POINTS TO THE ARGUMENT LIST */
   RESTRICT(1);
   /* IF THIS PROCEDURE WILL CHANGE REGISTER 1 -- BY */
   /* GENERATING A SYSTEM MACRO FOR EXAMPLE -- THEN */
   /* A COPY OF REGISTER 1 SHOULD BE USED INSTEAD OF */
   /* REGISTER 1 ITSELF. */
   DCL /* DESCRIPTION OF ONE OF N SETS OF ARGUMENTS */
      1 ARG BASED(R1),
            2 AFIRST PTR,
            2 ASECOND PTR,
L1:/* PROCESS ONE SET OF ARGUMENTS */
      .
      .
      .
   IF ASECOND <0 THEN /* IS THIS THE FINAL ARGUMENT */
      RETURN;   /* YES */
   R1=R1+8;  /* SET POINTER TO NEXT SET OF ARGUMENTS */
   GOTO L1; /* PROCESS NEXT SET OF ARGUMENTS */
   END VPLIST;
```

Avoiding Parameter References

The use of parameters in a BSL procedure is both a programming convenience and a documentation aid. As with most conveniences, there is a cost:

1. Register 1 must be used for all parameter references. This means register 1 must be saved and restored around GENERATE statements that generate system macros or instructions that change register 1.

2. The code generated to refer to a parameter is less efficient than the code generated to refer to a local variable.

If you want your BSL object program to be competitive with an object program written in assembly language, you may wish to avoid the use of parameters. A good way to avoid referring to parameters is to make a local copy of the parameters at the start of the procedure. For example:

```
A: PROC(P1,P2);
  DCL P1 CHAR(4);
  DCL P1COPY CHAR(4);
  DCL P2(10) CHAR(256);
  DCL P2PTR REG(6) PTR;
  RESTRICT(6);
  DCL P2COPY(10) CHAR(256) BASED(P2PTR);
  P1COPY = P1;
  P2PTR = ADDR(P2);
```

In this example P1 is a scalar variable that is copied to P1COPY. Subsequent references to P1 should be made by using P1COPY.

P2 is an array that is too large to copy, so P2COPY is declared with the attributes of P2 and based on a pointer that is set to the address of P2. References to P2COPY will now be references to P2.

Setting a Return Code

To set a return code, you place a value in register 15 before returning control to the caller. For example:

```
X: PROC OPTIONS(DONTSAVE(15));
       /* REG 15 IS NOT SAVED OR RESTORED */
   DCL R15 REG(15) FIXED(31);
   R15 = 20;  /* SETS REG 15 TO VALUE OF 20 */
   RETURN;    /* RETURNS TO CALLER */
END;
```

If another BSL procedure receives the return code, the code cannot be tested while it is in register 15. Since the compiler uses register 15 to evaluate expressions, you must assign register 15 to another variable before _anything_ else is done. For example:

```
DCL RETCODE FIXED(15);
DCL R15 REG(15);
 .
 .
 .
CALL X;
RETCODE = R15;
 .
 .
 .
```

Note: In a reentrant procedure, or whenever the TRACE option is used, you may not place the return code directly into register 15. Instead, you must place the return code in the register 15 field of the calling program's save area. For example:

```
DCL  R13 PTR REG(13);
DCL  A PTR BASED(R13+4);
DCL  RETCODE BASED(A+16);
RETCODE = 20;
```

Obtaining the Optimum Code

The following paragraphs discuss various techniques that you can use to favorably affect the compiler-generated code.

Putting Variables in Registers

A good technique to use in a BSL program to improve speed and save space is to place certain key variables in registers. The compiler's cross-reference listing shows you which variables are used most often.

If you place variables in registers, you must RESTRICT those registers to prevent the compiler from generating conflicting register usage. You should also note that the RESTRICT statement does not carry into internal procedures.

When you make an assignment to a register variable, there are two special cases that cause the compiler to generate optimized code. This optimization occurs only when the statements are written in one of the formats described below.

Case 1: The statement format is REGPTR24 = REG + CONST; where REGPTR24 is a register variable declared as POINTER(24), and REG is a register variable, and CONST is a constant less than 4096. For example:

```
DCL R6 REG(6) PTR(24);
DCL R7 REG(7);
R6 = R6 + 7;          /* SPECIAL CASE*/
R6 = R7 + 12;         /* SPECIAL CASE */
```

Case 2: The statement format is REG = EXP; where REG is a register variable, and EXP is an expression that contains no subscript or substring, and no operators other than +, -, &, |, or &&. REG must not be one of the operands in the expression. For example:

```
DCL R7 REG(7);
R7 = A + B - 3        /* SPECIAL CASE */
R7 = A + R7 - 3;      /* NOT A SPECIAL CASE */
R7 = A(I) + B - 3;    /* NOT A SPECIAL CASE */
```

Eliminating Compiler-Generated Temporary Locations

There are certain programming practices that require the compiler to generate extra code to move items to temporary locations. If you eliminate the need for these temporary locations, you can save time and storage space in program execution.

When a pointer or arithmetic item is not on a proper boundary, the compiler generates code to move the item to a properly aligned temporary location, and loads it from there. To avoid this, place arithmetic items and pointers on their default boundaries, which are as follows:

Attribute	Best Boundary
FIXED(15)	Halfword
FIXED(31)	Fullword
PTR(16)	Halfword
PTR(31)	Fullword
PTR(32)	Fullword

When character substrings are used in arithmetic expressions, the compiler generates code to move the item to a temporary location, and loads it from there. To avoid this, do not use character substrings in arithmetic expression.

When a PTR(16) item is used in an arithmetic context, the pointer must be moved to a temporary location to avoid propagating the high-order bit. The arithmetic evaluation is then done from the temporary location. To avoid this, declare the item as PTR(15), unless you really need a 16-bit pointer.

When a PTR(24) item is used in a arithmetic context, the pointer must be moved to a temporary location so that the high-order byte will not affect the result. The arithmetic evaluation is then done from the temporary location. To avoid this, declare the item as PTR(31), unless you actually use the high-order byte for some other purpose.

Improving Iterative DO Loops

The compiler generates the most efficient DO loop code when you write your DO loops according to the guide lines presented in the following paragraphs.

The control variable, the TO value, and the BY value of the DO loop statement should be simple variables that do not have any of the following attributes:

BASED
PARAMETER
EXTERNAL
Dimension (subscripted)

If possible, the control variable should be declared in a register.

If a DO statement is only for counting the number of iterations through a loop, write it as:

```
DO REG = EXP TO 1 BY -1;
     generates
BCT REG, start of loop
```

where REG is a register variable, and EXP is any expression. Note that if REG is set negative or zero in the loop, the iteration continues even though REG is less than 1.

Location Free Code

You can obtain location free code by avoiding BSL language that results in the generation of address constants. Currently, the compiler generates address constants for:

- NONLOCAL EXTERNAL items.
- POINTER items with initial values.
- Argument lists in non-reentrant procedures.

The only way you can eliminate address constants for NONLOCAL EXTERNAL items is by not using such items.

You can eliminate address constants for POINTER items with initial values by not declaring the INITIAL attribute. Instead, you can initialize the items with assignment statements.

The only way you can eliminate address constants for argument lists in a non-reentrant procedure is by not using such argument lists.

In addition to the elimination of address constants, you must follow self-relocating coding practices as described in the DOS Assembler Manual publication (Form C24-3414). Thus, pointers kept across interrupt points would have to be revalidated after a possible code move.

Data Manipulation and Reference

The following paragraphs offer some programming techniques that you can use to handle data in your BSL procedure.

Using the Same Area in Different Ways

It is sometimes useful to refer to parts of the same storage area in different ways. Two methods of doing this are described below:

Case 1: You can declare the area as a structure. For example:

```
DCL 1 F FIXED(31),
        2 L  FIXED(15),
        2 R  FIXED(15),
           3 * CHAR(1),
           3 B BIT(8);
```

Case 2: You can base items on the address of a common area. This is similar to the EQUIVALENCE feature in FORTRAN. For example:

```
DCL F FIXED(31);
DCL L FIXED(15) BASED(ADDR(F));
   /* LEFT 2 BYTES OF F */
DCL R FIXED(15) BASED(ADDR(F)+2);
   /* RIGHT 2 BYTES OF F */
DCL B BIT(8) BASED(ADDR(F)+3);
   /* TREAT LOW-ORDER BYTE OF F AS BIT 8 */
```

Scanning a String of Bits

If you want to scan a string of bits to find the first nonzero bit, consider the method shown in the following example:

```
DCL FLAGS BIT(32), MASK BIT(32);
 MASK = '7FFFFFFF'X;
 DO I = 1 TO 32;
 IF FLAGS > MASK THEN GOTO OUT;
 ELSE MASK = MASK/2;
  /* SHIFT MASK ONE BIT RIGHT TO SELECT NEXT BIT */
 END;
```

Propagating a Character

If you want to propagate a given character through a string, an efficient method is shown in the following example:

```
DCL A CHAR(50);
 A(1) = 'Z';         /* PUTS CHARACTER Z INTO BYTE A(1) */
 A(2:50) = A(1:49); /* FILLS REST OF A WITH CHARACTER Z */
```

Setting an Area to Zero

If you have an area which is greater than four bytes in length and a multiple of eight bits, you can set the entire area to zeros with the following technique:

```
DCL A(10) FIXED(31); /* AREA TO BE CLEARED */
DCL B BIT(320) BASED(ADDR(A)); /* DESCRIBE AREA AS A BIT STRING */
 B = ''B;  /* THIS CAUSES THE CONSTANT TO BE PADDED RIGHT WITH 320 BIT
              ZEROS.  SINCE A IS EQUIVALENT TO B, A IS SET TO ZEROS */
```

Searching a Table

Your BSL programs may frequently need to search a table to match a given argument, and then branch to a particular label. Two techniques for doing this are described below.

Case 1: If the search arguments are in a small, ordered range, you may use the technique shown in this example:

```
        /* I IS THE ARGUMENT */
        /* I IS FIXED(31) */
        /* THE VALUE OF I MAY BE 1 TO 10 */
DCL (L1,L2,L3,L4,L5,L6,L7,L8,L9,L10) LABEL;
        /* DECLARE BRANCH TABLE POINTS */
DCL BRANT(10) PTR(31) INIT(ADDR(L1),ADDR(L2),ADDR(L3),
    ADDR(L4),ADDR(L5),ADDR(L6),ADDR(L7),ADDR(L8),
    ADDR(L9),ADDR(L10));
 GO TO BRANT(I);
 /* THIS TRANSFERS CONTROL TO THE REQUIRED POINT */
```

Case 2: If the search arguments are not in a small, ordered range, you may use the technique shown in this example:

```
        /* Q IS THE ARGUMENT */
        /* Q IS CHAR(1) */
        /* Q CAN HAVE NINE HEX VALUES */
DCL Q CHAR(1);
DCL (L1,L2,L3,L4,L5,L6,L7,L8,L9) LABEL;
        /* REQUIRED BRANCH POINTS */
DCL 1 BTAB(9) BDY(WORD),
      2 ARG CHAR(1) INIT('F3'X,'91'X,'29'X,'33'X,'52'X,'81'X,'00'X,
             'CC'X,'FF'X),
        /* ALLOWABLE VALUES OF ARGUMENT */
      2 BPT PTR(24) INIT(ADDR(L1),ADDR(L2),ADDR(L3),
             ADDR(L4),ADDR(L5),ADDR(L6),ADDR(L7),ADDR(L8),
             ADDR(L9));
        /* BRANCH POINTS FOR EACH CODE */
 DO J = 1 TO 9 BY 1;
 IF  ARG(J) = Q THEN GOTO BPT(J);
        /* BRANCH TO PROPER ROUTINE */
 END;
        /* IF YOU FALL OUT OF LOOP TO THIS POINT THEN Q DID NOT MATCH
           ANY ARG IN THE TABLE */
GOTO ERROR;
```

Using the GENERATE Statement

There are two forms of the GENERATE statement in BSL:

- The simple GENERATE maps the contents of the GENERATE statement into the object margin (see GENMGIN compiler option) of the output assembler text starting in column 10. The simple GENERATE is limited to a single card.

- The block GENERATE maps a series of cards into the object margin (see GENMGIN compiler option) of the output assembler text (the source margin specification is ignored). The end of a block GENERATE is indicated by a special control statement ($ENDGEN).

In practice, you should use the simple GENERATE to generate a single instruction or macro instruction; and use the block GENERATE to generate a number of lines of executable code or data items. Another consideration is the mapping of the output -- the block GENERATE allows you to insert text in column one of the output; the simple GENERATE does not allow this.

Note: To provide proper instruction alignment, the compiler adds a DS 0H instruction following every simple GENERATE statement and those block GENERATE statements which are not of the form GENERATE DATA.

If you include a data item in the text of a block GENERATE statement which is not of the form GENERATE DATA, you may need to include code to branch around the data item. In the GENERATE DATA statement, however, no branch code is necessary.

If a variable is defined in a GENERATE statement and is referred to in both the GENERATE statements and the BSL source statements, you must declare the variable as GENERATED. This attribute shows the compiler that the named variable is defined in a GENERATE statement and, therefore, need not be redefined in BSL.

Appendix A contains a detailed listing of a program that uses GENERATE statements. The form of the GENERATE statement is described in the BSL Language Specification manual.

The Compile-Time Processor and the GENERATE Statement

When the BSL compile-time processor substitutes a value for a compile-time variable, the replacement value is preceded by and followed by a blank. This can result in an illegal assembly language statement if the compile-time variable is in a GENERATE statement. The problem can be avoided in one of the following two ways:

1. Stop the compile-time processor from scanning the GENERATE statement by making it look like a comment during compile-time processing. For example:

```
     GEN; /*
       .
       .
       .
    $ENDGEN */
```

   After reading a 'GEN;' or a '$ENDGEN', the BSL compiler ignores the remainder of the card. The compile-time processor will not make substitutions in a comment. Therefore, the compile-time processor has been stopped from changing the text of the GENERATE statement, without changing the way that the BSL compiler handles the GENERATE statement.

2. Define the compile-time variable in such a way that blanks will be legal after the substitution is made. For example, if you would like to write:

   ```
   GEN (Lƀƀƀƀƀ15,CTVBL);
   ```

   and have the compile-time processor replace CTVBL by its value, it could be done as follows:

```
        % DCL CTVBL CHAR;
        % DCL CODE CHAR;
        % CTVBL = 'ACTNAME';
        % CODE ='Lƀƀƀƀƀ15,'||CTVBL;
        GEN (CODE);
```

This will result in:

```
        GEN (ƀLƀƀƀƀ15,ACTNAME );
```

Program Parameterization

Example 1 shows a sample BSL program to read a card and then print it. The program could have been written to allow either the input card buffer (INSZ) or the output line buffer (OUTSZ) to be any length for a given run. The program could also have been written to allow source text from a library to be included at points within the source program during compilation. And the program could have been written to determine how many input records could fit on an output line, and compile only the code to print that number of records.

The sample program should have been parameterized (generalized) when first written by using compile-time macro facilities to do all these things. This would provide a more flexible, functional program. The program could be used as an application with many different values, rather than only once as it stands in its original form.

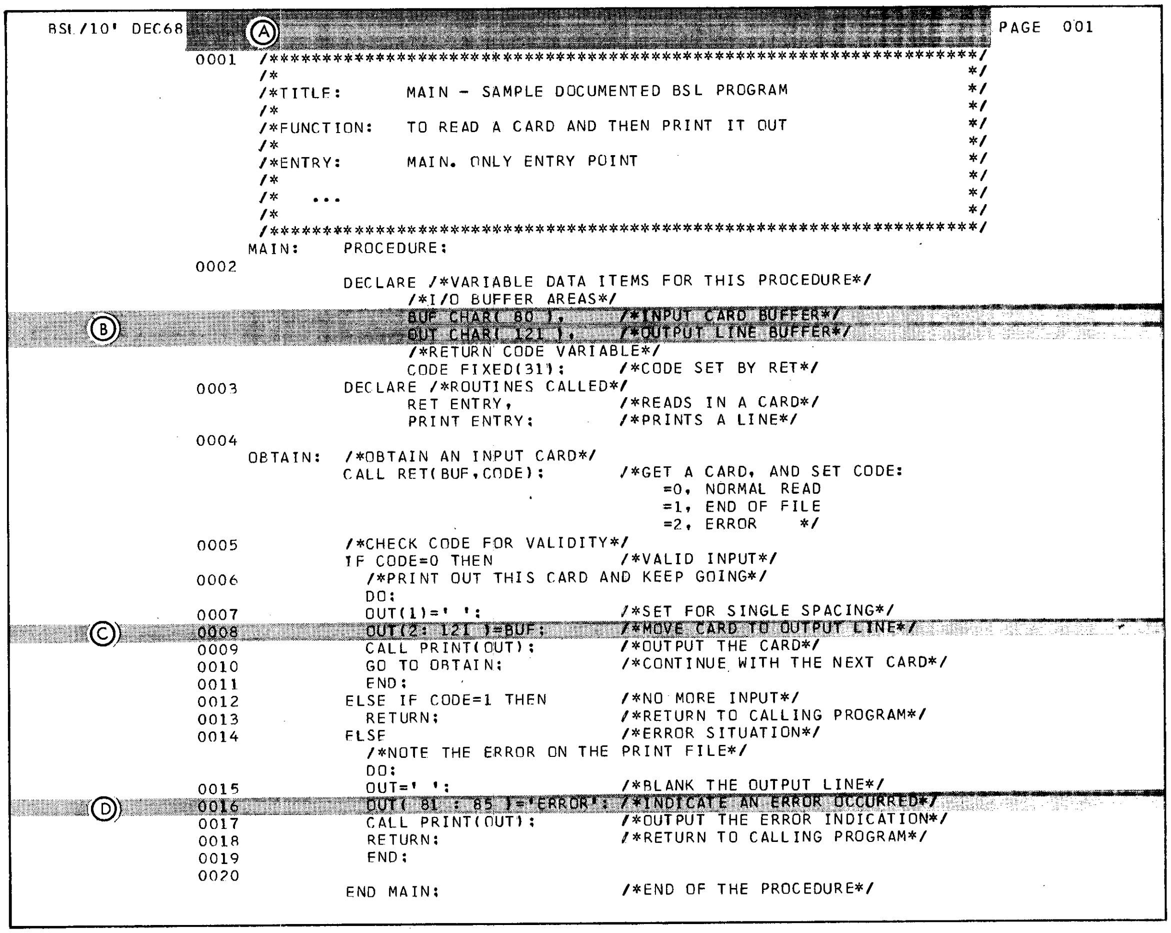
Remember, this generalizing takes place at compile-time, not at object time. In the one compilation, two distinct phases occur. These are:

- MACRO PHASE: The compile-time processor modifies the source text according to user instructions.

- COMPILE PHASE: The compiler translates the modified source text into assembler text.

Using the sample program in Example 1 as a base, inserts can be added to generalize the program. (The place for each insert is noted by shading.)

• Example 1. BSL Program to Read a Card and Print It



```
  BSL/10' DEC68                                                                        PAGE  001
            0001  /*******************************************************************/
                  /*                                                                 */
                  /*TITLE:       MAIN - SAMPLE DOCUMENTED BSL PROGRAM                */
                  /*                                                                 */
                  /*FUNCTION:    TO READ A CARD AND THEN PRINT IT OUT                */
                  /*                                                                 */
                  /*ENTRY:       MAIN. ONLY ENTRY POINT                              */
                  /*                                                                 */
                  /*    ...                                                          */
                  /*                                                                 */
                  /*******************************************************************/
                 MAIN:    PROCEDURE:
            0002
                          DECLARE /*VARIABLE DATA ITEMS FOR THIS PROCEDURE*/
                                /*I/O BUFFER AREAS*/
                                BUF CHAR( 80 ),         /*INPUT CARD BUFFER*/
                                OUT CHAR( 121 ),        /*OUTPUT LINE BUFFER*/
                                /*RETURN CODE VARIABLE*/
                                CODE FIXED(31):         /*CODE SET BY RET*/
            0003          DECLARE /*ROUTINES CALLED*/
                                RET ENTRY,              /*READS IN A CARD*/
                                PRINT ENTRY:            /*PRINTS A LINE*/
            0004
                 OBTAIN:  /*OBTAIN AN INPUT CARD*/
                          CALL RET(BUF,CODE):           /*GET A CARD, AND SET CODE:
                                                              =0, NORMAL READ
                                                              =1, END OF FILE
                                                              =2, ERROR    */
            0005          /*CHECK CODE FOR VALIDITY*/
                          IF CODE=0 THEN                /*VALID INPUT*/
            0006            /*PRINT OUT THIS CARD AND KEEP GOING*/
                            DO:
            0007            OUT(1)=' ':                 /*SET FOR SINGLE SPACING*/
            0008            OUT(2: 121 )=BUF:           /*MOVE CARD TO OUTPUT LINE*/
            0009            CALL PRINT(OUT):            /*OUTPUT THE CARD*/
            0010            GO TO OBTAIN:               /*CONTINUE WITH THE NEXT CARD*/
            0011            END:
            0012          ELSE IF CODE=1 THEN           /*NO MORE INPUT*/
            0013            RETURN:                     /*RETURN TO CALLING PROGRAM*/
            0014          ELSE                          /*ERROR SITUATION*/
                            /*NOTE THE ERROR ON THE PRINT FILE*/
                            DO:
            0015            OUT=' ':                    /*BLANK THE OUTPUT LINE*/
            0016            OUT( 81 : 85 )='ERROR':     /*INDICATE AN ERROR OCCURRED*/
            0017            CALL PRINT(OUT):            /*OUTPUT THE ERROR INDICATION*/
            0018            RETURN:                     /*RETURN TO CALLING PROGRAM*/
            0019            END:
            0020
                          END MAIN:                     /*END OF THE PROCEDURE*/
```

```
Insert A

%  DECLARE                       /* COMPILE TIME VARIABLES */
       INSZ FIXED,               /* INPUT BUFFER SIZE */
       OUTSZ FIXED,              /* OUTPUT BUFFER SIZE */
       LECOL FIXED,              /* LEFT ERROR COLUMN */
       RECOL FIXED,              /* RIGHT ERROR COLUMN */
       ;
%  INSZ=80;
%  OUTSZ=121;
%  RECOL=INSZ+5;
   LECOL=RECOL-4                 /* SET LEFT ERROR COLUMN */;
   .
   .
   .
Insert B

BUF CHAR(INSZ),                  /* INPUT CARD BUFFER */
OUT CHAR(OUTSZ),                 /* OUTPUT LINE BUFFER */
   .
   .
   .
Insert C

OUT(2:OUTSZ)=BUF;                /* MOVE CARD TO OUTPUT LINE */
   .
   .
   .
Insert D

OUT(LECOL:RECOL)='ERROR';        /* INDICATE AN ERROR OCCURRED */
   .
   .
   .
```

Changing Buffer Sizes

The inserts A, B, C, and D generalize the buffer sizes as well as the position where the word 'ERROR' will appear in the output line.

Insert A comes immediately after the compiler control statement containing the MACRO option. This insert does several things:

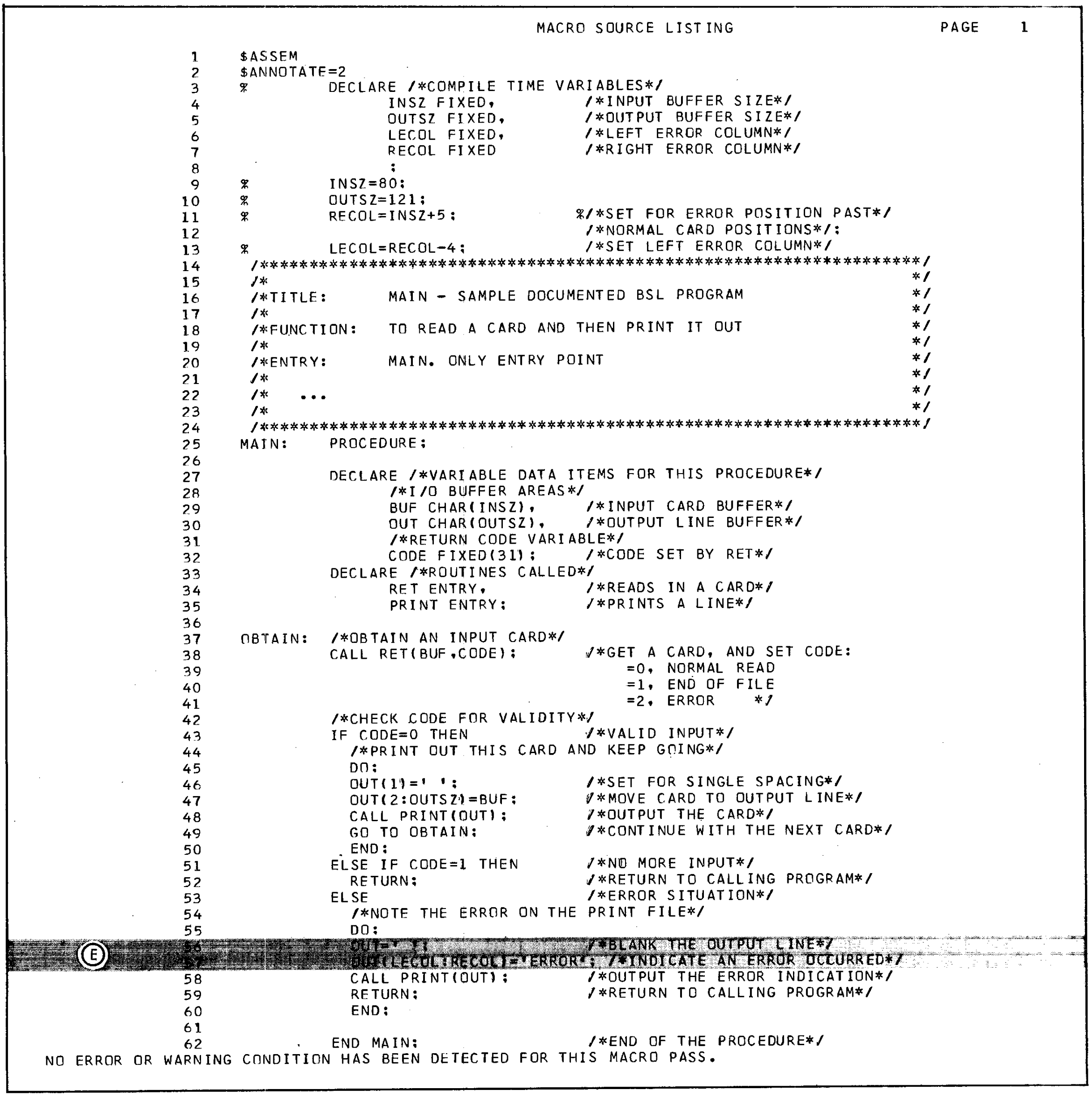
- The DECLARE statements symbolically name the general items - buffer lengths, right and left error column indicators.

- The assignment statements give numerical values to the buffer lengths and the right and left error column indicators.

Inserts B, C, and D replace the original numerical values with variable names. Example 2, a macro source listing, illustrates the placement of the inserts.

The resulting program is of far greater value than the original. This generalized program may be used for this application a number of times with different numerical values for buffer lengths.

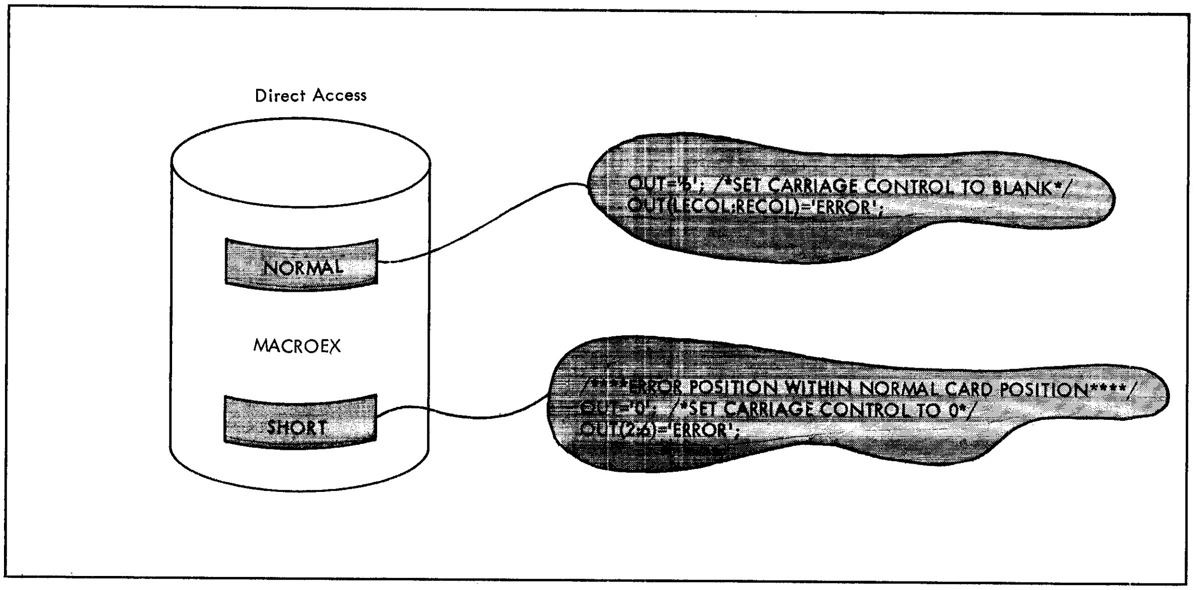
• Example 2. BSL Macro Source Listing (Changing Buffer Sizes)



```
                                                 MACRO SOURCE LISTING                     PAGE    1
                    $ASSEM
                    $ANNOTATE=2
                    %        DECLARE /*COMPILE TIME VARIABLES*/
                                   INSZ FIXED,        /*INPUT BUFFER SIZE*/
                                   OUTSZ FIXED,       /*OUTPUT BUFFER SIZE*/
                                   LECOL FIXED,       /*LEFT ERROR COLUMN*/
                                   RECOL FIXED        /*RIGHT ERROR COLUMN*/
                                   ;
                    %        INSZ=80;
                    %        OUTSZ=121;
                    %        RECOL=INSZ+5;           %/*SET FOR ERROR POSITION PAST*/
                                                      /*NORMAL CARD POSITIONS*/;
                    %        LECOL=RECOL-4;           /*SET LEFT ERROR COLUMN*/
                     /*******************************************************************/
                     /*                                                                 */
                     /*TITLE:      MAIN - SAMPLE DOCUMENTED BSL PROGRAM                 */
                     /*                                                                 */
                     /*FUNCTION:   TO READ A CARD AND THEN PRINT IT OUT                 */
                     /*                                                                 */
                     /*ENTRY:      MAIN. ONLY ENTRY POINT                               */
                     /*                                                                 */
                     /*   ...                                                           */
                     /*                                                                 */
                     /*******************************************************************/
                    MAIN:    PROCEDURE;

                             DECLARE /*VARIABLE DATA ITEMS FOR THIS PROCEDURE*/
                                   /*I/O BUFFER AREAS*/
                                   BUF CHAR(INSZ),    /*INPUT CARD BUFFER*/
                                   OUT CHAR(OUTSZ),   /*OUTPUT LINE BUFFER*/
                                   /*RETURN CODE VARIABLE*/
                                   CODE FIXED(31);    /*CODE SET BY RET*/
                             DECLARE /*ROUTINES CALLED*/
                                   RET ENTRY,         /*READS IN A CARD*/
                                   PRINT ENTRY;       /*PRINTS A LINE*/

                    OBTAIN:  /*OBTAIN AN INPUT CARD*/
                             CALL RET(BUF,CODE);      /*GET A CARD, AND SET CODE:
                                                           =0, NORMAL READ
                                                           =1, END OF FILE
                                                           =2, ERROR    */
                             /*CHECK CODE FOR VALIDITY*/
                             IF CODE=0 THEN          /*VALID INPUT*/
                               /*PRINT OUT THIS CARD AND KEEP GOING*/
                               DO;
                               OUT(1)=' ';            /*SET FOR SINGLE SPACING*/
                               OUT(2:OUTSZ)=BUF;      /*MOVE CARD TO OUTPUT LINE*/
                               CALL PRINT(OUT);       /*OUTPUT THE CARD*/
                               GO TO OBTAIN;          /*CONTINUE WITH THE NEXT CARD*/
                               END;
                             ELSE IF CODE=1 THEN      /*NO MORE INPUT*/
                               RETURN;                /*RETURN TO CALLING PROGRAM*/
                             ELSE                     /*ERROR SITUATION*/
                               /*NOTE THE ERROR ON THE PRINT FILE*/
                               DO;
                               OUT=' ';               /*BLANK THE OUTPUT LINE*/
                               OUT(LECOL:RECOL)='ERROR'; /*INDICATE AN ERROR OCCURRED*/
                               CALL PRINT(OUT);       /*OUTPUT THE ERROR INDICATION*/
                               RETURN;                /*RETURN TO CALLING PROGRAM*/
                               END;

                             END MAIN;                /*END OF THE PROCEDURE*/
NO ERROR OR WARNING CONDITION HAS BEEN DETECTED FOR THIS MACRO PASS.
```

Including Text From A Library

It is possible that an error could result from the generalizing of RECOL and LECOL. If OUTSZ is less than either RECOL or LECOL, or both, the word 'ERROR' would not be placed in the output line. To avoid this possibility, and to further generalize the sample program, source text from an external library (Figure 13) could be added during compilation.



```
OUT='b'; /*SET CARRIAGE CONTROL TO BLANK*/
OUT(LECOL:RECOL)='ERROR';
```

```
/****ERROR POSITION WITHIN NORMAL CARD POSITION****/
OUT='0'; /*SET CARRIAGE CONTROL TO 0*/
OUT(2:6)='ERROR';
```

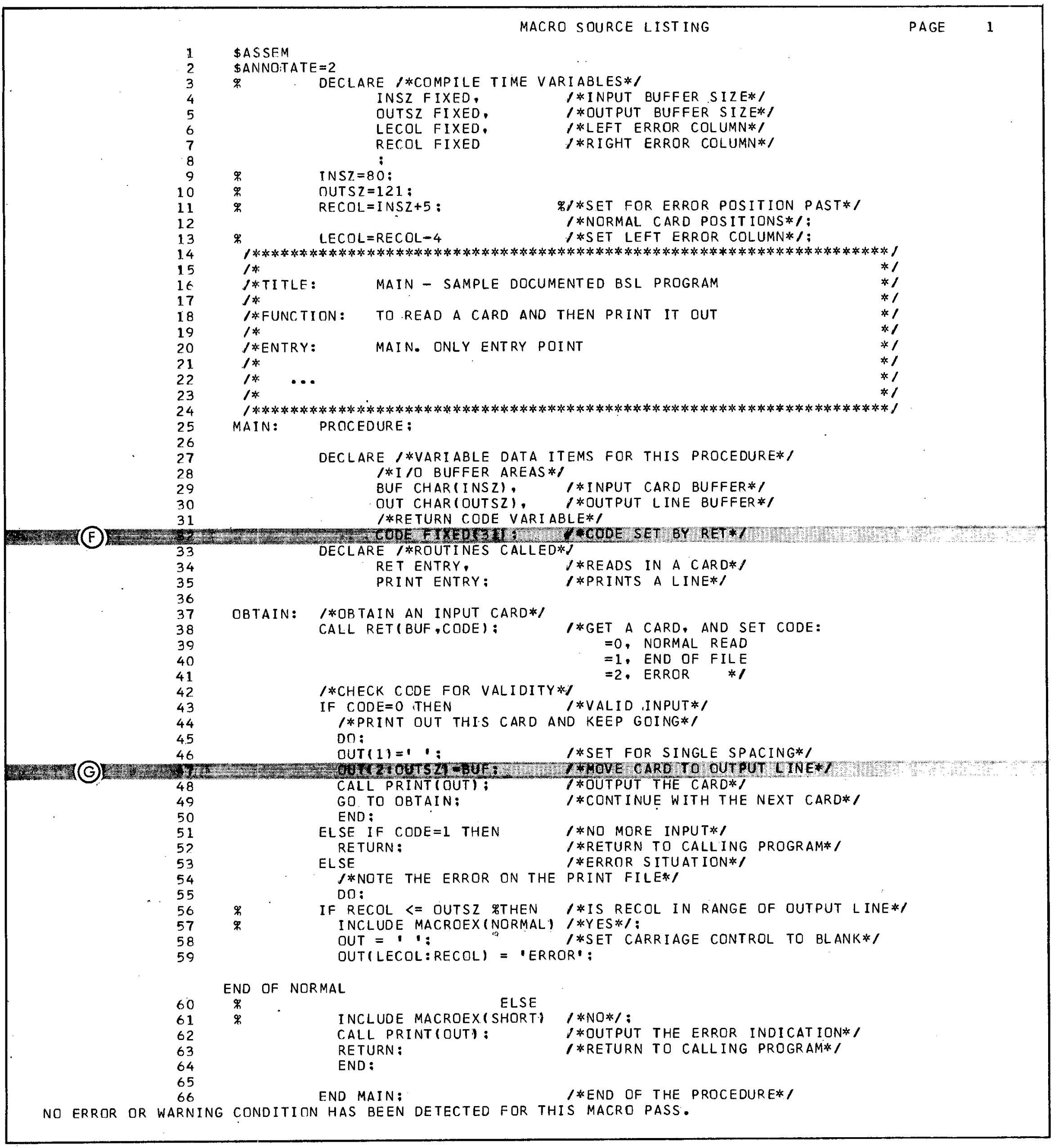
• Figure 13. Source Text From an External Library

Insert E will replace the two instructions of the sample program (<u>both</u> the OUT='b'; and the OUT(LECOL:RECOL)='ERROR'; statements in Example 2).

```
Insert E

%IF RECOL<=OUTSZ % THEN          /* IS RECOL IN RANGE OF OUTPUT LINE */
    %INCLUDE MACROEX(NORMAL)     /* YES */;
%ELSE
    %INCLUDE MACROEX(SHORT)      /* NO */;
```

One of the two paths will be taken at compile time, depending upon the values of RECOL and OUTSZ. One of two sets of data will be included in the program. Example 3 shows a macro source listing that adds source text from MACROEX(NORMAL).

- Example 3. BSL Macro Source Listing (Including Text)



```
                                                  MACRO SOURCE LISTING                           PAGE     1
                  1   $ASSEM
                  2   $ANNOTATE=2
                  3   %        DECLARE /*COMPILE TIME VARIABLES*/
                  4                  INSZ FIXED,         /*INPUT BUFFER SIZE*/
                  5                  OUTSZ FIXED,        /*OUTPUT BUFFER SIZE*/
                  6                  LECOL FIXED,        /*LEFT ERROR COLUMN*/
                  7                  RECOL FIXED         /*RIGHT ERROR COLUMN*/
                  8                  ;
                  9   %         INSZ=80;
                 10   %         OUTSZ=121;
                 11   %         RECOL=INSZ+5;           %/*SET FOR ERROR POSITION PAST*/
                 12                                      /*NORMAL CARD POSITIONS*/;
                 13   %         LECOL=RECOL-4            /*SET LEFT ERROR COLUMN*/;
                 14    /*******************************************************************/
                 15    /*                                                                 */
                 16    /*TITLE:      MAIN - SAMPLE DOCUMENTED BSL PROGRAM                 */
                 17    /*                                                                 */
                 18    /*FUNCTION:   TO READ A CARD AND THEN PRINT IT OUT                 */
                 19    /*                                                                 */
                 20    /*ENTRY:      MAIN. ONLY ENTRY POINT                               */
                 21    /*                                                                 */
                 22    /*   ...                                                           */
                 23    /*                                                                 */
                 24    /*******************************************************************/
                 25   MAIN:     PROCEDURE;
                 26
                 27             DECLARE /*VARIABLE DATA ITEMS FOR THIS PROCEDURE*/
                 28                   /*I/O BUFFER AREAS*/
                 29                   BUF CHAR(INSZ),     /*INPUT CARD BUFFER*/
                 30                   OUT CHAR(OUTSZ),    /*OUTPUT LINE BUFFER*/
                 31                   /*RETURN CODE VARIABLE*/
                 32                   CODE FIXED(31);     /*CODE SET BY RET*/
                 33             DECLARE /*ROUTINES CALLED*/
                 34                   RET ENTRY,          /*READS IN A CARD*/
                 35                   PRINT ENTRY;        /*PRINTS A LINE*/
                 36
                 37   OBTAIN:  /*OBTAIN AN INPUT CARD*/
                 38             CALL RET(BUF,CODE);       /*GET A CARD, AND SET CODE:
                 39                                           =0, NORMAL READ
                 40                                           =1, END OF FILE
                 41                                           =2, ERROR    */
                 42             /*CHECK CODE FOR VALIDITY*/
                 43             IF CODE=0 THEN            /*VALID INPUT*/
                 44               /*PRINT OUT THIS CARD AND KEEP GOING*/
                 45               DO;
                 46               OUT(1)=' ';             /*SET FOR SINGLE SPACING*/
                 47               OUT(2:OUTSZ)=BUF;       /*MOVE CARD TO OUTPUT LINE*/
                 48               CALL PRINT(OUT);        /*OUTPUT THE CARD*/
                 49               GO TO OBTAIN;           /*CONTINUE WITH THE NEXT CARD*/
                 50               END;
                 51             ELSE IF CODE=1 THEN       /*NO MORE INPUT*/
                 52               RETURN;                 /*RETURN TO CALLING PROGRAM*/
                 53             ELSE                      /*ERROR SITUATION*/
                 54               /*NOTE THE ERROR ON THE PRINT FILE*/
                 55               DO;
                 56   %         IF RECOL <= OUTSZ %THEN   /*IS RECOL IN RANGE OF OUTPUT LINE*/
                 57   %           INCLUDE MACROEX(NORMAL) /*YES*/;
                 58               OUT = ' ';              /*SET CARRIAGE CONTROL TO BLANK*/
                 59               OUT(LECOL:RECOL) = 'ERROR';

                   END OF NORMAL
                 60   %                            ELSE
                 61   %           INCLUDE MACROEX(SHORT)  /*NO*/;
                 62               CALL PRINT(OUT);        /*OUTPUT THE ERROR INDICATION*/
                 63               RETURN;                 /*RETURN TO CALLING PROGRAM*/
                 64               END;
                 65
                 66             END MAIN;                 /*END OF THE PROCEDURE*/
NO ERROR OR WARNING CONDITION HAS BEEN DETECTED FOR THIS MACRO PASS.
```

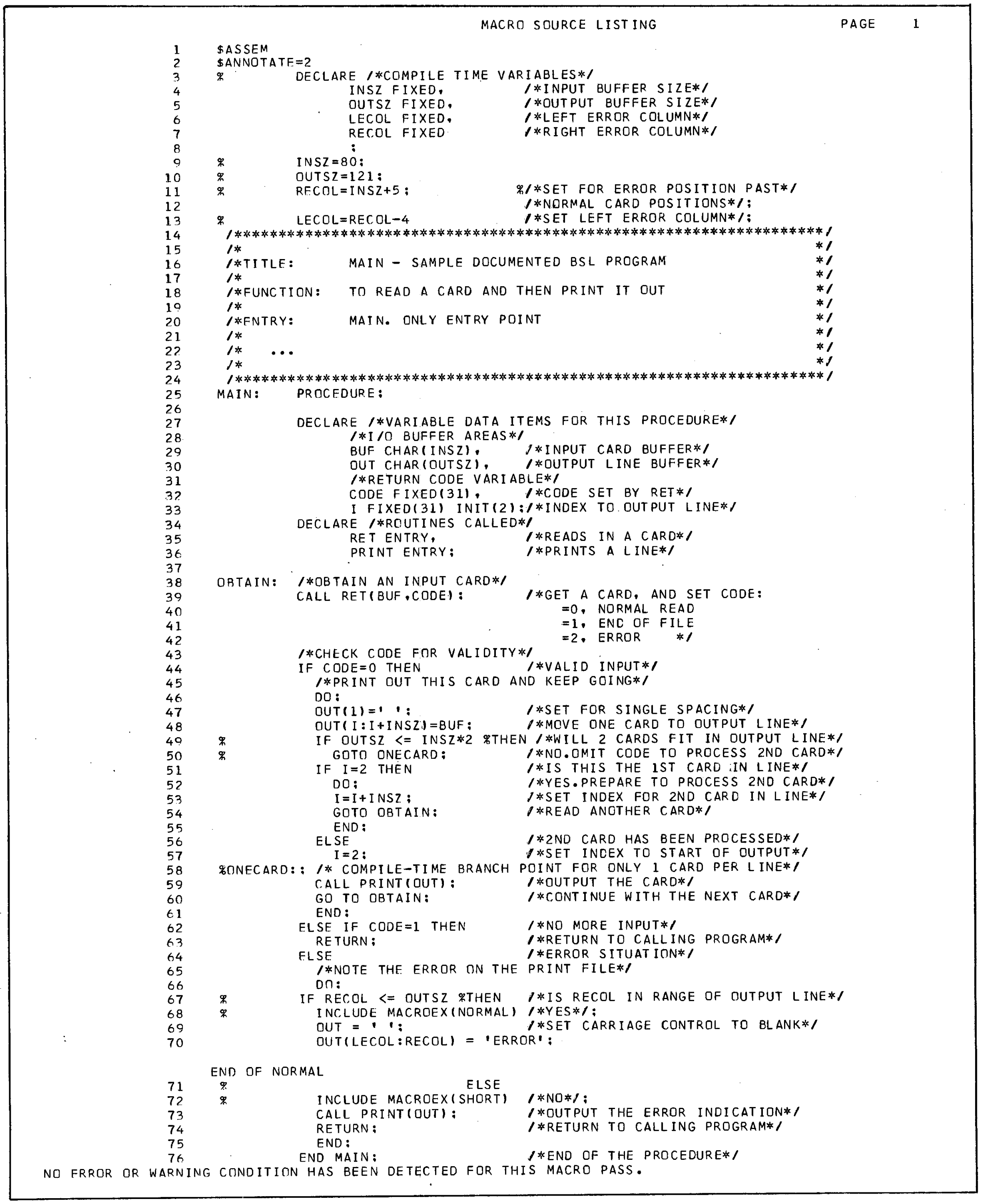
Altering the Source Code

To determine the number of input records that would fit on an output line for any given run, and to compile only the code to print that number of records, two changes to the program, as shown in Example 3, are needed. (Note the places for changes are indicated in Example 3.)

```
 Insert F

 CODE FIXED (31),          /* CODE SET BY RET */
 I FIXED (31) INIT (2);    /* INDEX TO OUTPUT LINE */

 Insert G

 OUT(I:I+INSZ)=BUF;        /* MOVE ONE CARD TO OUTPUT LINE */

%IF OUTSZ <=INSZ*2 % THEN /* WILL 2 CARDS FIT IN OUTPUT LINE */
%    GOTO ONECARD;         /* NO. OMIT CODE TO PROCESS 2ND CARD */
 IF I=2 THEN               /* IS THIS THE FIRST CARD IN LINE */
     DO;                   /* YES. PREPARE TO PROCESS 2ND CARD */
     I=I+INSZ;             /* SET INDEX FOR 2ND CARD IN LINE */
     GOTO OBTAIN;          /* READ ANOTHER CARD */
     END;
 ELSE                      /* 2ND CARD HAS BEEN PROCESSED */
     I=2;                  /* SET INDEX TO START OF OUTPUT */
%ONECARD: ; /* COMPILE TIME BRANCH POINT FOR ONLY 1 CARD PER LINE */
```

Insert F initializes the variable I. Insert G moves data from one input card to the output buffer. The compile-time statements determine if data from two input cards can fit into the output buffer. One of two branches is taken as a result of the determination. Example 4 shows the macro source listing at compile-time, and indicates an altered input line.

- **Example 4. BSL Macro Source Listing (Altering Source Code)**



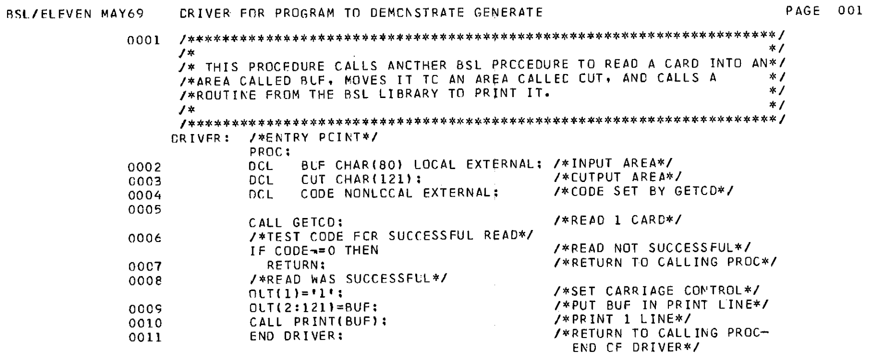
```
                                             MACRO SOURCE LISTING                PAGE    1
                  $ASSEM
                  $ANNOTATE=2
                  %       DECLARE /*COMPILE TIME VARIABLES*/
                                INSZ FIXED,       /*INPUT BUFFER SIZE*/
                                OUTSZ FIXED,      /*OUTPUT BUFFER SIZE*/
                                LECOL FIXED,      /*LEFT ERROR COLUMN*/
                                RECOL FIXED       /*RIGHT ERROR COLUMN*/
                                ;
                  %       INSZ=80;
                  %       OUTSZ=121;
                  %       RECOL=INSZ+5 ;        %/*SET FOR ERROR POSITION PAST*/
                                                  /*NORMAL CARD POSITIONS*/;
                  %       LECOL=RECOL-4           /*SET LEFT ERROR COLUMN*/;
                   /*************************************************************/
                   /*                                                           */
                   /*TITLE:     MAIN - SAMPLE DOCUMENTED BSL PROGRAM            */
                   /*                                                           */
                   /*FUNCTION:  TO READ A CARD AND THEN PRINT IT OUT            */
                   /*                                                           */
                   /*ENTRY:     MAIN. ONLY ENTRY POINT                          */
                   /*                                                           */
                   /*  ...                                                      */
                   /*                                                           */
                   /*************************************************************/
                  MAIN:   PROCEDURE;

                          DECLARE /*VARIABLE DATA ITEMS FOR THIS PROCEDURE*/
                                /*I/O BUFFER AREAS*/
                                BUF CHAR(INSZ),   /*INPUT CARD BUFFER*/
                                OUT CHAR(OUTSZ),  /*OUTPUT LINE BUFFER*/
                                /*RETURN CODE VARIABLE*/
                                CODE FIXED(31),   /*CODE SET BY RET*/
                                I FIXED(31) INIT(2);/*INDEX TO OUTPUT LINE*/
                          DECLARE /*ROUTINES CALLED*/
                                RET ENTRY,        /*READS IN A CARD*/
                                PRINT ENTRY;      /*PRINTS A LINE*/

                  OBTAIN: /*OBTAIN AN INPUT CARD*/
                          CALL RET(BUF,CODE);     /*GET A CARD, AND SET CODE:
                                                      =0, NORMAL READ
                                                      =1, END OF FILE
                                                      =2, ERROR    */
                          /*CHECK CODE FOR VALIDITY*/
                          IF CODE=0 THEN          /*VALID INPUT*/
                            /*PRINT OUT THIS CARD AND KEEP GOING*/
                            DO;
                            OUT(1)=' ';           /*SET FOR SINGLE SPACING*/
                            OUT(I:I+INSZ)=BUF;    /*MOVE ONE CARD TO OUTPUT LINE*/
                  %         IF OUTSZ <= INSZ*2 %THEN /*WILL 2 CARDS FIT IN OUTPUT LINE*/
                  %           GOTO ONECARD;       /*NO.OMIT CODE TO PROCESS 2ND CARD*/
                            IF I=2 THEN           /*IS THIS THE 1ST CARD IN LINE*/
                              DO;                 /*YES.PREPARE TO PROCESS 2ND CARD*/
                              I=I+INSZ ;          /*SET INDEX FOR 2ND CARD IN LINE*/
                              GOTO OBTAIN;        /*READ ANOTHER CARD*/
                              END;
                            ELSE                  /*2ND CARD HAS BEEN PROCESSED*/
                              I=2;                /*SET INDEX TO START OF OUTPUT*/
                  %ONECARD:; /* COMPILE-TIME BRANCH POINT FOR ONLY 1 CARD PER LINE*/
                            CALL PRINT(OUT);      /*OUTPUT THE CARD*/
                            GO TO OBTAIN;         /*CONTINUE WITH THE NEXT CARD*/
                            END;
                          ELSE IF CODE=1 THEN     /*NO MORE INPUT*/
                            RETURN;               /*RETURN TO CALLING PROGRAM*/
                          ELSE                    /*ERROR SITUATION*/
                            /*NOTE THE ERROR ON THE PRINT FILE*/
                            DO;
                  %       IF RECOL <= OUTSZ %THEN   /*IS RECOL IN RANGE OF OUTPUT LINE*/
                  %         INCLUDE MACROEX(NORMAL) /*YES*/;
                            OUT = ' ';            /*SET CARRIAGE CONTROL TO BLANK*/
                            OUT(LECOL:RECOL) = 'ERROR';

                 END OF NORMAL
                  %                         ELSE
                  %         INCLUDE MACROEX(SHORT)  /*NO*/;
                            CALL PRINT(OUT);      /*OUTPUT THE ERROR INDICATION*/
                            RETURN;               /*RETURN TO CALLING PROGRAM*/
                            END;
                          END MAIN;               /*END OF THE PROCEDURE*/
NO FRROR OR WARNING CONDITION HAS BEEN DETECTED FOR THIS MACRO PASS.
```

Appendix A: Example of Program Using BSL GENERATE Statements

| Example 5 shows a BSL program which uses both forms of the GENERATE statement. The program reads a card into an area called BUF.

Of special interest to OS users is the reference to the OFLAGS field of the DCB, which appears in a block GENERATE. You should also note that CARDIN, the DCB, is declared as GENERATED.

• Example 5. Use of the GENERATE Statements (Part 1 of 10)

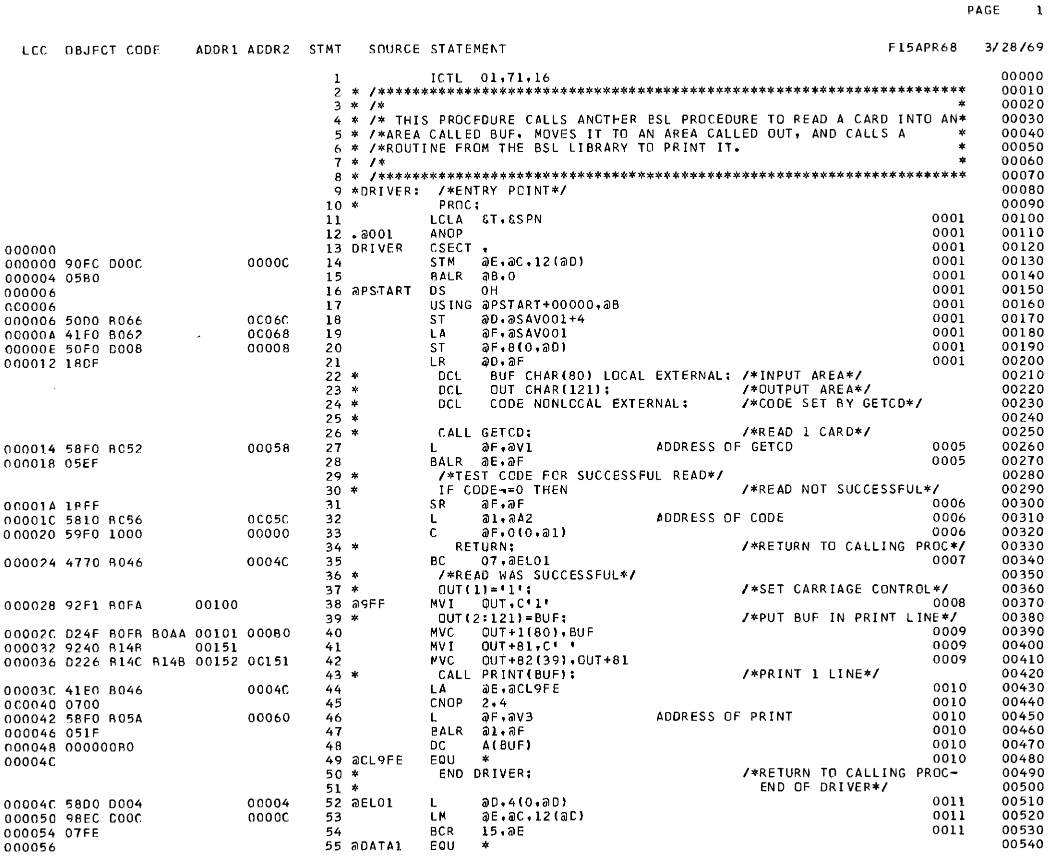


```
BSL/ELEVEN MAY69    CRIVER FOR PROGRAM TO DEMCNSTRATE GENERATE                    PAGE  001

           0001  /*******************************************************************/
                 /*                                                                  */
                 /* THIS PROCEDURE CALLS ANCTHER BSL PRCCEDURE TO READ A CARD INTO AN*/
                 /*AREA CALLED BLF, MOVES IT TC AN AREA CALLEC CUT, AND CALLS A      */
                 /*ROUTINE FROM THE BSL LIBRARY TO PRINT IT.                         */
                 /*                                                                  */
                 /*******************************************************************/
                CRIVFR:  /*ENTRY PCINT*/
                         PROC;
           0002          DCL   BLF CHAR(80) LOCAL EXTERNAL; /*INPUT AREA*/
           0003          DCL   CUT CHAR(121);               /*CUTPUT AREA*/
           0004          DCL   CODE NONLCCAL EXTERNAL;      /*CODE SET BY GETCD*/
           0005
                         CALL GETCD;                        /*READ 1 CARD*/
           0006          /*TEST CODE FCR SUCCESSFUL READ*/
                         IF CODE¬=0 THEN                    /*READ NOT SUCCESSFUL*/
           0007            RETURN;                          /*RETURN TO CALLING PROC*/
           0008          /*READ WAS SUCCESSFUL*/
                         OLT(1)='1';                        /*SET CARRIAGE CONTROL*/
           0009          OLT(2:121)=BUF;                    /*PUT BUF IN PRINT LINE*/
           0010          CALL PRINT(BUF);                   /*PRINT 1 LINE*/
           0011          END DRIVER;                        /*RETURN TO CALLING PROC-
                                                              END CF DRIVER*/
```

• Example 5. Use of the GENERATE Statements (Part 2 of 10)

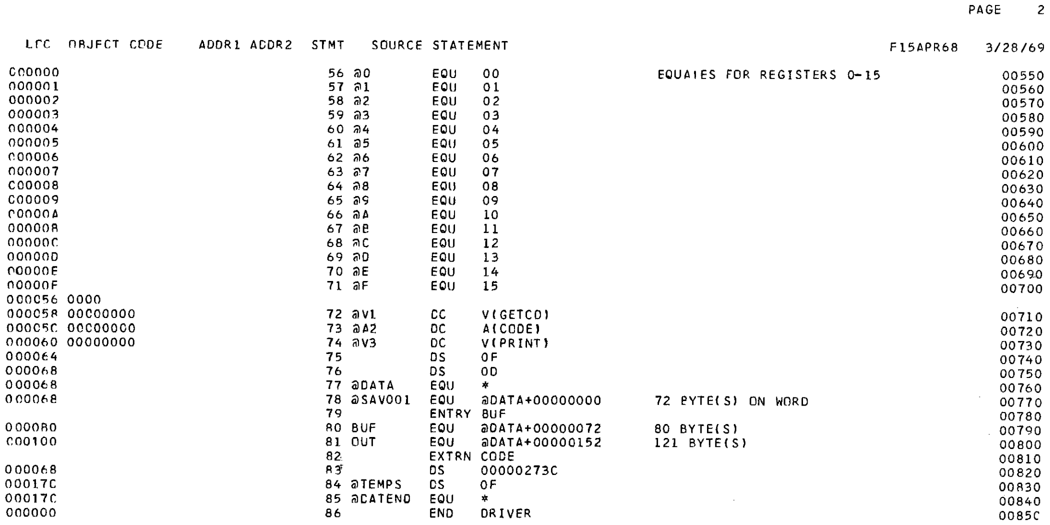
```
BSL/ELEVEN MAY69    DRIVER FOR PROGRAM TO DEMONSTRATE GENERATE                       PAGE  002
     DCL'D IN     NAME             ATTRIBUTE AND CROSS REFERENCE TABLE

        2         BUF             STATIC, LOCAL, CHARACTER(80), EXTERNAL, BOUNDARY(BYTE,1)
                                  9, 10

        4         CODE            STATIC, NONLOCAL, FIXED(31), EXTERNAL, BOUNDARY(WORD,1)
                                  6

        1         DRIVER          STATIC, LOCAL, ENTRY, EXTERNAL
                                  1, 11

        5         GETCD           STATIC, NONLOCAL, ENTRY, EXTERNAL
                                  5, 11

        3         OUT             STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1)
                                  8, 9

       10         PRINT           STATIC, NONLOCAL, ENTRY, EXTERNAL
                                  10, 11

              *** PROC. DRIVER   HAD  NO ERRORS
```

- Example 5. Use of the GENERATE Statements (Part 3 of 10)



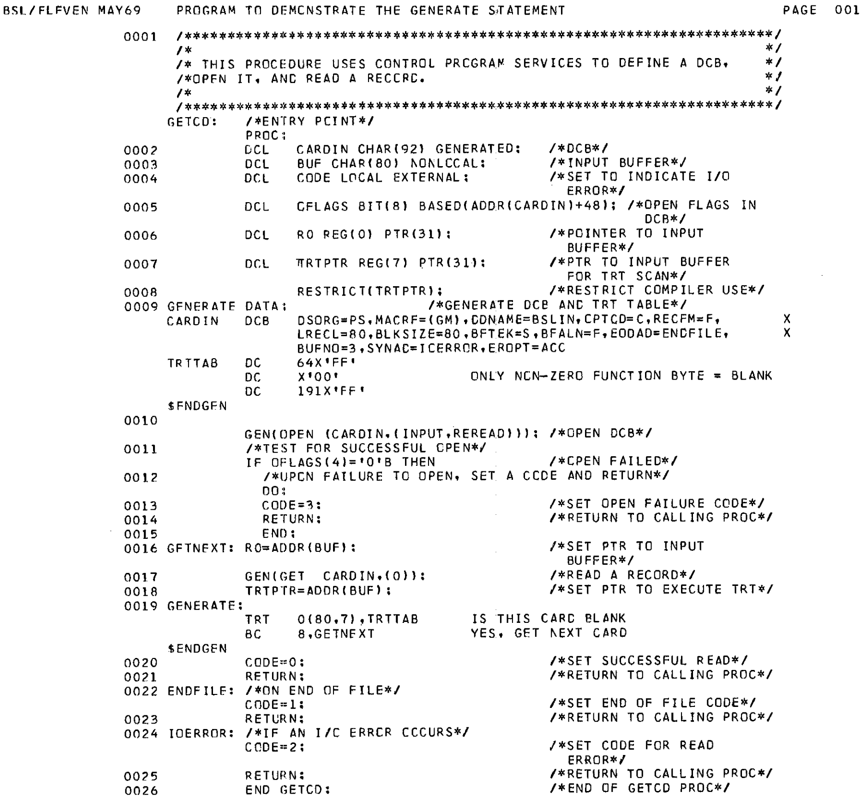
```
                                                                                                                PAGE    1

  LCC  OBJFCT CODF    ADDR1 ACDR2  STMT   SOURCE STATEMENT                                            F15APR68   3/28/69
                                      1          ICTL  01,71,16                                                    00000
                                      2 * /********************************************************************    00010
                                      3 * /*                                                                  *    00020
                                      4 * /* THIS PROCFDURE CALLS ANCTHER BSL PROCEDURE TO READ A CARD INTO AN*    00030
                                      5 * /*AREA CALLED BUF, MOVES IT TO AN AREA CALLED OUT, AND CALLS A      *    00040
                                      6 * /*ROUTINE FROM THE BSL LIBRARY TO PRINT IT.                         *    00050
                                      7 * /*                                                                  *    00060
                                      8 * /********************************************************************    00070
                                      9 *DRIVER:  /*ENTRY PCINT*/                                                  00080
                                     10 *         PROC;                                                            00090
                                     11          LCLA  &T,&SPN                                         0001    00100
                                     12 .@001    ANOP                                                  0001    00110
000000                               13 DRIVER   CSECT ,                                               0001    00120
000000 90FC D00C            0000C    14          STM   @E,@C,12(@D)                                    0001    00130
000004 05B0                          15          BALR  @B,0                                            0001    00140
000006                               16 @PSTART  DS    0H                                              0001    00150
0C0006                               17          USING @PSTART+00000,@B                                0001    00160
000006 50D0 R066            0C06C    18          ST    @D,@SAV001+4                                    0001    00170
00000A 41F0 B062            0C068    19          LA    @F,@SAV001                                      0001    00180
00000E 50F0 D008            00008    20          ST    @F,8(0,@D)                                      0001    00190
000012 18DF                          21          LR    @D,@F                                           0001    00200
                                     22 *         DCL   BUF CHAR(80) LOCAL EXTERNAL; /*INPUT AREA*/                00210
                                     23 *         DCL   OUT CHAR(121);               /*OUTPUT AREA*/               00220
                                     24 *         DCL   CODE NONLCCAL EXTERNAL;      /*CODE SET BY GETCD*/         00230
                                     25 *                                                                          00240
                                     26 *         CALL GETCD;                        /*READ 1 CARD*/               00250
000014 58F0 BC52            00058    27          L     @F,@V1               ADDRESS OF GETCD               0005    00260
000018 05EF                          28          BALR  @E,@F                                           0005    00270
                                     29 *         /*TEST CODE FCR SUCCESSFUL READ*/                                00280
                                     30 *         IF CODE¬=0 THEN                    /*READ NOT SUCCESSFUL*/       00290
00001A 1BFF                          31          SR    @F,@F                                           0006    00300
00001C 5810 BC56            0C05C    32          L     @1,@A2               ADDRESS OF CODE                0006    00310
000020 59F0 1000            00000    33          C     @F,0(0,@1)                                      0006    00320
                                     34 *           RETURN;                  /*RETURN TO CALLING PROC*/    00330
000024 4770 B046            0004C    35          BC    07,@EL01                                        0007    00340
                                     36 *         /*READ WAS SUCCESSFUL*/                                          00350
                                     37 *         OUT(1)='1';                        /*SET CARRIAGE CONTROL*/      00360
000028 92F1 R0FA      00100          38 @9FF     MVI   OUT,C'1'                                        0008    00370
                                     39 *         OUT(2:121)=BUF:                    /*PUT BUF IN PRINT LINE*/     00380
00002C D24F B0FB B0AA 00101 000B0    40          MVC   OUT+1(80),BUF                                   0009    00390
000032 9240 B14B      00151          41          MVI   OUT+81,C' '                                     0009    00400
000036 D226 B14C B14B 00152 0C151    42          MVC   OUT+82(39),OUT+81                               0009    00410
                                     43 *         CALL PRINT(BUF):                   /*PRINT 1 LINE*/              00420
00003C 41E0 B046            0004C    44          LA    @E,@CL9FE                                       0010    00430
0C0040 0700                          45          CNOP  2,4                                             0010    00440
000042 58F0 B05A            00060    46          L     @F,@V3               ADDRESS OF PRINT               0010    00450
000046 051F                          47          BALR  @1,@F                                           0010    00460
000048 000000B0                      48          DC    A(BUF)                                          0010    00470
00004C                               49 @CL9FE   EQU   *                                               0010    00480
                                     50 *         END DRIVER;                        /*RETURN TO CALLING PROC-     00490
                                     51 *                                              END OF DRIVER*/             00500
00004C 58D0 D004            00004    52 @EL01    L     @D,4(0,@D)                                      0011    00510
000050 98EC D00C            0000C    53          LM    @E,@C,12(@C)                                    0011    00520
000054 07FE                          54          BCR   15,@E                                           0011    00530
000056                               55 @DATA1   EQU   *                                                       00540
```

• Example 5. Use of the GENERATE Statements (Part 4 of 10)



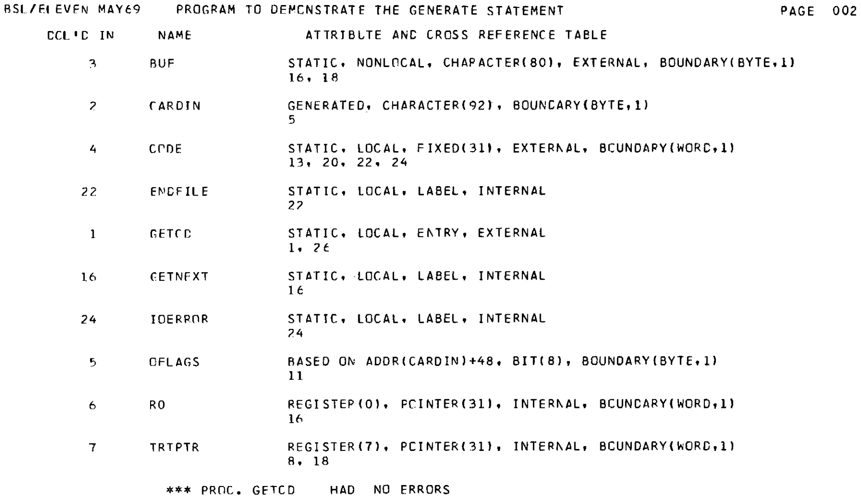
```
                                                                                                   PAGE    2

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT                                F15APR68    3/28/69

000000                                56 @0        EQU    00               EQUATES FOR REGISTERS 0-15        00550
000001                                57 @1        EQU    01                                                 00560
000002                                58 @2        EQU    02                                                 00570
000003                                59 @3        EQU    03                                                 00580
000004                                60 @4        EQU    04                                                 00590
000005                                61 @5        EQU    05                                                 00600
000006                                62 @6        EQU    06                                                 00610
000007                                63 @7        EQU    07                                                 00620
000008                                64 @8        EQU    08                                                 00630
000009                                65 @9        EQU    09                                                 00640
00000A                                66 @A        EQU    10                                                 00650
00000B                                67 @B        EQU    11                                                 00660
00000C                                68 @C        EQU    12                                                 00670
00000D                                69 @D        EQU    13                                                 00680
00000E                                70 @E        EQU    14                                                 00690
00000F                                71 @F        EQU    15                                                 00700
000056 0000
000058 00000000                       72 @V1       DC     V(GETCD)                                           00710
00005C 00000000                       73 @A2       DC     A(CODE)                                            00720
000060 00000000                       74 @V3       DC     V(PRINT)                                           00730
000064                                75           DS     0F                                                 00740
000068                                76           DS     0D                                                 00750
000068                                77 @DATA     EQU    *                                                  00760
000068                                78 @SAV001   EQU    @DATA+00000000    72 BYTE(S) ON WORD               00770
                                      79           ENTRY  BUF                                                00780
000080                                80 BUF       EQU    @DATA+00000072    80 BYTE(S)                       00790
000100                                81 OUT       EQU    @DATA+00000152    121 BYTE(S)                      00800
                                      82           EXTRN  CODE                                               00810
000068                                83           DS     00000273C                                          00820
00017C                                84 @TEMPS    DS     0F                                                 00830
00017C                                85 @DATEND   EQU    *                                                  00840
000000                                86           END    DRIVER                                             00850
```

• Example 5. Use of the GENERATE Statements (Part 5 of 10)



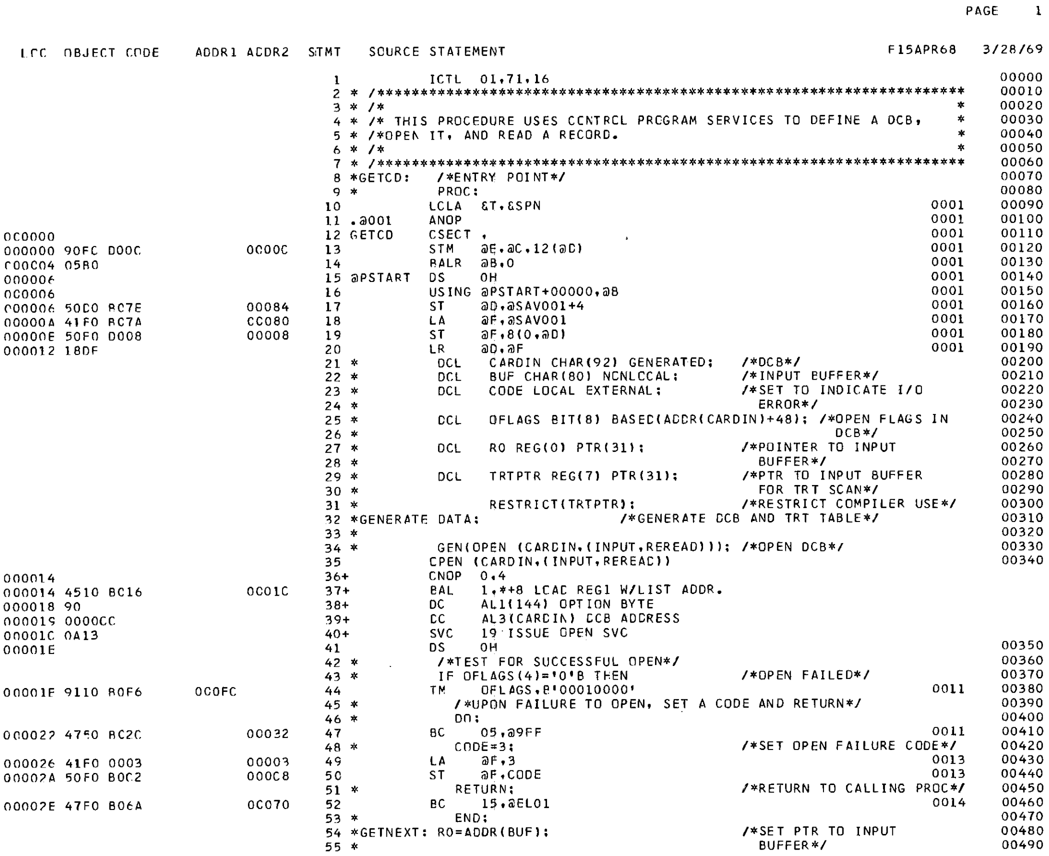
```
BSL/FLFVEN MAY69    PROGRAM TO DEMCNSTRATE THE GENERATE STATEMENT                    PAGE  001

              0001 /*********************************************************************/
                   /*                                                                   */
                   /* THIS PROCEDURE USES CONTROL PRCGRAM SERVICES TO DEFINE A DCB,     */
                   /*OPFN IT, ANC READ A RECCRC.                                        */
                   /*                                                                   */
                   /*********************************************************************/
                  GETCD:   /*ENTRY PCINT*/
                           PROC;
              0002         DCL   CARDIN CHAR(92) GENERATED:   /*DCB*/
              0003         DCL   BUF CHAR(80) NONLCCAL:       /*INPUT BUFFER*/
              0004         DCL   CODE LOCAL EXTERNAL;         /*SET TO INDICATE I/O
                                                                ERROR*/
              0005         DCL   CFLAGS BIT(8) BASED(ADDR(CARDIN)+48); /*OPEN FLAGS IN
                                                                           DCB*/
              0006         DCL   RO REG(0) PTR(31);           /*POINTER TO INPUT
                                                                BUFFER*/
              0007         DCL   TRTPTR REG(7) PTR(31):       /*PTR TO INPUT BUFFER
                                                                FOR TRT SCAN*/
              0008               RESTRICT(TRTPTR);            /*RESTRICT COMPILER USE*/
              0009 GFNERATE DATA;              /*GENERATE DCB AND TRT TABLE*/
                  CARDIN   DCB   DSORG=PS,MACRF=(GM),DDNAME=BSLIN,CPTCD=C,RECFM=F,         X
                                 LRECL=80,BLKSIZE=80,BFTEK=S,BFALN=F,EODAD=ENDFILE,        X
                                 BUFNO=3,SYNAC=ICERROR,EROPT=ACC
                  TRTTAB   DC    64X'FF'
                           DC    X'00'                ONLY NCN-ZERO FUNCTION BYTE = BLANK
                           DC    191X'FF'
                  $FNDGFN
              0010
                           GEN(OPEN (CARDIN,(INPUT,REREAD))); /*OPEN DCB*/
              0011         /*TEST FOR SUCCESSFUL CPEN*/
                           IF OFLAGS(4)='0'B THEN             /*CPEN FAILED*/
              0012           /*UPCN FAILURE TO OPEN, SET A CCDE AND RETURN*/
                             DO;
              0013           CODE=3:                          /*SET OPEN FAILURE CODE*/
              0014           RETURN:                          /*RETURN TO CALLING PROC*/
              0015           END;
              0016 GFTNEXT: RO=ADDR(BUF):                     /*SET PTR TO INPUT
                                                                BUFFER*/
              0017         GEN(GET  CARDIN,(0)):              /*READ A RECORD*/
              0018         TRTPTR=ADDR(BUF);                  /*SET PTR TO EXECUTE TRT*/
              0019 GENERATE:
                           TRT   0(80,7),TRTTAB       IS THIS CARC BLANK
                           BC    8,GETNFXT            YES, GET NEXT CARD
                  $ENDGEN
              0020         CODE=0:                            /*SET SUCCESSFUL READ*/
              0021         RETURN:                            /*RETURN TO CALLING PROC*/
              0022 ENDFILE: /*ON END OF FILE*/
                           CODE=1:                            /*SET END OF FILE CODE*/
              0023         RETURN:                            /*RETURN TO CALLING PROC*/
              0024 IOERROR: /*IF AN I/C ERRCR CCCURS*/
                           CODE=2;                            /*SET CODE FOR READ
                                                                ERROR*/
              0025         RETURN:                            /*RETURN TO CALLING PROC*/
              0026         END GETCD:                         /*END OF GETCD PROC*/
```

• Example 5. Use of the GENERATE Statements (Part 6 of 10)



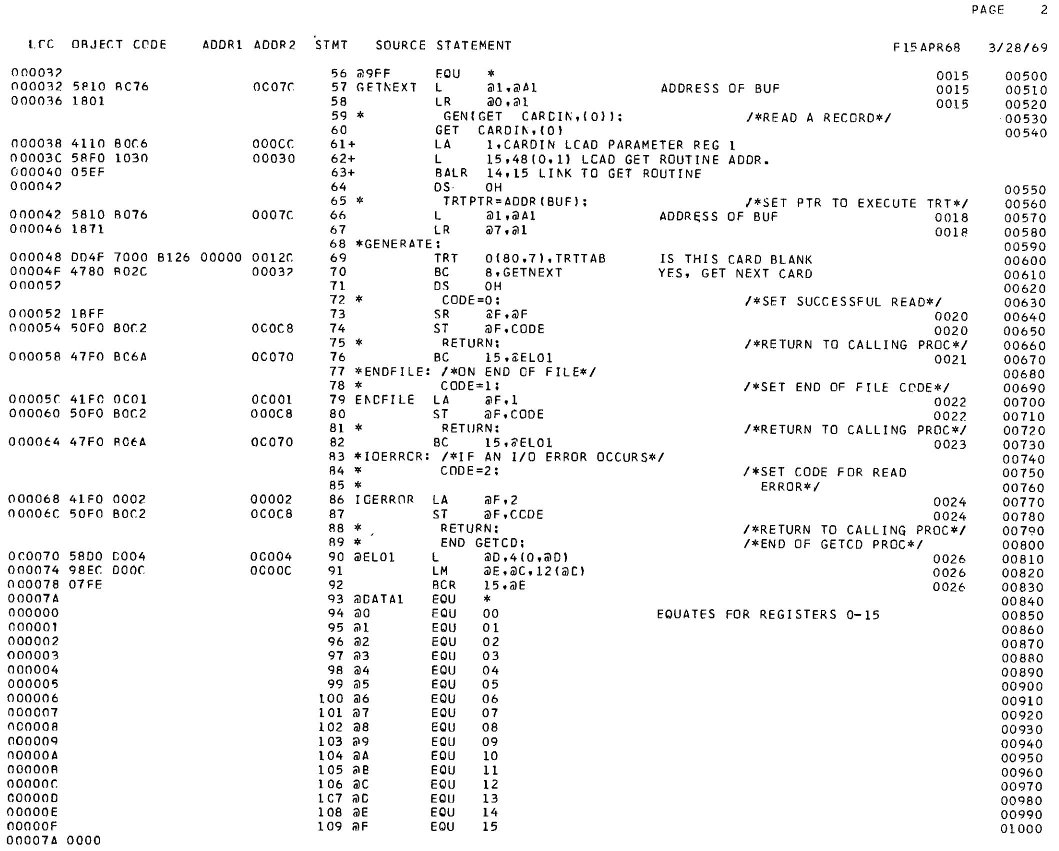
```
BSL/EI EVEN MAY69    PROGRAM TO DEMCNSTRATE THE GENERATE STATEMENT                 PAGE  002
     CCL'C IN      NAME               ATTRIBLTE ANC CROSS REFERENCE TABLE

           3       BUF              STATIC, NONLOCAL, CHAPACTER(80), EXTERNAL, BOUNDARY(BYTE,1)
                                    16, 18

           2       CARDIN           GENERATED, CHARACTER(92), BOUNCARY(BYTE,1)
                                    5

           4       CODE             STATIC, LOCAL, FIXED(31), EXTERNAL, BCUNDARY(WORC,1)
                                    13, 20, 22, 24

          22       ENDFILE          STATIC, LOCAL, LABEL, INTERNAL
                                    22

           1       GETCD            STATIC, LOCAL, ENTRY, EXTERNAL
                                    1, 26

          16       GETNEXT          STATIC, LOCAL, LABEL, INTERNAL
                                    16

          24       IOERROR          STATIC, LOCAL, LABEL, INTERNAL
                                    24

           5       OFLAGS           BASED ON ADDR(CARDIN)+48, BIT(8), BOUNDARY(BYTE,1)
                                    11

           6       RO               REGISTEP(0), PCINTER(31), INTERNAL, BCUNDARY(WORD,1)
                                    16

           7       TRTPTR           REGISTER(7), PCINTER(31), INTERNAL, BCUNDARY(WORD,1)
                                    8, 18

                 *** PROC. GETCD     HAD  NO ERRORS
```

• Example 5. Use of the GENERATE Statements (Part 7 of 10)



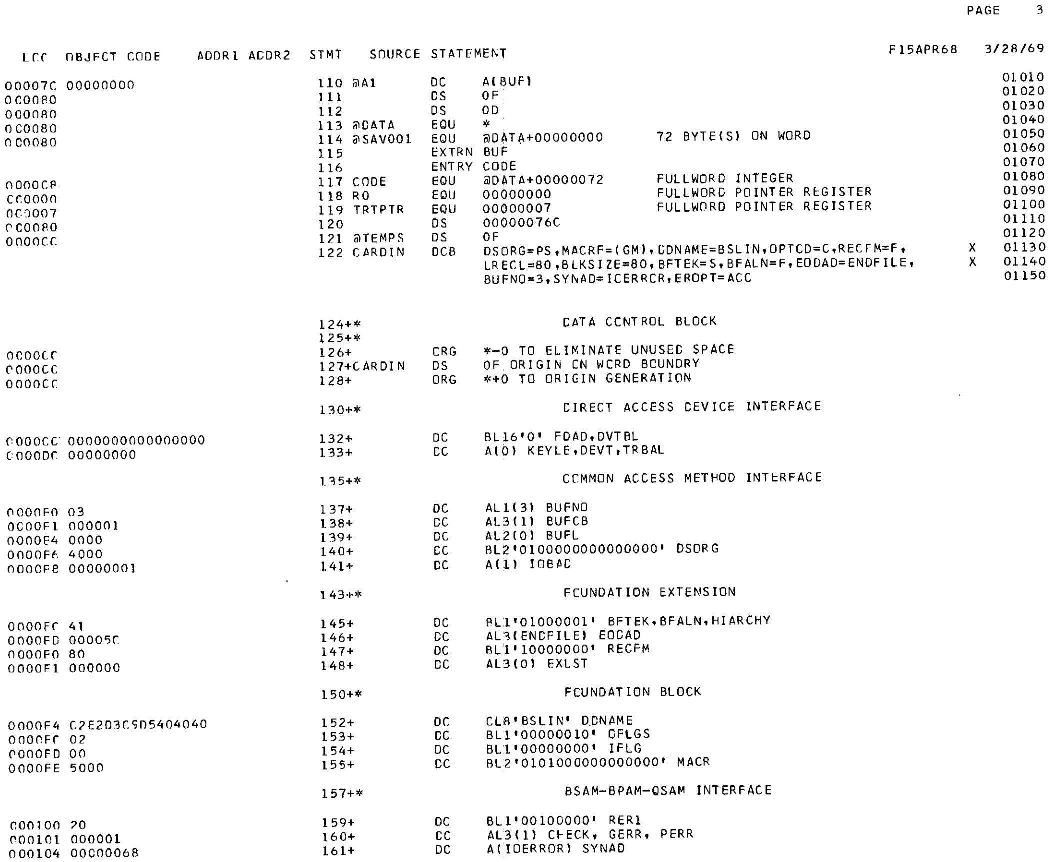
```
                                                                                                             PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2 STMT   SOURCE STATEMENT                                           F15APR68   3/28/69

                                     1          ICTL  01,71,16                                                   00000
                                     2 * /*******************************************************************    00010
                                     3 * /*                                                                 *    00020
                                     4 * /* THIS PROCEDURE USES CCNTRCL PRCGRAM SERVICES TO DEFINE A DCB,   *    00030
                                     5 * /*OPEN IT, AND READ A RECORD.                                      *    00040
                                     6 * /*                                                                 *    00050
                                     7 * /*******************************************************************    00060
                                     8 *GETCD:   /*ENTRY POINT*/                                                 00070
                                     9 *         PROC:                                                           00080
                                    10          LCLA  &T,&SPN                                        0001    00090
                                    11 .@001    ANOP                                                 0001    00100
0C0000                              12 GETCD    CSECT ,                                              0001    00110
000000 90FC D00C           0C00C    13          STM   @E,@C,12(@D)                                   0001    00120
C00C04 05B0                         14          BALR  @B,0                                           0001    00130
000006                              15 @PSTART  DS    0H                                             0001    00140
0C0006                              16          USING @PSTART+00000,@B                               0001    00150
C00006 50C0 BC7E           00084    17          ST    @D,@SAV001+4                                   0001    00160
00000A 41F0 BC7A           CC080    18          LA    @F,@SAV001                                     0001    00170
00000E 50F0 D008           00008    19          ST    @F,8(0,@D)                                     0001    00180
000012 18DF                         20          LR    @D,@F                                          0001    00190
                                    21 *         DCL   CARDIN CHAR(92) GENERATED;  /*DCB*/                        00200
                                    22 *         DCL   BUF CHAR(80) NCNLCCAL;      /*INPUT BUFFER*/               00210
                                    23 *         DCL   CODE LOCAL EXTERNAL;        /*SET TO INDICATE I/O          00220
                                    24 *                                             ERROR*/                     00230
                                    25 *         DCL   OFLAGS BIT(8) BASEC(ACDR(CARDIN)+48); /*OPEN FLAGS IN      00240
                                    26 *                                                      DCB*/               00250
                                    27 *         DCL   R0 REG(0) PTR(31);          /*POINTER TO INPUT             00260
                                    28 *                                             BUFFER*/                    00270
                                    29 *         DCL   TRTPTR REG(7) PTR(31);      /*PTR TO INPUT BUFFER          00280
                                    30 *                                             FOR TRT SCAN*/              00290
                                    31 *               RESTRICT(TRTPTR);           /*RESTRICT COMPILER USE*/      00300
                                    32 *GENERATE DATA;                /*GENERATE DCB AND TRT TABLE*/              00310
                                    33 *                                                                          00320
                                    34 *         GEN(OPEN (CARDIN,(INPUT,REREAD))); /*OPEN DCB*/                  00330
                                    35          CPEN (CARDIN,(INPUT,REREAC))                                     00340
000014                              36+         CNOP  0,4
000014 4510 BC16           0C01C    37+         BAL   1,*+8 LCAC REG1 W/LIST ADDR.
000018 90                           38+         DC    AL1(144) OPTION BYTE
000019 0000CC                       39+         CC    AL3(CARDIN) CCB ADDRESS
00001C 0A13                         40+         SVC   19 ISSUE OPEN SVC
00001E                              41          DS    0H                                                         00350
                                    42 *         /*TEST FOR SUCCESSFUL OPEN*/                                     00360
                                    43 *         IF OFLAGS(4)='0'B THEN            /*OPEN FAILED*/                00370
00001E 9110 B0F6     0C0FC          44          TM    OFLAGS,B'00010000'                             0011    00380
                                    45 *           /*UPON FAILURE TO OPEN, SET A CODE AND RETURN*/                00390
                                    46 *           DO;                                                           00400
000022 4750 BC2C           00032    47          BC    05,@9FF                                        0011    00410
                                    48 *           CODE=3;                         /*SET OPEN FAILURE CODE*/      00420
000026 41F0 0003           00003    49          LA    @F,3                                           0013    00430
00002A 50F0 B0C2           000C8    50          ST    @F,CODE                                        0013    00440
                                    51 *           RETURN;                         /*RETURN TO CALLING PROC*/     00450
00002E 47F0 B06A           0C070    52          BC    15,@EL01                                       0014    00460
                                    53 *           END;                                                          00470
                                    54 *GETNEXT: R0=ADDR(BUF);                     /*SET PTR TO INPUT             00480
                                    55 *                                             BUFFER*/                    00490
```

•Example 5. Use of the GENERATE Statements (Part 8 of 10)



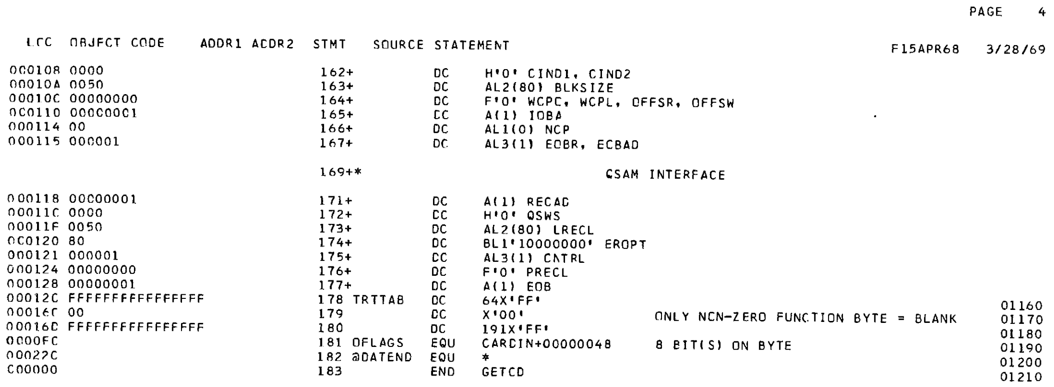
```
                                                                                                           PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2 STMT  SOURCE STATEMENT                                          F15APR68   3/28/69

000032                             56 @9FF     EQU   *                                                 0015    00500
000032 5810 BC76            0C07C  57 GETNEXT  L     @1,@A1             ADDRESS OF BUF                 0015    00510
000036 1801                        58          LR    @0,@1                                             0015    00520
                                   59 *         GEN(GET  CARCIN,(0)):            /*READ A RECORD*/             00530
                                   60          GET   CARDIN,(0)                                                00540
000038 4110 B0C6            000CC  61+         LA    1,CARDIN LCAD PARAMETER REG 1
00003C 58F0 1030            00030  62+         L     15,48(0,1) LCAD GET ROUTINE ADDR.
000040 05EF                        63+         BALR  14,15 LINK TO GET ROUTINE
000042                             64          DS    0H                                                        00550
                                   65 *         TRTPTR=ADDR(BUF):                /*SET PTR TO EXECUTE TRT*/    00560
000042 5810 B076            0007C  66          L     @1,@A1             ADDRESS OF BUF                 0018    00570
000046 1871                        67          LR    @7,@1                                             0018    00580
                                   68 *GENERATE:                                                               00590
000048 DD4F 7000 B126 00000 0012C  69          TRT   0(80,7),TRTTAB     IS THIS CARD BLANK                     00600
00004F 4780 B02C            00032  70          BC    8,GETNEXT          YES, GET NEXT CARD                     00610
000052                             71          DS    0H                                                        00620
                                   72 *         CODE=0:                          /*SET SUCCESSFUL READ*/       00630
000052 18FF                        73          SR    @F,@F                                             0020    00640
000054 50F0 B0C2            0C0C8  74          ST    @F,CODE                                           0020    00650
                                   75 *         RETURN;                          /*RETURN TO CALLING PROC*/    00660
000058 47F0 BC6A            0C070  76          BC    15,@EL01                                          0021    00670
                                   77 *ENDFILE: /*ON END OF FILE*/                                             00680
                                   78 *         CODE=1;                          /*SET END OF FILE CCDE*/      00690
00005C 41F0 0C01            0C001  79 ENCFILE  LA    @F,1                                              0022    00700
000060 50F0 B0C2            000C8  80          ST    @F,CODE                                           0022    00710
                                   81 *         RETURN:                          /*RETURN TO CALLING PROC*/    00720
000064 47F0 B06A            0C070  82          BC    15,@EL01                                          0023    00730
                                   83 *IOERRCR: /*IF AN I/O ERROR OCCURS*/                                     00740
                                   84 *         CODE=2;                          /*SET CODE FOR READ           00750
                                   85 *                                            ERROR*/                     00760
000068 41F0 0002            00002  86 IGERROR  LA    @F,2                                              0024    00770
00006C 50F0 B0C2            0C0C8  87          ST    @F,CCDE                                           0024    00780
                                   88 *         RETURN:                          /*RETURN TO CALLING PROC*/    00790
                                   89 *         END GETCD;                       /*END OF GETCD PROC*/         00800
0C0070 58D0 D004            0C004  90 @EL01    L     @D,4(0,@D)                                        0026    00810
000074 98EC D00C            0000C  91          LM    @E,@C,12(@C)                                      0026    00820
000078 07FE                        92          BCR   15,@E                                             0026    00830
00007A                             93 @DATA1   EQU   *                                                         00840
000000                             94 @0       EQU   00                 EQUATES FOR REGISTERS 0-15             00850
000001                             95 @1       EQU   01                                                        00860
000002                             96 @2       EQU   02                                                        00870
000003                             97 @3       EQU   03                                                        00880
000004                             98 @4       EQU   04                                                        00890
000005                             99 @5       EQU   05                                                        00900
000006                            100 @6       EQU   06                                                        00910
000007                            101 @7       EQU   07                                                        00920
000008                            102 @8       EQU   08                                                        00930
000009                            103 @9       EQU   09                                                        00940
00000A                            104 @A       EQU   10                                                        00950
00000B                            105 @B       EQU   11                                                        00960
00000C                            106 @C       EQU   12                                                        00970
00000D                            1C7 @C       EQU   13                                                        00980
00000E                            108 @E       EQU   14                                                        00990
00000F                            109 @F       EQU   15                                                        01000
00007A 0000
```

• Example 5. Use of the GENERATE Statements (Part 9 of 10)



```
                                                                                    PAGE    3

 LCC   OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                       F15APR68   3/28/69

00007C 00000000                    110 @A1       DC    A(BUF)                                      01010
0C0080                             111           DS    OF                                          01020
000080                             112           DS    OD                                          01030
0C0080                             113 @DATA     EQU   *                                           01040
0C0080                             114 @SAV001   EQU   @DATA+00000000       72 BYTE(S) ON WORD     01050
                                   115           EXTRN BUF                                         01060
                                   116           ENTRY CODE                                        01070
0000C8                             117 CODE      EQU   @DATA+00000072       FULLWORD INTEGER       01080
CC0000                             118 RO        EQU   00000000             FULLWORD POINTER REGISTER 01090
0C0007                             119 TRTPTR    EQU   00000007             FULLWORD POINTER REGISTER 01100
CC0080                             120           DS    00000076C                                   01110
0000CC                             121 @TEMPS    DS    OF                                          01120
                                   122 CARDIN    DCB   DSORG=PS,MACRF=(GM),DDNAME=BSLIN,OPTCD=C,RECFM=F,       X 01130
                                                       LRECL=80,BLKSIZE=80,BFTEK=S,BFALN=F,EODAD=ENDFILE,      X 01140
                                                       BUFNO=3,SYNAD=ICERRCR,EROPT=ACC                           01150


                                   124+*                       CATA CCNTROL BLOCK
                                   125+*
0C00CC                             126+          CRG   *-0 TO ELIMINATE UNUSEC SPACE
C000CC                             127+CARDIN    DS    OF ORIGIN CN WCRD BCUNDRY
0000CC                             128+          ORG   *+0 TO ORIGIN GENERATION

                                   130+*                       CIRECT ACCESS CEVICE INTERFACE

C000CC 0000000000000000            132+          DC    BL16'0' FDAD,DVTBL
C000DC 00000000                    133+          CC    A(0) KEYLE,DEVT,TRBAL

                                   135+*                       CCMMON ACCESS METHOD INTERFACE

0000F0 03                          137+          DC    AL1(3) BUFNO
0C00F1 000001                      138+          CC    AL3(1) BUFCB
0000E4 0000                        139+          DC    AL2(0) BUFL
0000F6 4000                        140+          CC    BL2'0100000000000000' DSORG
0000F8 00000001                    141+          DC    A(1) IOBAC

                                   143+*                       FCUNDATION EXTENSION

0000EC 41                          145+          DC    BL1'01000001' BFTEK,BFALN,HIARCHY
0000FD 00005C                      146+          CC    AL3(ENCFILE) EOCAD
0000F0 80                          147+          DC    BL1'10000000' RECFM
0000F1 000000                      148+          CC    AL3(0) EXLST

                                   150+*                       FCUNDATION BLOCK

0000F4 C2E2D3C9D5404040            152+          DC    CL8'BSLIN' DDNAME
0000FC 02                          153+          CC    BL1'00000010' OFLGS
C000FD 00                          154+          DC    BL1'00000000' IFLG
0000FE 5000                        155+          CC    BL2'0101000000000000' MACR

                                   157+*                       BSAM-BPAM-QSAM INTERFACE

C00100 20                          159+          DC    BL1'00100000' RER1
C00101 000001                      160+          CC    AL3(1) CHECK, GERR, PERR
000104 00000068                    161+          DC    A(IOERROR) SYNAD
```

• Example 5. Use of the GENERATE Statements (Part 10 of 10)

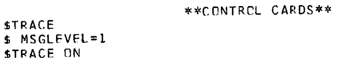


```
                                                                                        PAGE    4

  LFC  ORJFCT CODE     ADDR1 ACDR2  STMT   SOURCE STATEMENT                       F15APR68   3/28/69

0C0108 0000                          162+          DC     H'O' CIND1, CIND2
00010A 0050                          163+          DC     AL2(80) BLKSIZE
00010C 00000000                      164+          DC     F'O' WCPC, WCPL, OFFSR, OFFSW
0C0110 000C00C1                      165+          CC     A(1) IOBA                                    .
000114 00                            166+          DC     AL1(0) NCP
000115 000001                        167+          DC     AL3(1) EOBR, ECBAD

                                     169+*                                  CSAM INTERFACE

000118 00C00001                      171+          DC     A(1) RECAD
00011C 0000                          172+          CC     H'O' QSWS
00011F 0050                          173+          DC     AL2(80) LRECL
0C0120 80                            174+          DC     BL1'10000000' EROPT
000121 000001                        175+          CC     AL3(1) CNTRL
000124 00000000                      176+          DC     F'O' PRECL
000128 00000001                      177+          DC     A(1) EOB
00012C FFFFFFFFFFFFFFFF              178 TRTTAB    DC     64X'FF'                                         01160
00016C 00                            179           CC     X'00'            ONLY NCN-ZERO FUNCTION BYTE = BLANK   01170
00016D FFFFFFFFFFFFFFFF              180           DC     191X'FF'                                        01180
0000FC                               181 OFLAGS    EQU    CARCIN+00000048  8 BIT(S) ON BYTE               01190
00022C                               182 @DATEND   EQU    *                                               01200
C00000                               183           END    GETCD                                           01210
```

Appendix B: Example of Program Using BSL TRACE Option

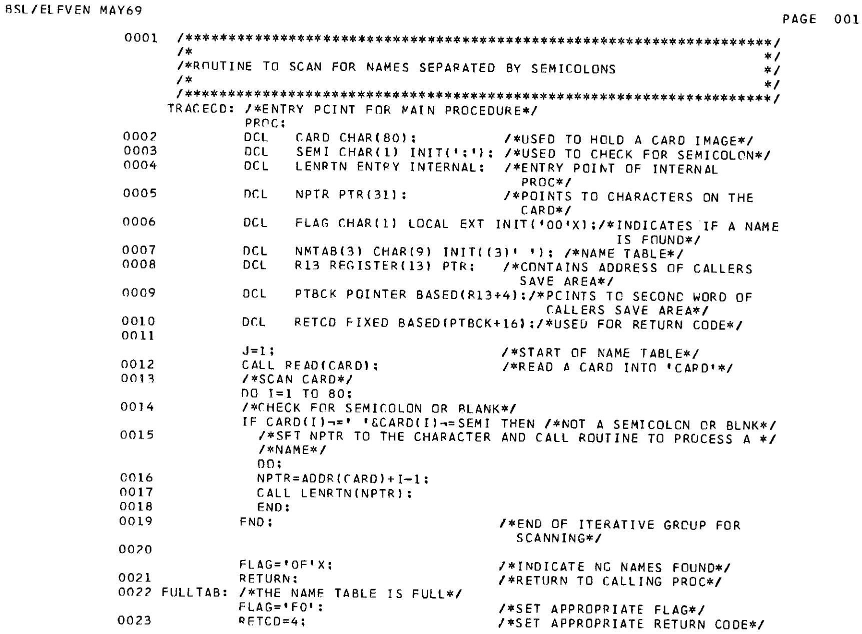
Example 6 illustrates the use of the BSL TRACE option.

- Example 6. Use of BSL TRACE Option (Part 1 of 11)



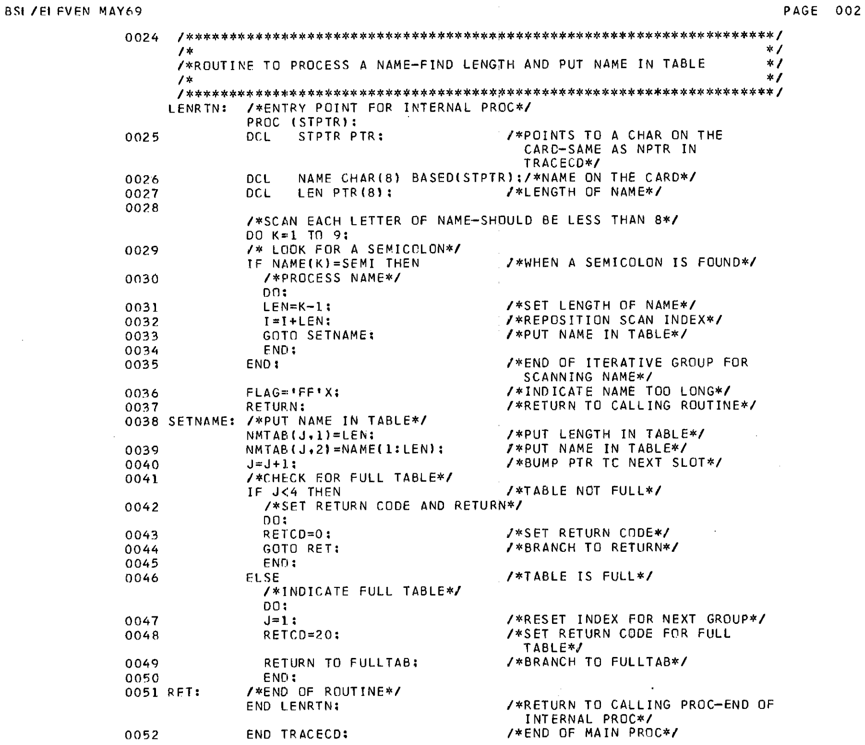
```
                    **CONTROL CARDS**
$TRACE
$ MSGLEVEL=1
$TRACE ON
```

• Example 6. Use of BSL TRACE Option (Part 2 of 11)



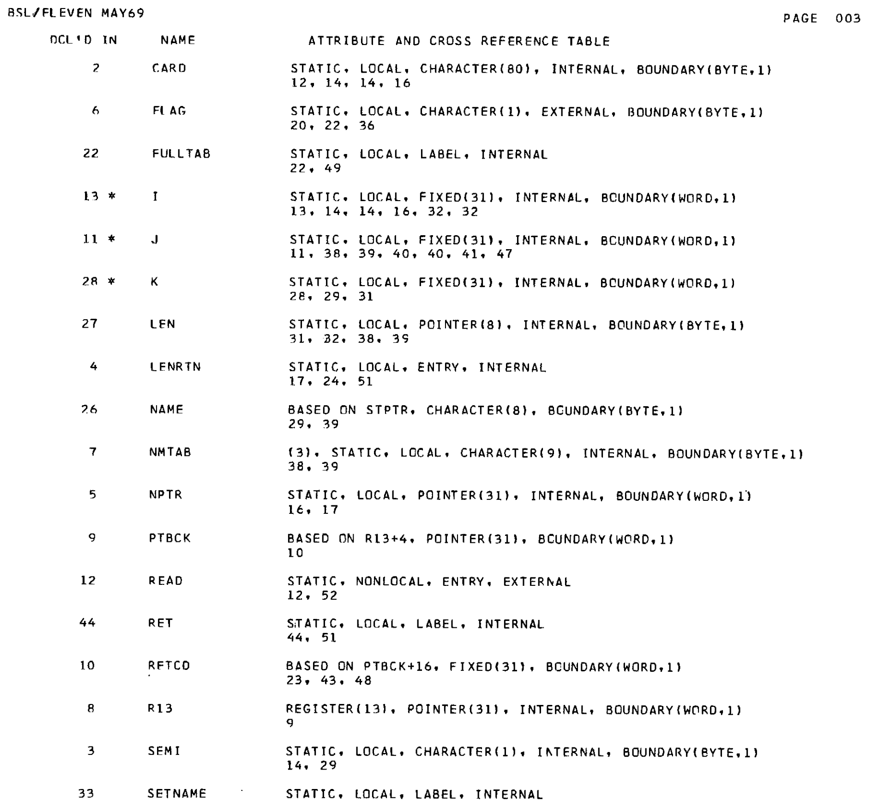
```
BSL/ELFVEN MAY69
                                                                                  PAGE  001
          0001   /******************************************************************/
                 /*                                                                */
                 /*ROUTINE TO SCAN FOR NAMES SEPARATED BY SEMICOLONS              */
                 /*                                                                */
                 /******************************************************************/
                TRACECD: /*ENTRY PCINT FOR MAIN PROCEDURE*/
                         PROC;
          0002           DCL   CARD CHAR(80);             /*USED TO HOLD A CARD IMAGE*/
          0003           DCL   SEMI CHAR(1) INIT(';'); /*USED TO CHECK FOR SEMICOLON*/
          0004           DCL   LENRTN ENTRY INTERNAL:  /*ENTRY POINT OF INTERNAL
                                                         PROC*/
          0005           DCL   NPTR PTR(31):             /*POINTS TO CHARACTERS ON THE
                                                         CARD*/
          0006           DCL   FLAG CHAR(1) LOCAL EXT INIT('00'X);/*INDICATES IF A NAME
                                                                 IS FOUND*/
          0007           DCL   NMTAB(3) CHAR(9) INIT((3)' '); /*NAME TABLE*/
          0008           DCL   R13 REGISTER(13) PTR:   /*CONTAINS ADDRESS OF CALLERS
                                                         SAVE AREA*/
          0009           DCL   PTBCK POINTER BASED(R13+4):/*PCINTS TC SECONC WORD OF
                                                            CALLERS SAVE AREA*/
          0010           DCL   RETCD FIXED BASED(PTBCK+16):/*USED FOR RETURN CODE*/
          0011
                         J=1;                             /*START OF NAME TABLE*/
          0012           CALL READ(CARD):                 /*READ A CARD INTO 'CARD'*/
          0013           /*SCAN CARD*/
                         DO I=1 TO 80:
          0014           /*CHECK FOR SEMICOLON OR BLANK*/
                         IF CARD(I)¬=' '&CARD(I)¬=SEMI THEN /*NOT A SEMICOLCN OR BLNK*/
          0015             /*SFT NPTR TO THE CHARACTER AND CALL ROUTINE TO PROCESS A */
                           /*NAME*/
                           DO;
          0016             NPTR=ADDR(CARD)+I-1:
          0017             CALL LENRTN(NPTR):
          0018             END:
          0019           FND;                             /*END OF ITERATIVE GRCUP FOR
                                                            SCANNING*/
          0020
                         FLAG='OF'X:                      /*INDICATE NG NAMES FOUND*/
          0021           RETURN:                          /*RETURN TO CALLING PROC*/
          0022 FULLTAB: /*THE NAME TABLE IS FULL*/
                         FLAG='FO':                       /*SET APPROPRIATE FLAG*/
          0023           RETCD=4;                         /*SET APPROPRIATE RETURN CODE*/
```

• Example 6. Use of BSL TRACE Option (Part 3 of 11)



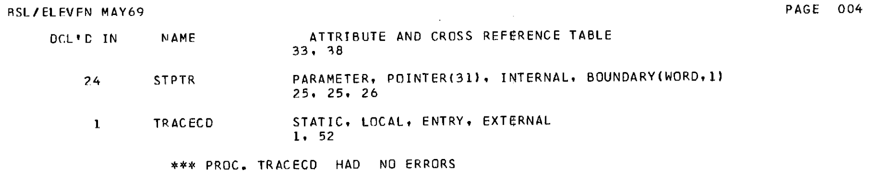
```
BSL /ELEVEN MAY69                                                                       PAGE  002
              0024 /**********************************************************************/
                   /*                                                                    */
                   /*ROUTINE TO PROCESS A NAME-FIND LENGTH AND PUT NAME IN TABLE         */
                   /*                                                                    */
                   /**********************************************************************/
                   LENRTN:  /*ENTRY POINT FOR INTERNAL PROC*/
                            PROC (STPTR):
              0025          DCL   STPTR PTR:                /*POINTS TO A CHAR ON THE
                                                              CARD-SAME AS NPTR IN
                                                              TRACECD*/
              0026          DCL   NAME CHAR(8) BASED(STPTR):/*NAME ON THE CARD*/
              0027          DCL   LEN PTR(8):               /*LENGTH OF NAME*/
              0028
                            /*SCAN EACH LETTER OF NAME-SHOULD BE LESS THAN 8*/
                            DO K=1 TO 9:
              0029          /* LOOK FOR A SEMICOLON*/
                            IF NAME(K)=SEMI THEN            /*WHEN A SEMICOLON IS FOUND*/
              0030            /*PROCESS NAME*/
                              DO:
              0031            LEN=K-1;                      /*SET LENGTH OF NAME*/
              0032            I=I+LEN:                      /*REPOSITION SCAN INDEX*/
              0033            GOTO SETNAME;                 /*PUT NAME IN TABLE*/
              0034            END:
              0035          END;                            /*END OF ITERATIVE GROUP FOR
                                                              SCANNING NAME*/
              0036          FLAG='FF'X;                     /*INDICATE NAME TOO LONG*/
              0037          RETURN:                         /*RETURN TO CALLING ROUTINE*/
              0038 SETNAME: /*PUT NAME IN TABLE*/
                            NMTAB(J,1)=LEN:                 /*PUT LENGTH IN TABLE*/
              0039          NMTAB(J,2)=NAME(1:LEN);         /*PUT NAME IN TABLE*/
              0040          J=J+1:                          /*BUMP PTR TC NEXT SLOT*/
              0041          /*CHECK FOR FULL TABLE*/
                            IF J<4 THEN                     /*TABLE NOT FULL*/
              0042            /*SET RETURN CODE AND RETURN*/
                              DO:
              0043            RETCD=0:                      /*SET RETURN CODE*/
              0044            GOTO RET:                     /*BRANCH TO RETURN*/
              0045            END:
              0046          ELSE                            /*TABLE IS FULL*/
                              /*INDICATE FULL TABLE*/
                              DO:
              0047            J=1:                          /*RESET INDEX FOR NEXT GROUP*/
              0048            RETCD=20:                     /*SET RETURN CODE FOR FULL
                                                              TABLE*/
              0049            RETURN TO FULLTAB;            /*BRANCH TO FULLTAB*/
              0050            END:
              0051 RET:     /*END OF ROUTINE*/
                            END LENRTN;                     /*RETURN TO CALLING PROC-END OF
                                                              INTERNAL PROC*/
              0052          END TRACECD:                    /*END OF MAIN PROC*/
```

• Example 6. Use of BSL TRACE Option (Part 4 of 11)



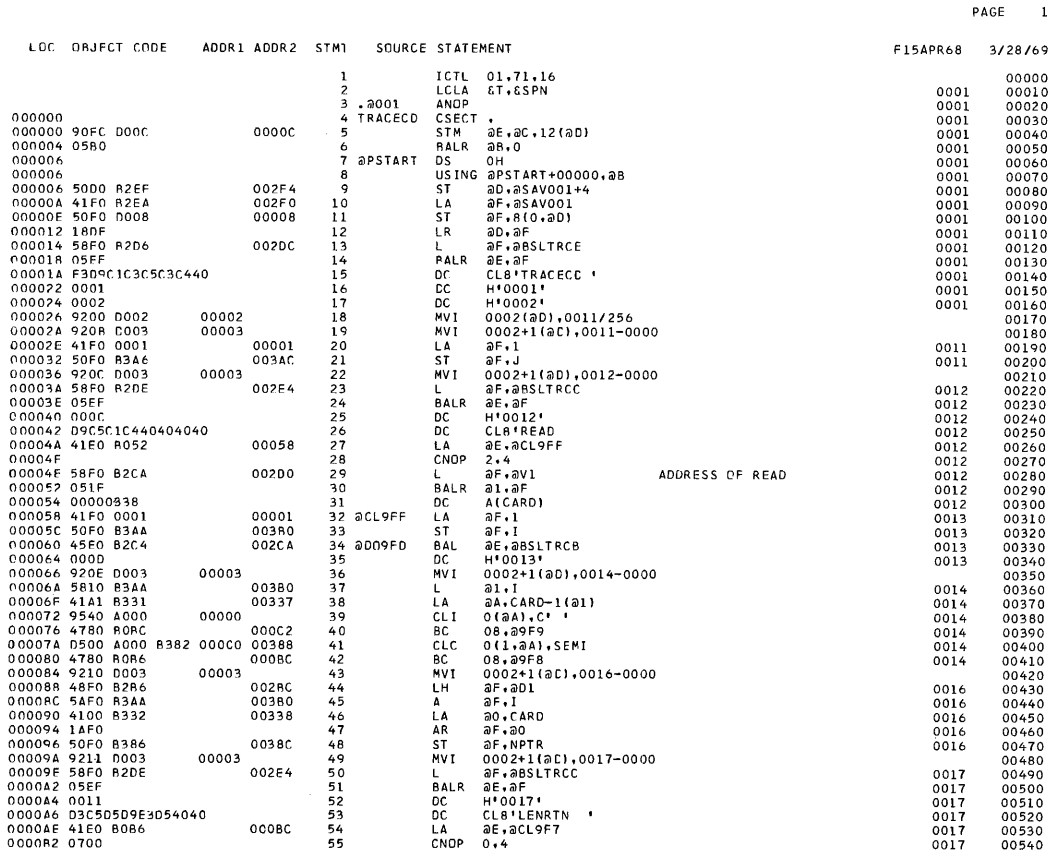
```
BSL/FLEVEN MAY69                                                                          PAGE  003
      DCL'D IN      NAME                 ATTRIBUTE AND CROSS REFERENCE TABLE

          2         CARD               STATIC, LOCAL, CHARACTER(80), INTERNAL, BOUNDARY(BYTE,1)
                                       12, 14, 14, 16

          6         FLAG               STATIC, LOCAL, CHARACTER(1), EXTERNAL, BOUNDARY(BYTE,1)
                                       20, 22, 36

         22         FULLTAB            STATIC, LOCAL, LABEL, INTERNAL
                                       22, 49

         13 *       I                  STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                       13, 14, 14, 16, 32, 32

         11 *       J                  STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                       11, 38, 39, 40, 40, 41, 47

         28 *       K                  STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                       28, 29, 31

         27         LEN                STATIC, LOCAL, POINTER(8), INTERNAL, BOUNDARY(BYTE,1)
                                       31, 32, 38, 39

          4         LENRTN             STATIC, LOCAL, ENTRY, INTERNAL
                                       17, 24, 51

         26         NAME               BASED ON STPTR, CHARACTER(8), BOUNDARY(BYTE,1)
                                       29, 39

          7         NMTAB              (3), STATIC, LOCAL, CHARACTER(9), INTERNAL, BOUNDARY(BYTE,1)
                                       38, 39

          5         NPTR               STATIC, LOCAL, POINTER(31), INTERNAL, BOUNDARY(WORD,1)
                                       16, 17

          9         PTBCK              BASED ON R13+4, POINTER(31), BOUNDARY(WORD,1)
                                       10

         12         READ               STATIC, NONLOCAL, ENTRY, EXTERNAL
                                       12, 52

         44         RET                STATIC, LOCAL, LABEL, INTERNAL
                                       44, 51

         10         RETCD              BASED ON PTBCK+16, FIXED(31), BOUNDARY(WORD,1)
                                       23, 43, 48

          8         R13                REGISTER(13), POINTER(31), INTERNAL, BOUNDARY(WORD,1)
                                       9

          3         SEMI               STATIC, LOCAL, CHARACTER(1), INTERNAL, BOUNDARY(BYTE,1)
                                       14, 29

         33         SETNAME            STATIC, LOCAL, LABEL, INTERNAL
```

• Example 6. Use of BSL TRACE Option (Part 5 of 11)



```
BSL/ELEVEN MAY69                                                                      PAGE  004
     DCL'D IN      NAME             ATTRIBUTE AND CROSS REFERENCE TABLE
                                  33, 38

        24        STPTR           PARAMETER, POINTER(31), INTERNAL, BOUNDARY(WORD,1)
                                  25, 25, 26

         1        TRACECD         STATIC, LOCAL, ENTRY, EXTERNAL
                                  1, 52

                *** PROC. TRACECD  HAD  NO ERRORS
```

• Example 6. Use of BSL TRACE Option (Part 6 of 11)



```
                                                                                                PAGE   1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                  F15APR68   3/28/69

                                      1          ICTL  01,71,16                                               00000
                                      2          LCLA  &T,&SPN                                        0001    00010
                                      3 .@001    ANOP                                                 0001    00020
000000                                4 TRACECD  CSECT ,                                              0001    00030
000000 90FC D00C           0000C      5          STM   @E,@C,12(@D)                                   0001    00040
000004 05B0                           6          BALR  @B,0                                           0001    00050
000006                                7 @PSTART  DS    0H                                             0001    00060
000006                                8          USING @PSTART+00000,@B                               0001    00070
000006 50D0 B2EF           002F4      9          ST    @D,@SAV001+4                                   0001    00080
00000A 41F0 B2EA           002F0     10          LA    @F,@SAV001                                     0001    00090
00000E 50F0 D008           00008     11          ST    @F,8(0,@D)                                     0001    00100
000012 18DF                          12          LR    @D,@F                                          0001    00110
000014 58F0 B2D6           002DC     13          L     @F,@BSLTRCE                                    0001    00120
000018 05EF                          14          BALR  @E,@F                                          0001    00130
00001A F3D9C1C3C5C3C440              15          DC    CL8'TRACECC '                                  0001    00140
000022 0001                          16          DC    H'0001'                                        0001    00150
000024 0002                          17          DC    H'0002'                                        0001    00160
000026 9200 D002     00002           18          MVI   0002(@D),0011/256                                      00170
00002A 920B D003     00003           19          MVI   0002+1(@C),0011-0000                                   00180
00002E 41F0 0001           00001     20          LA    @F,1                                           0011    00190
000032 50F0 B3A6           003AC     21          ST    @F,J                                           0011    00200
000036 920C D003     00003           22          MVI   0002+1(@D),0012-0000                                   00210
00003A 58F0 B2DE           002E4     23          L     @F,@BSLTRCC                                    0012    00220
00003E 05EF                          24          BALR  @E,@F                                          0012    00230
000040 000C                          25          DC    H'0012'                                        0012    00240
000042 D9C5C1C440404040              26          DC    CL8'READ                                       0012    00250
00004A 41E0 B052           00058     27          LA    @E,@CL9FF                                      0012    00260
00004F                               28          CNOP  2,4                                            0012    00270
00004E 58F0 B2CA           002D0     29          L     @F,@V1              ADDRESS OF READ            0012    00280
000052 051F                          30          BALR  @1,@F                                          0012    00290
000054 00000838                      31          DC    A(CARD)                                        0012    00300
000058 41F0 0001           00001     32 @CL9FF   LA    @F,1                                           0013    00310
00005C 50F0 B3AA           003B0     33          ST    @F,I                                           0013    00320
000060 45E0 B2C4           002CA     34 @D09FD   BAL   @E,@BSLTRCB                                    0013    00330
000064 000D                          35          DC    H'0013'                                        0013    00340
000066 920E D003     00003           36          MVI   0002+1(@D),0014-0000                                   00350
00006A 5810 B3AA           003B0     37          L     @1,I                                           0014    00360
00006F 41A1 B331           00337     38          LA    @A,CARD-1(@1)                                  0014    00370
000072 9540 A000     00000           39          CLI   0(@A),C' '                                     0014    00380
000076 4780 B0BC           000C2     40          BC    08,@9F9                                        0014    00390
00007A D500 A000 B382 000C0 00388    41          CLC   0(1,@A),SEMI                                   0014    00400
000080 4780 B0B6           000BC     42          BC    08,@9F8                                        0014    00410
000084 9210 D003     00003           43          MVI   0002+1(@C),0016-0000                                   00420
000088 48F0 B2B6           002BC     44          LH    @F,@D1                                         0016    00430
00008C 5AF0 B3AA           003B0     45          A     @F,I                                           0016    00440
000090 4100 B332           00338     46          LA    @0,CARD                                        0016    00450
000094 1AF0                          47          AR    @F,@0                                          0016    00460
000096 50F0 B386           0038C     48          ST    @F,NPTR                                        0016    00470
00009A 9211 D003     00003           49          MVI   0002+1(@C),0017-0000                                   00480
00009E 58F0 B2DE           002E4     50          L     @F,@BSLTRCC                                    0017    00490
0000A2 05EF                          51          BALR  @E,@F                                          0017    00500
0000A4 0011                          52          DC    H'0017'                                        0017    00510
0000A6 D3C5D5D9E3D54040              53          DC    CL8'LENRTN  '                                  0017    00520
0000AE 41E0 B0B6           000BC     54          LA    @E,@CL9F7                                      0017    00530
0000B2 0700                          55          CNOP  0,4                                            0017    00540
```

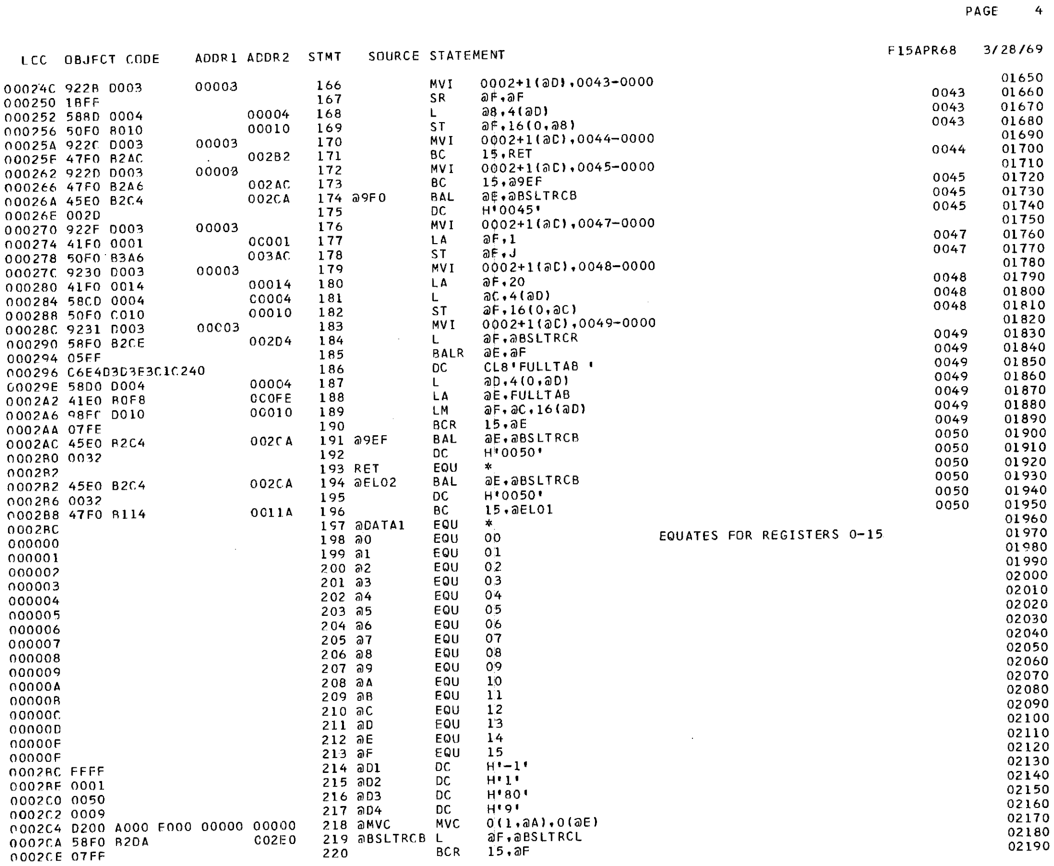
• Example 6. Use of BSL TRACE Option (Part 7 of 11)

```
                                                                                            PAGE    2

  LCC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                            F15APR68   3/28/69

0000B4 4510 B132            00138   56          BAL   @1,LENRTN                               0017   00550
0000B8 0000038C                     57          DC    A(NPTR)                                 0017   00560
0000BC                              58 @CL9F7   EQU   *                                       0017   00570
0000BC 45E0 B2C4            002CA   59 @9F8     BAL   @E,@BSLTRCB                             0018   00580
0000C0 0012                         60          DC    H'0018'                                 0018   00590
0000C2 45E0 B2C4            002CA   61 @9F9     BAL   @E,@BSLTRCB                             0018   00600
0000C6 0012                         62          DC    H'0018'                                 0018   00610
0000C8 9213 D003      00003         63          MVI   0002+1(@C),0019-0000                           00620
0000CC 58F0 B3AA            003B0   64          L     @F,I                                    0019   00630
0000D0 4AF0 B2B8            002BE   65          AH    @F,@D2                                  0019   00640
0000D4 50F0 B3AA            003B0   66 @D09FC   ST    @F,I                                    0019   00650
0000D8 50F0 B3FE            00404   67          ST    @F,@T1                                  0019   00660
0000DC 45E0 B2C4            002CA   68          BAL   @E,@BSLTRCB                             0019   00670
0000E0 0013                         69          DC    H'0019'                                 0019   00680
0000E2 58F0 B3FE            00404   70          L     @F,@T1                                  0019   00690
0000E6 49F0 B2BA            002C0   71          CH    @F,@D3                                  0019   00700
0000EA 47C0 B05A            00060   72          BC    12,@D09FC                               0019   00710
0000EE 9214 D003      00003         73          MVI   0002+1(@C),0020-0000                           00720
0000F2 920F B38A      00390         74          MVI   FLAG,X'0F'                              0020   00730
0000F6 9215 D003      00003         75          MVI   0002+1(@C),0021-0000                           00740
0000FA 47F0 B114            0011A   76          BC    15,@EL01                                0021   00750
0000FE 45E0 B2C4            002CA   77 FULLTAB  BAL   @E,@BSLTRCB                             0022   00760
000102 0016                         78          DC    H'0022'                                 0022   00770
000104 D200 B38A B2E2 00390 002E8   79          MVC   FLAG(1),@C3                             0022   00780
00010A 9217 D003      00003         80          MVI   0002+1(@C),0023-0000                           00790
00010E 41F0 0004            00004   81          LA    @F,4                                    0023   00800
000112 581D 0004            00004   82          L     @1,4(@D)                                0023   00810
000116 50F0 1010            00010   83          ST    @F,16(0,@1)                             0023   00820
00011A 45E0 B2C4            002CA   84 @EL01    BAL   @E,@BSLTRCB                             0024   00830
00011E 0018                         85          DC    H'0024'                                 0024   00840
000120 58F0 B2CE            002D4   86          L     @F,@BSLTRCR                             0024   00850
000124 05EF                         87          BALR  @E,@F                                   0024   00860
000126 60C3C1D3D3C5D960             88          DC    CL8'-CALLER-'                           0024   00870
00012E 58D0 D004            00004   89          L     @D,4(0,@D)                              0024   00880
000132 98EC D00C            0000C   90          LM    @E,@C,12(@D)                            0024   00890
000136 07FE                         91          BCR   15,@E                                   0024   00900
000138 90EC D00C            0000C   92 LENRTN   STM   @E,@C,12(@D)                            0024   00910
00013C 50D0 B3B2            003B8   93          ST    @D,@SAV002+4                            0024   00920
000140 41F0 B3AE            003B4   94          LA    @F,@SAV002                              0024   00930
000144 50F0 D008            00008   95          ST    @F,8(0,@D)                              0024   00940
000148 18DF                         96          LR    @D,@F                                   0024   00950
00014A 58F0 B2D6            002DC   97          L     @F,@BSLTRCE                             0024   00960
00014E 05EF                         98          BALR  @E,@F                                   0024   00970
000150 D3C5D5D9E3D54040             99          DC    CL8'LENRTN '                            0024   00980
000158 0018                        100          DC    H'0024'                                 0024   00990
00015A 0002                        101          DC    H'0002'                                 0024   01000
00015C 41F0 0001            00001  102          LA    @F,1                                    0028   01010
000160 50F0 B3FA            00400  103          ST    @F,K                                    0028   01020
000164 45E0 B2C4            002CA  104 @D09F5   BAL   @E,@BSLTRCB                             0028   01030
000168 001C                        105          DC    H'0028'                                 0028   01040
00016A 921D D003      00003        106          MVI   0002+1(@C),0029-0000                           01050
00016E 58C0 B3FA            00400  107          L     @C,K                                    0029   01060
000172 5880 1000            00000  108          L     @8,0(0,@1)            ADDRESS OF STPTR  0029   01070
000176 5898 0000            00000  109          L     @9,0(@8)              STPTR             0029   01080
00017A 41AC 9000            00000  110          LA    @A,0(@C,@9)                             0029   01090
```

• Example 6. Use of BSL TRACE Option (Part 8 of 11)

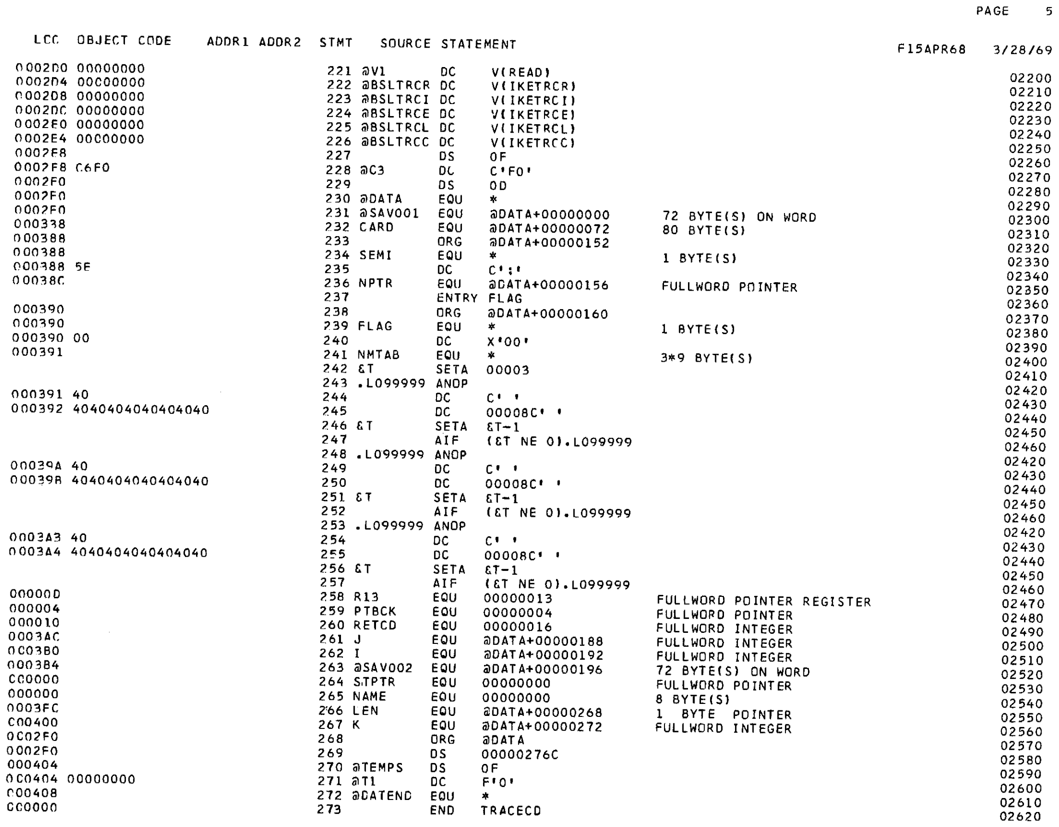
```
                                                                                                   PAGE    3

  LCC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                   F15APR68  3/28/69

00017E 06A0                         111          BCTR  @A,0                                       0029    01100
000180 D500 A000 B382 00000 00388   112          CLC   0(1,@A),SEMI                               0029    01110
000186 4770 B1AC            001B2   113          BC    07,@9F1                                    0029    01120
00018A 921F D003      00003         114          MVI   0002+1(@C),0031-0000                               01130
00018F 58F0 B3FA            00400   115          L     @F,K                                       0031    01140
000192 06F0                         116          BCTR  @F,0                                       0031    01150
000194 42F0 B3F6            003FC   117          STC   @F,LEN                                     0031    01160
000198 9220 D003      00003         118          MVI   0002+1(@C),0032-0000                               01170
00019C 18FF                         119          SR    @F,@F                                      0032    01180
00019F 43F0 B3F6            003FC   120          IC    @F,LEN                                     0032    01190
0001A2 5AF0 B3AA            C03B0   121          A     @F,I                                       0032    01200
0001A6 50F0 B3AA            003B0   122          ST    @F,I                                       0032    01210
0001AA 9221 DC03      00003         123          MVI   0002+1(@C),0033-0000                               01220
0001AE 47F0 B1E8            001EE   124          BC    15,SETNAME                                 0033    01230
0001B2 45E0 B2C4            002CA   125 @9F1     BAL   @E,@BSLTRCB                                0034    01240
0001B6 0022                         126          DC    H'0034'                                    0034    01250
0001B8 9223 D003      00003         127          MVI   0002+1(@C),0035-0000                               01260
0001BC 58F0 B3FA            004C0   128          L     @F,K                                       0035    01270
0001C0 4AF0 B2B8            002BE   129          AH    @F,@D2                                     0035    01280
0001C4 50F0 B3FA            C0400   130 @DD9F4   ST    @F,K                                       0035    01290
0001C8 50F0 B3FE            00404   131          ST    @F,@T1                                     0035    01300
0001CC 45E0 B2C4            0C2CA   132          BAL   @E,@BSLTRCB                                0035    01310
0001D0 0023                         133          DC    H'0035'                                    0035    01320
0001D2 58F0 B3FE            00404   134          L     @F,@T1                                     0035    01330
0001D6 49F0 B2BC            C02C2   135          CH    @F,@D4                                     0035    01340
0001DA 47C0 B15F            00164   136          BC    12,@DD9F5                                  0035    01350
0001DE 9224 D003      00003         137          MVI   0002+1(@C),0036-0000                               01360
0001E2 92FF B38A      00390         138          MVI   FLAG,X'FF'                                 0036    01370
0001E6 9225 DC03      00003         139          MVI   0002+1(@C),0037-0000                               01380
0001EA 47F0 B114            0011A   140          BC    15,@EL01                                   0037    01390
0001EE 45E0 B2C4            002CA   141 SETNAME  BAL   @E,@BSLTRCB                                0038    01400
0001F2 0026                         142          DC    H'0038'                                    0038    01410
0001F4 58C0 B3A6            003AC   143          L     @C,J                                       0038    01420
0001F8 4CC0 B2BC            002C2   144          MH    @C,@D4                                     0038    01430
0001FC 41AC B382            00388   145          LA    @A,NMTAB-9(@C)                             0038    01440
000200 D200 A000 B3F6 00000 003FC   146          MVC   0(1,@A),LEN                                0038    01450
000206 9227 D003      00003         147          MVI   0002+1(@C),0039-0000                               01460
00020A 1B88                         148          SR    @8,@8                                      0039    01470
00020C 4380 B3F6            003FC   149          IC    @8,LEN                                     0039    01480
000210 41AC B383            00389   150          LA    @A,NMTAB-8(@C)                             0039    01490
000214 4190 0001            00001   151          LA    @9,1                                       0039    01500
000218 5860 1000            00000   152          L     @6,0(0,@1)           ADDRESS OF STPTR      0039    01510
00021C 5876 0000            00000   153          L     @7,0(@6)             STPTR                 0039    01520
000220 41E9 7000            00000   154          LA    @E,0(@9,@7)                                0039    01530
000224 06E0                         155          BCTR  @E,0                                       0039    01540
000226 1B89                         156          SR    @8,@9                                      0039    01550
000228 4480 B2BE            002C4   157          EX    @8,@MVC                                    0039    01560
00022C 9228 D003      00003         158          MVI   0002+1(@C),0040-0000                               01570
000230 41F0 0001            00001   159          LA    @F,1                                       0040    01580
000234 5AF0 B3A6            003AC   160          A     @F,J                                       0040    01590
000238 50F0 B3A6            0C3AC   161          ST    @F,J                                       0040    01600
00023C 9229 D003      00003         162          MVI   0002+1(@C),0041-0000                               01610
000240 41F0 0004            00004   163          LA    @F,4                                       0041    01620
000244 59F0 B3A6            003AC   164          C     @F,J                                       0041    01630
000248 47C0 B264            0026A   165          BC    12,@9F0                                    0041    01640
```

• Example 6. Use of BSL TRACE Option (Part 9 of 11)



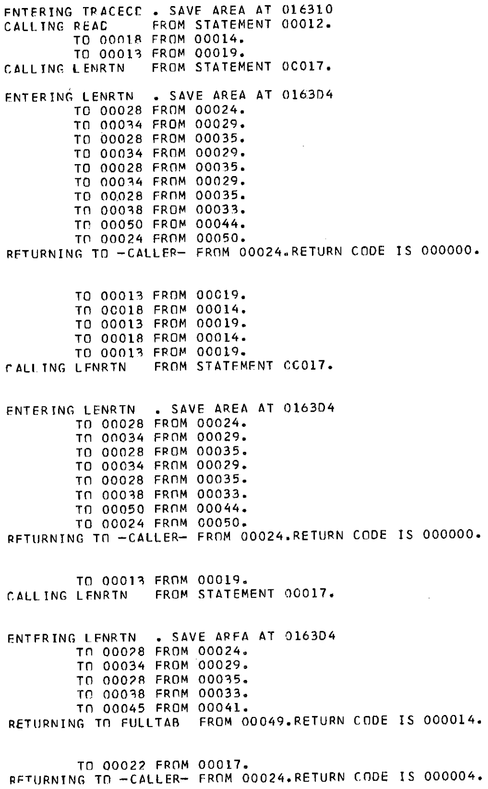
```
                                                                                                              PAGE    4

  LCC   OBJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATEMENT                                           F15APR68   3/28/69

00024C 922B D003        00003         166              MVI   0002+1(@D),0043-0000                                     01650
000250 1BFF                           167              SR    @F,@F                                           0043     01660
000252 588D 0004              00004   168              L     @8,4(@D)                                        0043     01670
000256 50F0 8010              00010   169              ST    @F,16(0,@8)                                     0043     01680
00025A 922C D003        00003         170              MVI   0002+1(@C),0044-0000                                     01690
00025E 47F0 B2AC              002B2   171              BC    15,RET                                          0044     01700
000262 922D D003        00003         172              MVI   0002+1(@C),0045-0000                                     01710
000266 47F0 B2A6              002AC   173              BC    15,@9EF                                         0045     01720
00026A 45E0 B2C4              002CA   174 @9F0         BAL   @E,@BSLTRCB                                     0045     01730
00026E 002D                           175              DC    H'0045'                                         0045     01740
000270 922F D003        00003         176              MVI   0002+1(@C),0047-0000                                     01750
000274 41F0 0001              0C001   177              LA    @F,1                                            0047     01760
000278 50F0 B3A6              003AC   178              ST    @F,J                                            0047     01770
00027C 9230 D003        00003         179              MVI   0002+1(@C),0048-0000                                     01780
000280 41F0 0014              00014   180              LA    @F,20                                           0048     01790
000284 58CD 0004              C0004   181              L     @C,4(@D)                                        0048     01800
000288 50F0 C010              00010   182              ST    @F,16(0,@C)                                     0048     01810
00028C 9231 D003        00C03         183              MVI   0002+1(@C),0049-0000                                     01820
000290 58F0 B2CE              002D4   184              L     @F,@BSLTRCR                                     0049     01830
000294 05EF                           185              BALR  @E,@F                                           0049     01840
000296 C6E4D3D3E3C1C240               186              DC    CL8'FULLTAB '                                   0049     01850
00029E 58D0 D004              00004   187              L     @D,4(0,@D)                                      0049     01860
0002A2 41E0 B0F8              0C0FE   188              LA    @E,FULLTAB                                      0049     01870
0002A6 98FC D010              00010   189              LM    @F,@C,16(@D)                                    0049     01880
0002AA 07FE                           190              BCR   15,@E                                           0049     01890
0002AC 45E0 B2C4              002CA   191 @9EF         BAL   @E,@BSLTRCB                                     0050     01900
0002B0 0032                           192              DC    H'0050'                                         0050     01910
0002B2                                193 RET          EQU   *                                               0050     01920
0002B2 45E0 B2C4              002CA   194 @EL02        BAL   @E,@BSLTRCB                                     0050     01930
0002B6 0032                           195              DC    H'0050'                                         0050     01940
0002B8 47F0 B114              0011A   196              BC    15,@EL01                                        0050     01950
0002BC                                197 @DATA1       EQU   *                                                        01960
000000                                198 @0           EQU   00                 EQUATES FOR REGISTERS 0-15            01970
000001                                199 @1           EQU   01                                                       01980
000002                                200 @2           EQU   02                                                       01990
000003                                201 @3           EQU   03                                                       02000
000004                                202 @4           EQU   04                                                       02010
000005                                203 @5           EQU   05                                                       02020
000006                                204 @6           EQU   06                                                       02030
000007                                205 @7           EQU   07                                                       02040
000008                                206 @8           EQU   08                                                       02050
000009                                207 @9           EQU   09                                                       02060
00000A                                208 @A           EQU   10                                                       02070
00000B                                209 @B           EQU   11                                                       02080
00000C                                210 @C           EQU   12                                                       02090
00000D                                211 @D           EQU   13                                                       02100
00000F                                212 @E           EQU   14                                                       02110
00000F                                213 @F           EQU   15                                                       02120
0002BC FFFF                           214 @D1          DC    H'-1'                                                    02130
0002BE 0001                           215 @D2          DC    H'1'                                                     02140
0002C0 0050                           216 @D3          DC    H'80'                                                    02150
0002C2 0009                           217 @D4          DC    H'9'                                                     02160
0002C4 D200 A000 E000 00000 00000     218 @MVC         MVC   0(1,@A),0(@E)                                            02170
0002CA 58F0 B2DA              C02E0   219 @BSLTRCB     L     @F,@BSLTRCL                                              02180
0002CE 07FF                           220              BCR   15,@F                                                    02190
```

• Example 6. Use of BSL TRACE Option (Part 10 of 11)



```
                                                                                                          PAGE    5

  LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                           F15APR68   3/28/69

0002D0 00000000                      221 @V1       DC    V(READ)                                                 02200
0002D4 00000000                      222 @BSLTRCR  DC    V(IKETRCR)                                              02210
0002D8 00000000                      223 @BSLTRCI  DC    V(IKETRCI)                                              02220
0002DC 00000000                      224 @BSLTRCE  DC    V(IKETRCE)                                              02230
0002E0 00000000                      225 @BSLTRCL  DC    V(IKETRCL)                                              02240
0002E4 00000000                      226 @BSLTRCC  DC    V(IKETRCC)                                              02250
0002E8                               227           DS    0F                                                      02260
0002E8 C6F0                          228 @C3       DC    C'F0'                                                   02270
0002F0                               229           DS    0D                                                      02280
0002F0                               230 @DATA     EQU   *                                                       02290
0002F0                               231 @SAV001   EQU   @DATA+00000000     72 BYTE(S) ON WORD                   02300
000338                               232 CARD      EQU   @DATA+00000072     80 BYTE(S)                           02310
000388                               233           ORG   @DATA+00000152                                          02320
000388                               234 SEMI      EQU   *                  1 BYTE(S)                            02330
000388 5E                            235           DC    C';'                                                    02340
00038C                               236 NPTR      EQU   @DATA+00000156     FULLWORD POINTER                     02350
                                     237           ENTRY FLAG                                                    02360
000390                               238           ORG   @DATA+00000160                                          02370
000390                               239 FLAG      EQU   *                  1 BYTE(S)                            02380
000390 00                            240           DC    X'00'                                                   02390
000391                               241 NMTAB     EQU   *                  3*9 BYTE(S)                          02400
                                     242 &T        SETA  00003                                                   02410
                                     243 .L099999  ANOP                                                          02420
000391 40                            244           DC    C' '                                                    02430
000392 4040404040404040              245           DC    00008C' '                                               02440
                                     246 &T        SETA  &T-1                                                    02450
                                     247           AIF   (&T NE 0).L099999                                       02460
                                     248 .L099999  ANOP                                                          02420
00039A 40                            249           DC    C' '                                                    02430
00039B 4040404040404040              250           DC    00008C' '                                               02440
                                     251 &T        SETA  &T-1                                                    02450
                                     252           AIF   (&T NE 0).L099999                                       02460
                                     253 .L099999  ANOP                                                          02420
0003A3 40                            254           DC    C' '                                                    02430
0003A4 4040404040404040              255           DC    00008C' '                                               02440
                                     256 &T        SETA  &T-1                                                    02450
                                     257           AIF   (&T NE 0).L099999                                       02460
00000D                               258 R13       EQU   00000013           FULLWORD POINTER REGISTER            02470
000004                               259 PTBCK     EQU   00000004           FULLWORD POINTER                     02480
000010                               260 RETCD     EQU   00000016           FULLWORD INTEGER                     02490
0003AC                               261 J         EQU   @DATA+00000188     FULLWORD INTEGER                     02500
0003B0                               262 I         EQU   @DATA+00000192     FULLWORD INTEGER                     02510
0003B4                               263 @SAV002   EQU   @DATA+00000196     72 BYTE(S) ON WORD                   02520
000000                               264 STPTR     EQU   00000000           FULLWORD POINTER                     02530
000000                               265 NAME      EQU   00000000           8 BYTE(S)                            02540
0003FC                               266 LEN       EQU   @DATA+00000268     1  BYTE  POINTER                     02550
000400                               267 K         EQU   @DATA+00000272     FULLWORD INTEGER                     02560
0002F0                               268           ORG   @DATA                                                   02570
0002F0                               269           DS    00000276C                                               02580
000404                               270 @TEMPS    DS    0F                                                      02590
000404 00000000                      271 @T1       DC    F'0'                                                    02600
000408                               272 @DATEND   EQU   *                                                       02610
000000                               273           END   TRACECD                                                 02620
```

• Example 6. Use of BSL TRACE Option (Part 11 of 11)

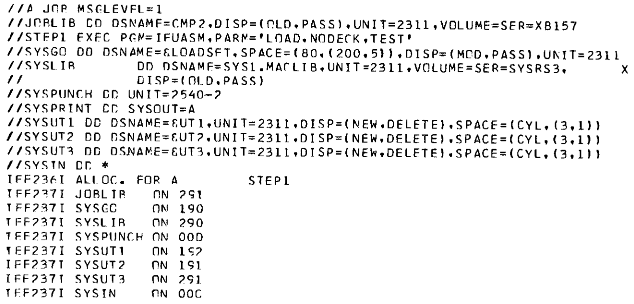


```
ENTERING TRACECD . SAVE AREA AT 016310
CALLING READ      FROM STATEMENT 00012.
        TO 00018 FROM 00014.
        TO 00013 FROM 00019.
CALLING LENRTN    FROM STATEMENT 00017.

ENTERING LENRTN   . SAVE AREA AT 0163D4
        TO 00028 FROM 00024.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00038 FROM 00033.
        TO 00050 FROM 00044.
        TO 00024 FROM 00050.
RETURNING TO -CALLER- FROM 00024.RETURN CODE IS 000000.


        TO 00013 FROM 00019.
        TO 00018 FROM 00014.
        TO 00013 FROM 00019.
        TO 00018 FROM 00014.
        TO 00013 FROM 00019.
CALLING LENRTN    FROM STATEMENT 00017.


ENTERING LENRTN   . SAVE AREA AT 0163D4
        TO 00028 FROM 00024.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00038 FROM 00033.
        TO 00050 FROM 00044.
        TO 00024 FROM 00050.
RETURNING TO -CALLER- FROM 00024.RETURN CODE IS 000000.


        TO 00013 FROM 00019.
CALLING LENRTN    FROM STATEMENT 00017.


ENTERING LENRTN   . SAVE AREA AT 0163D4
        TO 00028 FROM 00024.
        TO 00034 FROM 00029.
        TO 00028 FROM 00035.
        TO 00038 FROM 00033.
        TO 00045 FROM 00041.
RETURNING TO FULLTAB  FROM 00049.RETURN CODE IS 000014.


        TO 00022 FROM 00017.
RETURNING TO -CALLER- FROM 00024.RETURN CODE IS 000004.
```

Appendix C: Example of BSL Program Using OS TESTRAN

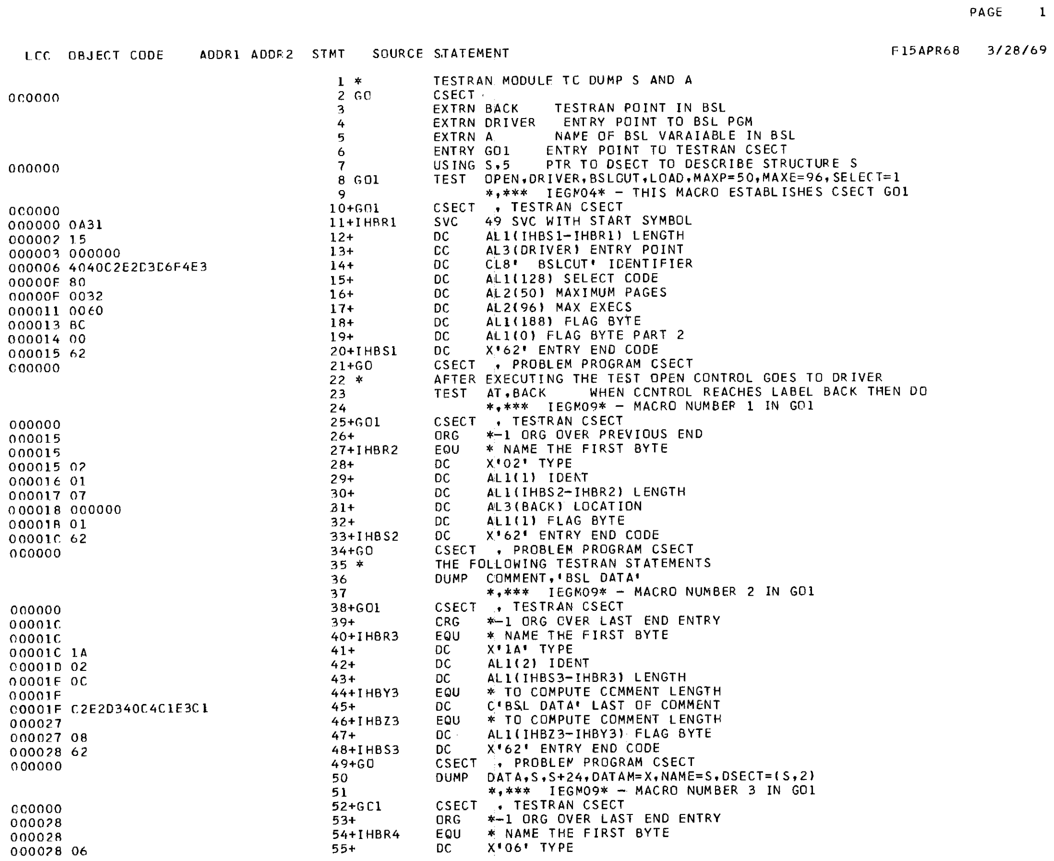
Example 7 illustrates the use of OS TESTRAN with your BSL program. The example shows a method of displaying two areas in the BSL program. One area is AUTOMATIC, the other is STATIC, and both areas are displayed at the label BACK in the BSL program.

- Example 7. Use of OS TESTRAN (Part 1 of 17)



```
//A JOB MSGLEVEL=1
//JOBLIB DD DSNAME=CMP2,DISP=(OLD,PASS),UNIT=2311,VOLUME=SER=XB157
//STEP1 EXEC PGM=IFUASM,PARM='LOAD,NODECK,TEST'
//SYSGO DD DSNAME=&LOADSET,SPACE=(80,(200,5)),DISP=(MOD,PASS),UNIT=2311
//SYSLIB        DD DSNAME=SYS1.MACLIB,UNIT=2311,VOLUME=SER=SYSRS3,        X
//              DISP=(OLD,PASS)
//SYSPUNCH DD UNIT=2540-2
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSNAME=&UT1,UNIT=2311,DISP=(NEW,DELETE),SPACE=(CYL,(3,1))
//SYSUT2 DD DSNAME=&UT2,UNIT=2311,DISP=(NEW,DELETE),SPACE=(CYL,(3,1))
//SYSUT3 DD DSNAME=&UT3,UNIT=2311,DISP=(NEW,DELETE),SPACE=(CYL,(3,1))
//SYSIN DD *
IEF236I ALLOC. FOR A        STEP1
IEF237I JOBLIB   ON 291
IEF237I SYSGO    ON 190
IEF237I SYSLIB   ON 290
IEF237I SYSPUNCH ON 00D
IEF237I SYSUT1   ON 192
IEF237I SYSUT2   ON 191
IEF237I SYSUT3   ON 291
IEF237I SYSIN    ON 00C
```

• Example 7. Use of OS TESTRAN (Part 2 of 17)

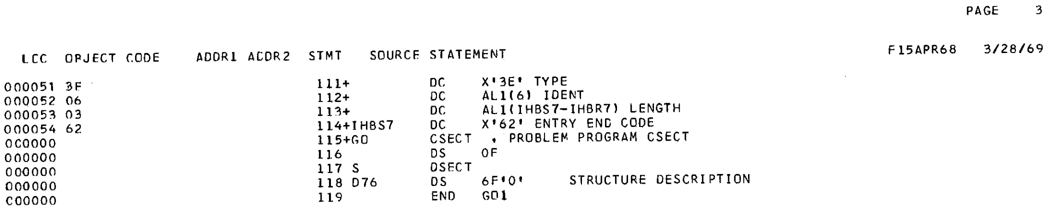


```
                                                                                                               PAGE    1

  LCC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                                F15APR68   3/28/69

                                     1 *         TESTRAN MODULE TC DUMP S AND A
000000                               2 GO        CSECT
                                     3           EXTRN BACK    TESTRAN POINT IN BSL
                                     4           EXTRN DRIVER   ENTRY POINT TO BSL PGM
                                     5           EXTRN A        NAME OF BSL VARAIABLE IN BSL
                                     6           ENTRY GO1     ENTRY POINT TO TESTRAN CSECT
000000                               7           USING S,5     PTR TO DSECT TO DESCRIBE STRUCTURE S
                                     8 GO1       TEST  OPEN,DRIVER,BSLCUT,LOAD,MAXP=50,MAXE=96,SELECT=1
                                     9                 *,***  IEGM04* - THIS MACRO ESTABLISHES CSECT GO1
000000                              10+GO1       CSECT  , TESTRAN CSECT
000000 0A31                         11+IHBR1     SVC   49 SVC WITH START SYMBOL
000002 15                           12+          DC    AL1(IHBS1-IHBR1) LENGTH
000003 000000                       13+          CC    AL3(DRIVER) ENTRY POINT
000006 4040C2E2D3D6F4E3             14+          DC    CL8'  BSLCUT' IDENTIFIER
00000F 80                           15+          DC    AL1(128) SELECT CODE
00000F 0032                         16+          DC    AL2(50) MAXIMUM PAGES
000011 0060                         17+          DC    AL2(96) MAX EXECS
000013 BC                           18+          DC    AL1(188) FLAG BYTE
000014 00                           19+          DC    AL1(0) FLAG BYTE PART 2
000015 62                           20+IHBS1     DC    X'62' ENTRY END CODE
000000                              21+GO        CSECT  , PROBLEM PROGRAM CSECT
                                    22 *         AFTER EXECUTING THE TEST OPEN CONTROL GOES TO DRIVER
                                    23           TEST  AT,BACK     WHEN CCNTROL REACHES LABEL BACK THEN DO
                                    24                 *,***  IEGM09* - MACRO NUMBER 1 IN GO1
000000                              25+GO1       CSECT  , TESTRAN CSECT
000015                              26+          ORG   *-1 ORG OVER PREVIOUS END
000015                              27+IHBR2     EQU   * NAME THE FIRST BYTE
000015 02                           28+          DC    X'02' TYPE
000016 01                           29+          DC    AL1(1) IDENT
000017 07                           30+          DC    AL1(IHBS2-IHBR2) LENGTH
000018 000000                       31+          DC    AL3(BACK) LOCATION
00001B 01                           32+          DC    AL1(1) FLAG BYTE
00001C 62                           33+IHBS2     DC    X'62' ENTRY END CODE
000000                              34+GO        CSECT  , PROBLEM PROGRAM CSECT
                                    35 *         THE FOLLOWING TESTRAN STATEMENTS
                                    36           DUMP  COMMENT,'BSL DATA'
                                    37                 *,***  IEGM09* - MACRO NUMBER 2 IN GO1
000000                              38+GO1       CSECT  , TESTRAN CSECT
00001C                              39+          CRG   *-1 ORG CVER LAST END ENTRY
00001C                              40+IHBR3     EQU   * NAME THE FIRST BYTE
00001C 1A                           41+          DC    X'1A' TYPE
00001D 02                           42+          DC    AL1(2) IDENT
00001E 0C                           43+          DC    AL1(IHBS3-IHBR3) LENGTH
00001F                              44+IHBY3     EQU   * TO COMPUTE CCMMENT LENGTH
00001F C2E2D340C4C1E3C1             45+          DC    C'BSL DATA' LAST OF COMMENT
000027                              46+IHBZ3     EQU   * TO COMPUTE COMMENT LENGTH
000027 08                           47+          DC    AL1(IHBZ3-IHBY3) FLAG BYTE
000028 62                           48+IHBS3     DC    X'62' ENTRY END CODE
000000                              49+GO        CSECT  , PROBLEM PROGRAM CSECT
                                    50           DUMP  DATA,S,S+24,DATAM=X,NAME=S,DSECT=(S,2)
                                    51                 *,***  IEGM09* - MACRO NUMBER 3 IN GO1
000000                              52+GC1       CSECT  , TESTRAN CSECT
000028                              53+          ORG   *-1 ORG OVER LAST END ENTRY
000028                              54+IHBR4     EQU   * NAME THE FIRST BYTE
000028 06                           55+          DC    X'06' TYPE
```

• Example 7. Use of OS TESTRAN (Part 3 of 17)

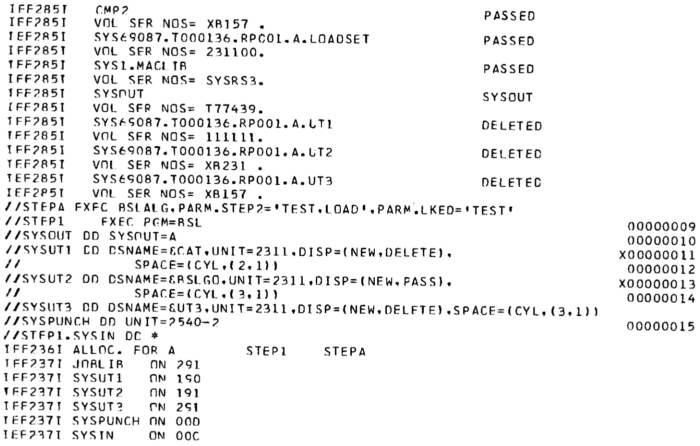
```
                                                                                        PAGE    2

  LCC  ORJECT CODE     ADDR1 ADDR2  STMT    SOURCE STATFMENT                     F15APR68    3/28/69

000029 03                            56+         DC     AL1(3) IDENT
00002A 12                            57+         DC     AL1(IHBS4-IHBR4) LENGTH
00002R 20                            58+         DC     X'20' A FIELD OR BYTE
00002C 5000                          59+         DC     SL2(S+0)
00002E 20                            60+         DC     X'20' A FIELD OR BYTE
00002F 5018                          61+         DC     SL2(S+24+0)
000031 01                            62+         DC     AL1(1) DATAM FO FIELD
000032 01                            63+         DC     AL1(1) LENGTH OF NAME FIELD
000033 E2                            64+         DC     C'S' NAME FIELD
000034 01                            65+         CC     AL1(1) LENGTH OF DSECT NAME
000035 02                            66+         DC     AL1(2) DSECT REPEAT COUNT
000036 5000                          67+         DC     SL2(S) RASE AND DISPLACEMENT
00003R E2                            68+         DC     C'S' DSECT NAME
000039 2C                            69+         DC     AL1(44) FLAG BYTE
00003A 62                            70+IHBS4    DC     X'62' ENTRY END CODE
000000                               71+GO       CSECT  , PROBLEM PROGRAM CSECT
                                     72 *        DUMPS STRUCTURE IN HEX
                                     73          DUMP   DATA,A,A+60,DATAM=CL10,NAME=A
                                     74                 *,***  IEGM09* - MACRO NUMBER 4 IN GO1
0C0000                               75+GO1      CSECT  , TESTRAN CSECT
00003A                               76+         ORG    *-1 ORG CVER LAST END ENTRY
00003A                               77+IHBR5    EQU    * NAME THE FIRST BYTE
00003A 06                            78+         DC     X'06' TYPE
00003B 04                            79+         DC     AL1(4) IDENT
00003C 11                            80+         DC     AL1(IHBS5-IHBR5) LENGTH
00003D 10                            81+         DC     X'10' A FLD OR BYTE
00003E 000000                        82+         DC     AL3(A+0) A FIELD AL BYTES
000041 10                            83+         DC     X'10' A FLD CR BYTE
000042 00003C                        84+         CC     AL3(A+60+0) A FIELD AL BYTES
000045 00                            85+         DC     AL1(0) DATAM FO FIELD
000046 000A                          86+         DC     AL2(10) DATAM L FIELD
0C0048 01                            87+         DC     AL1(1) LENGTH CF NAME FIELD
000049 C1                            88+         DC     C'A' NAME FIELD
00004A 38                            89+         DC     AL1(56) FLAG BYTE
00004R 62                            90+IHBS5    DC     X'62' ENTRY END CODE
0C0000                               91+GC       CSECT  , PROBLEM PROGRAM CSECT
                                     92 *        DUMPS ARRAY A IN CHARACTER
                                     93          DUMP   PANEL,DATAM=X        DUMPS REGISTERS
                                     94                 *,***  IEGM09* - MACRO NUMBER 5 IN GO1
000000                               95+GO1      CSECT  , TESTRAN CSECT
00004E                               96+         CRG    *-1 ORG CVER LAST END ENTRY
00004B                               97+IHBR6    EQU    * NAME THE FIRST BYTE
00004B 16                            98+         DC     X'16' TYPE
00004C 05                            99+         DC     AL1(5) IDENT
00004D 06                           100+         DC     AL1(IHBS6-IHBR6) LENGTH
00004E 01                           101+         DC     AL1(1) DATAM FO FIELD
00004F 20                           102+         DC     AL1(32) FLAG BYTE
000050 00                           103+         DC     AL1(0) FLAG BYTE PART 2
000051 62                           104+IHBS6    DC     X'62' ENTRY END CODE
000000                              105+GO       CSECT  , PROBLEM PROGRAM CSECT
                                    106          GO     BACK          RETURN TO CONTINUE EXECUTION CF DRIVER
                                    107                 *,***  IEGM09* - MACRO NUMBER 6 IN GO1
000000                              108+GC1      CSECT  , TESTRAN CSECT
000051                              109+         ORG    *-1 ORG CVER LAST END ENTRY
000051                              110+IHBR7    EQU    * SET UP NAME FCR LENGTH
```

• Example 7. Use of OS TESTRAN (Part 4 of 17)



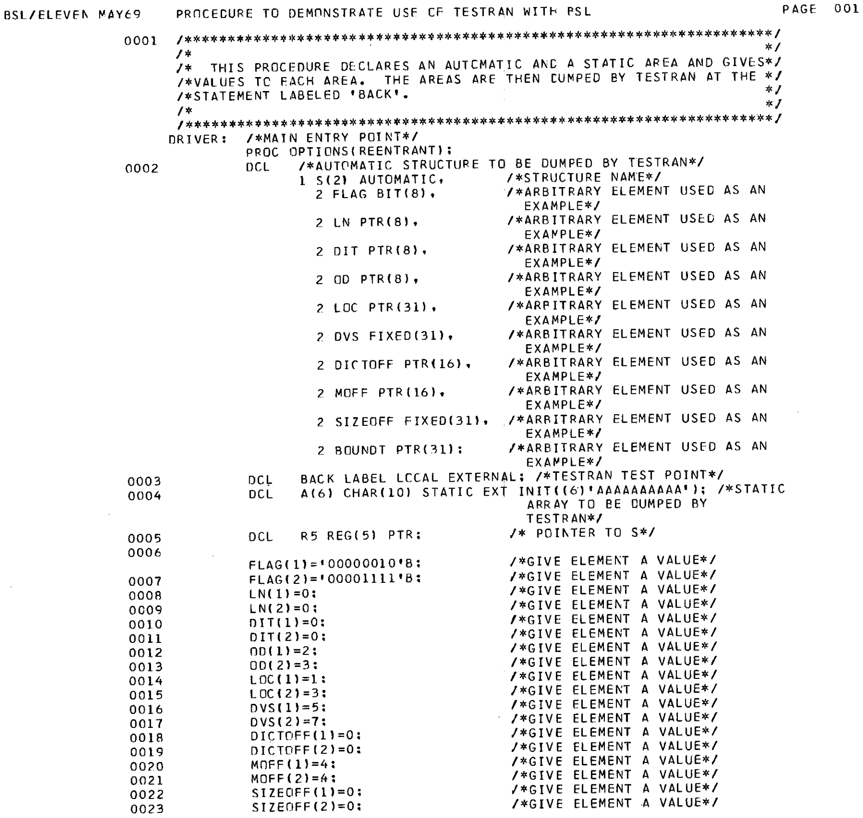
```
                                                                                          PAGE    3

  LCC  OPJECT CODE    ADDR1 ACDR2  STMT    SOURCE STATEMENT                         F15APR68   3/28/69

000051 3F                          111+          DC     X'3E' TYPE
000052 06                          112+          DC     AL1(6) IDENT
000053 03                          113+          DC     AL1(IHBS7-IHBR7) LENGTH
000054 62                          114+IHBS7     DC     X'62' ENTRY END CODE
0C0000                             115+GO        CSECT  , PROBLEM PROGRAM CSECT
000000                             116           DS     0F
000000                             117 S         DSECT
000000                             118 D76       DS     6F'0'      STRUCTURE DESCRIPTION
C00000                             119           END    G01
```

• Example 7. Use of OS TESTRAN (Part 5 of 17)



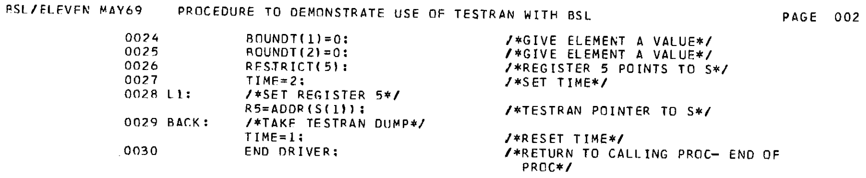
```
IEF285I   CMP2                                               PASSED
IEF285I   VOL SER NOS= XB157 .
IEF285I   SYS69087.T000136.RPC01.A.LOADSET                   PASSED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS1.MACLIB                                        PASSED
IEF285I   VOL SER NOS= SYSRS3.
IEF285I   SYSOUT                                             SYSOUT
IEF285I   VOL SER NOS= T77439.
IEF285I   SYS69087.T000136.RP001.A.UT1                       DELETED
IEF285I   VOL SER NOS= 111111.
IEF285I   SYS69087.T000136.RP001.A.UT2                       DELETED
IEF285I   VOL SER NOS= XB231 .
IEF285I   SYS69087.T000136.RP001.A.UT3                       DELETED
IEF285I   VOL SER NOS= XB157 .
//STEPA EXEC BSLALG,PARM.STEP2='TEST,LOAD',PARM.LKED='TEST'
//STEP1    EXEC PGM=BSL                                                00000009
//SYSOUT DD SYSOUT=A                                                   00000010
//SYSUT1 DD DSNAME=&CAT,UNIT=2311,DISP=(NEW,DELETE),                  X00000011
//             SPACE=(CYL,(2,1))                                       00000012
//SYSUT2 DD DSNAME=&BSLGO,UNIT=2311,DISP=(NEW,PASS),                  X00000013
//             SPACE=(CYL,(3,1))                                       00000014
//SYSUT3 DD DSNAME=&UT3,UNIT=2311,DISP=(NEW,DELETE),SPACE=(CYL,(3,1))
//SYSPUNCH DD UNIT=2540-2                                              00000015
//STEP1.SYSIN DD *
IEF236I ALLOC. FOR A        STEP1    STEPA
IEF237I JOBLIB   ON 291
IEF237I SYSUT1   ON 190
IEF237I SYSUT2   ON 191
IEF237I SYSUT3   ON 291
IEF237I SYSPUNCH ON 00D
IEF237I SYSIN    ON 00C
```

• Example 7. Use of OS TESTRAN (Part 6 of 17)



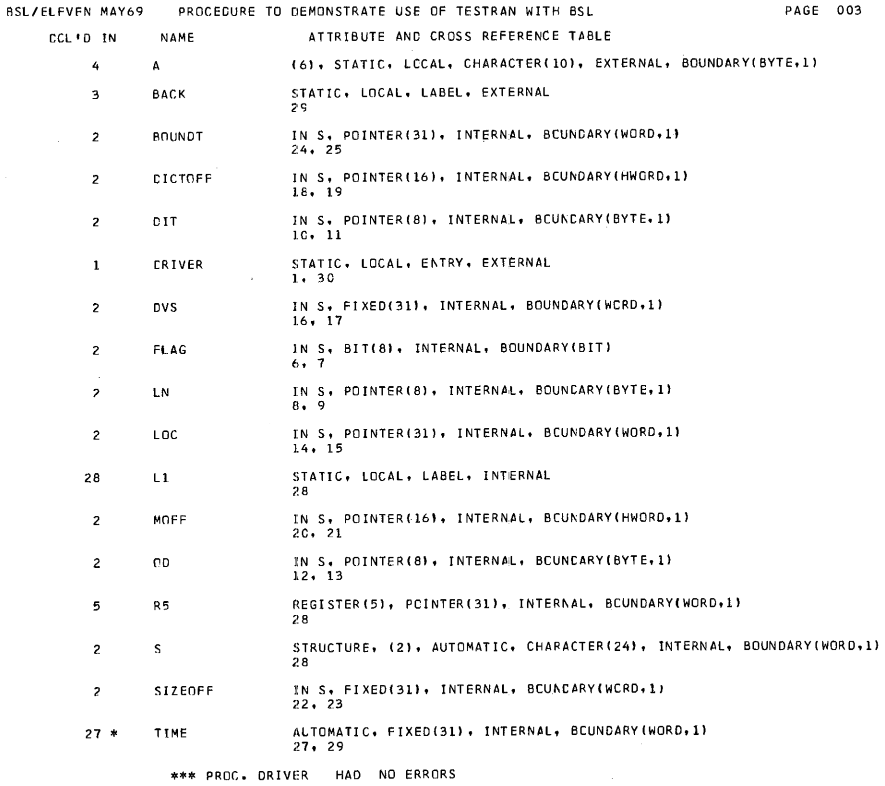
```
BSL/ELEVEN MAY69    PROCEDURE TO DEMONSTRATE USE CF TESTRAN WITH PSL                  PAGE  001

              0001  /*******************************************************************/
                    /*                                                                 */
                    /*  THIS PROCEDURE DECLARES AN AUTCMATIC ANC A STATIC AREA AND GIVES*/
                    /*VALUES TC EACH AREA.  THE AREAS ARE THEN CUMPED BY TESTRAN AT THE */
                    /*STATEMENT LABELED 'BACK'.                                        */
                    /*                                                                 */
                    /*******************************************************************/
                   DRIVER:  /*MAIN ENTRY POINT*/
                            PROC OPTIONS(REENTRANT):
              0002          DCL    /*AUTOMATIC STRUCTURE TO BE DUMPED BY TESTRAN*/
                                   1 S(2) AUTOMATIC,        /*STRUCTURE NAME*/
                                     2 FLAG BIT(8),         /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 LN PTR(8),           /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 DIT PTR(8),          /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 OD PTR(8),           /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 LOC PTR(31),         /*ARPITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 DVS FIXED(31),       /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 DICTOFF PTR(16),     /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 MOFF PTR(16),        /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 SIZEOFF FIXED(31),   /*ARRITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
                                     2 BOUNDT PTR(31):      /*ARBITRARY ELEMENT USED AS AN
                                                              EXAMPLE*/
              0003          DCL    BACK LABEL LCCAL EXTERNAL; /*TESTRAN TEST POINT*/
              0004          DCL    A(6) CHAR(10) STATIC EXT INIT((6)'AAAAAAAAAA'); /*STATIC
                                                              ARRAY TO BE DUMPED BY
                                                              TESTRAN*/
              0005          DCL    R5 REG(5) PTR:           /* POINTER TO S*/
              0006
                            FLAG(1)='00000010'B:            /*GIVE ELEMENT A VALUE*/
              0007          FLAG(2)='00001111'B:            /*GIVE ELEMENT A VALUE*/
              0008          LN(1)=0:                        /*GIVE ELEMENT A VALUE*/
              0009          LN(2)=0;                        /*GIVE ELEMENT A VALUE*/
              0010          DIT(1)=0:                       /*GIVE ELEMENT A VALUE*/
              0011          DIT(2)=0:                       /*GIVE ELEMENT A VALUE*/
              0012          OD(1)=2:                        /*GIVE ELEMENT A VALUE*/
              0013          OD(2)=3:                        /*GIVE ELEMENT A VALUE*/
              0014          LOC(1)=1:                       /*GIVE ELEMENT A VALUE*/
              0015          LOC(2)=3:                       /*GIVE ELEMENT A VALUE*/
              0016          DVS(1)=5:                       /*GIVE ELEMENT A VALUE*/
              0017          DVS(2)=7:                       /*GIVE ELEMENT A VALUE*/
              0018          DICTOFF(1)=0:                   /*GIVE ELEMENT A VALUE*/
              0019          DICTOFF(2)=0:                   /*GIVE ELEMENT A VALUE*/
              0020          MOFF(1)=4:                      /*GIVE ELEMENT A VALUE*/
              0021          MOFF(2)=4:                      /*GIVE ELEMENT A VALUE*/
              0022          SIZEOFF(1)=0:                   /*GIVE ELEMENT A VALUE*/
              0023          SIZEOFF(2)=0:                   /*GIVE ELEMENT A VALUE*/
```

• Example 7. Use of OS TESTRAN (Part 7 of 17)



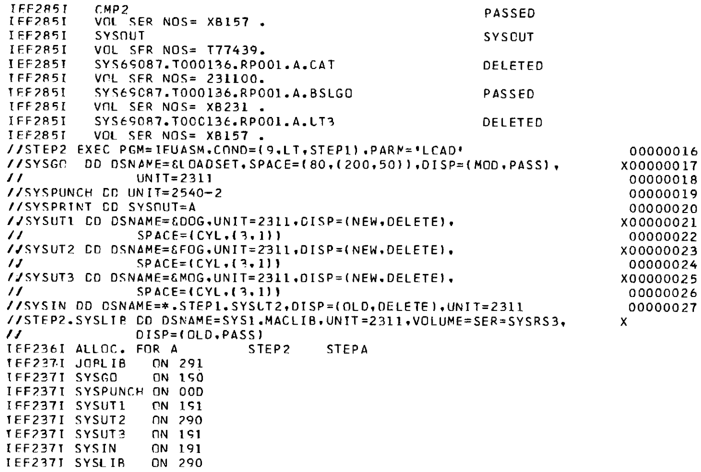
```
BSL/ELEVEN MAY69    PROCEDURE TO DEMONSTRATE USE OF TESTRAN WITH BSL                PAGE  002

            0024          BOUNDT(1)=0:                      /*GIVE ELEMENT A VALUE*/
            0025          BOUNDT(2)=0:                      /*GIVE ELEMENT A VALUE*/
            0026          RESTRICT(5):                      /*REGISTER 5 POINTS TO S*/
            0027          TIME=2;                           /*SET TIME*/
            0028 L1:      /*SET REGISTER 5*/
                          R5=ADDR(S(1));                    /*TESTRAN POINTER TO S*/
            0029 BACK:    /*TAKE TESTRAN DUMP*/
                          TIME=1;                           /*RESET TIME*/
            0030          END DRIVER;                       /*RETURN TO CALLING PROC- END OF
                                                              PROC*/
```

• Example 7. Use of OS TESTRAN (Part 8 of 17)



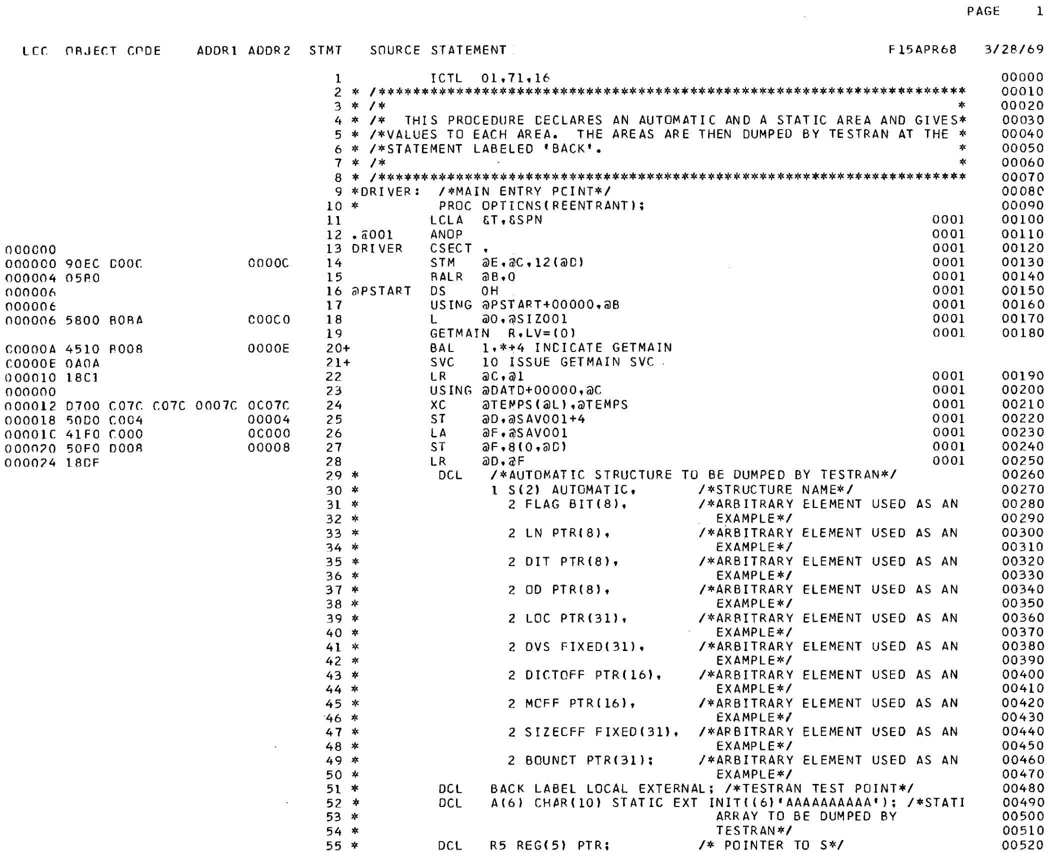
```
BSL/ELFVFN MAY69    PROCEDURE TO DEMONSTRATE USE OF TESTRAN WITH BSL                PAGE  003
     CCL'D IN      NAME               ATTRIBUTE AND CROSS REFERENCE TABLE

          4        A                (6), STATIC, LCCAL, CHARACTER(10), EXTERNAL, BOUNDARY(BYTE,1)

          3        BACK             STATIC, LOCAL, LABEL, EXTERNAL
                                    29

          2        BOUNDT           IN S, POINTER(31), INTERNAL, BCUNDARY(WORD,1)
                                    24, 25

          2        CICTOFF          IN S, POINTER(16), INTERNAL, BCUNDARY(HWORD,1)
                                    18, 19

          2        CIT              IN S, POINTER(8), INTERNAL, BCUNCARY(BYTE,1)
                                    10, 11

          1        CRIVER           STATIC, LOCAL, ENTRY, EXTERNAL
                                    1, 30

          2        DVS              IN S, FIXED(31), INTERNAL, BOUNDARY(WCRD,1)
                                    16, 17

          2        FLAG             IN S, BIT(8), INTERNAL, BOUNDARY(BIT)
                                    6, 7

          ?        LN               IN S, POINTER(8), INTERNAL, BOUNCARY(BYTE,1)
                                    8, 9

          2        LOC              IN S, POINTER(31), INTERNAL, BCUNDARY(WORD,1)
                                    14, 15

         28        L1               STATIC, LOCAL, LABEL, INTERNAL
                                    28

          2        MOFF             IN S, POINTER(16), INTERNAL, BCUNDARY(HWORD,1)
                                    20, 21

          2        OD               IN S, POINTER(8), INTERNAL, BCUNCARY(BYTE,1)
                                    12, 13

          5        R5               REGISTER(5), PCINTER(31), INTERNAL, BCUNDARY(WORD,1)
                                    28

          2        S                STRUCTURE, (2), AUTOMATIC, CHARACTER(24), INTERNAL, BOUNDARY(WORD,1)
                                    28

          ?        SIZEOFF          IN S, FIXED(31), INTERNAL, BCUNCARY(WCRD,1)
                                    22, 23

         27 *      TIME             AUTOMATIC, FIXED(31), INTERNAL, BCUNDARY(WORD,1)
                                    27, 29

              *** PROC. DRIVER   HAD  NO ERRORS
```

• Example 7. Use of OS TESTRAN (Part 9 of 17)



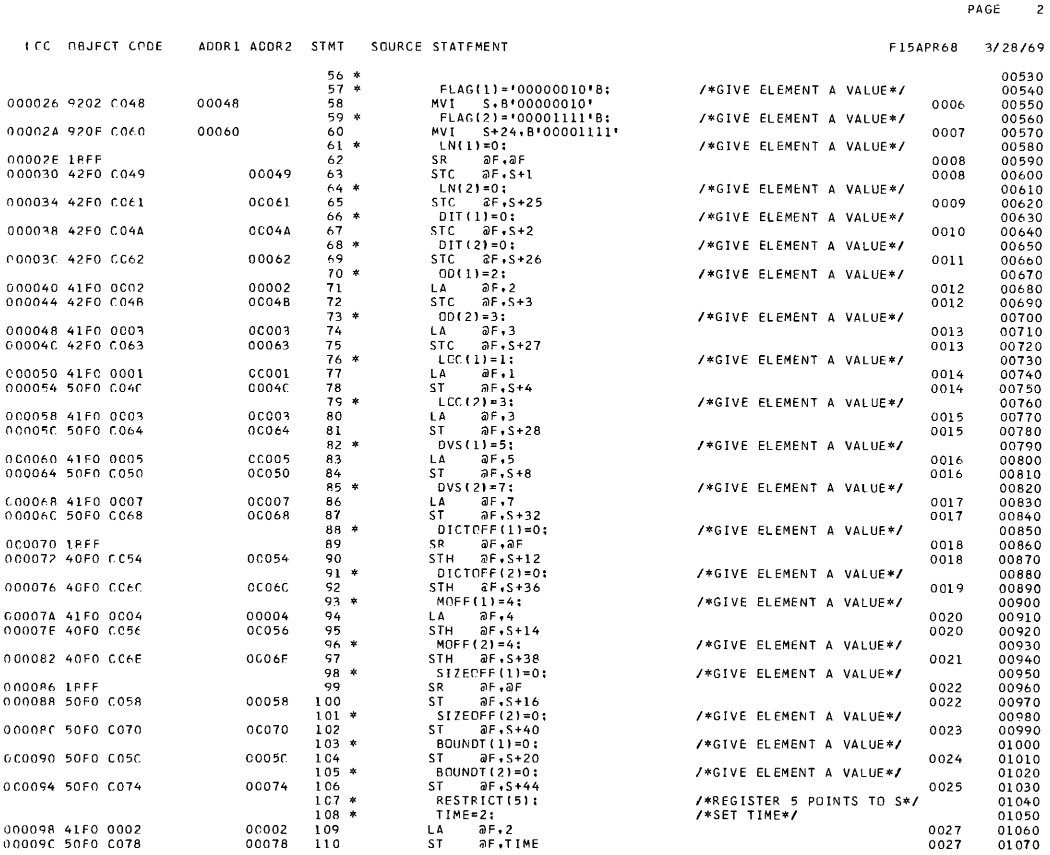
```
IEF285I   CMP2                                              PASSED
IEF285I   VOL SER NOS= XB157 .
IEF285I   SYSOUT                                            SYSOUT
IEF285I   VOL SER NOS= T77439.
IEF285I   SYS69087.T000136.RP001.A.CAT                      DELETED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS69087.T000136.RP001.A.BSLGO                    PASSED
IEF285I   VOL SER NOS= XB231 .
IEF285I   SYS69087.T000136.RP001.A.LT3                      DELETED
IEF285I   VOL SER NOS= XB157 .
//STEP2 EXEC PGM=IEUASM,COND=(9,LT,STEP1),PARM='LCAD'                   00000016
//SYSGO  DD DSNAME=&LOADSET,SPACE=(80,(200,50)),DISP=(MOD,PASS),       X00000017
//              UNIT=2311                                               00000018
//SYSPUNCH CD UNIT=2540-2                                               00000019
//SYSPRINT CD SYSOUT=A                                                  00000020
//SYSUT1 CD DSNAME=&DOG,UNIT=2311,DISP=(NEW,DELETE),                   X00000021
//              SPACE=(CYL,(3,1))                                       00000022
//SYSUT2 CD DSNAME=&FOG,UNIT=2311,DISP=(NEW,DELETE),                   X00000023
//              SPACE=(CYL,(3,1))                                       00000024
//SYSUT3 CD DSNAME=&MOG,UNIT=2311,DISP=(NEW,DELETE),                   X00000025
//              SPACE=(CYL,(3,1))                                       00000026
//SYSIN DD DSNAME=*.STEP1.SYSLT2,DISP=(OLD,DELETE),UNIT=2311            00000027
//STEP2.SYSLIB CD DSNAME=SYS1.MACLIB,UNIT=2311,VOLUME=SER=SYSRS3,      X
//              DISP=(OLD,PASS)
IEF236I ALLOC. FOR A        STEP2    STEPA
IEF237I JOBLIB   ON 291
IEF237I SYSGO    ON 190
IEF237I SYSPUNCH ON 00D
IEF237I SYSUT1   ON 191
IEF237I SYSUT2   ON 290
IEF237I SYSUT3   ON 191
IEF237I SYSIN    ON 191
IEF237I SYSLIB   ON 290
```

• Example 7. Use of OS TESTRAN (Part 10 of 17)



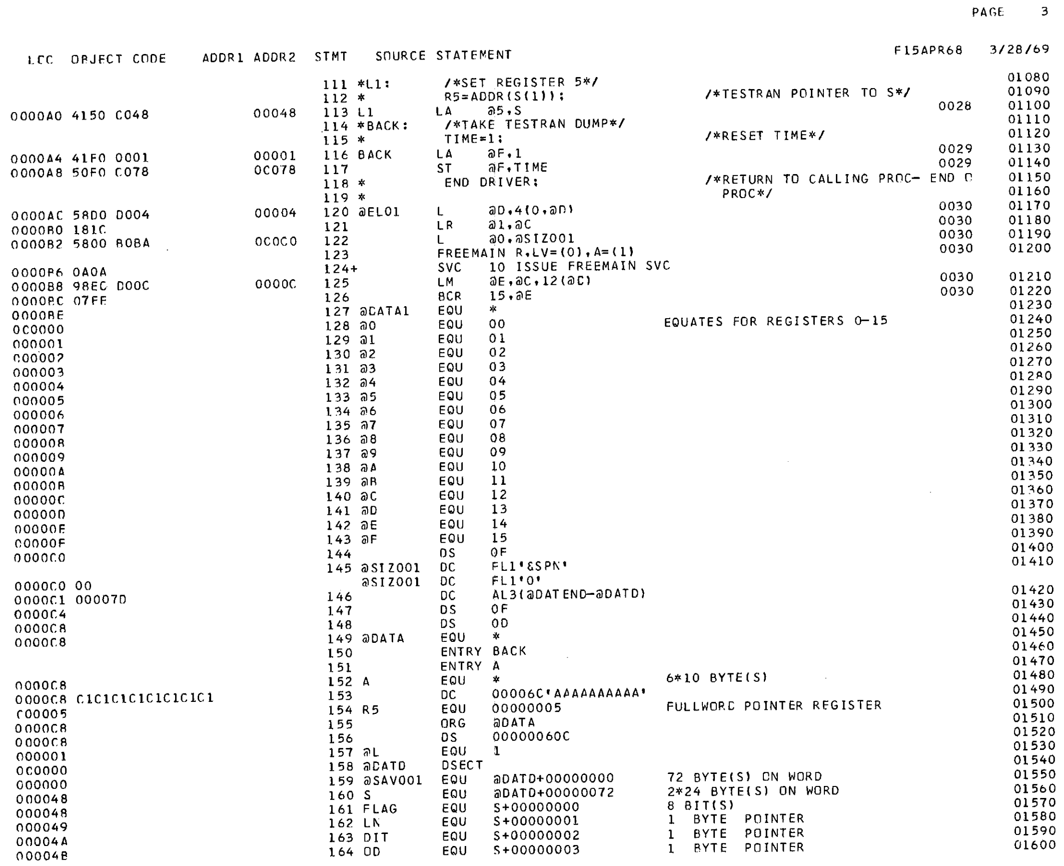
```
                                                                                                             PAGE    1

  LCC  OBJECT CPDE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                          F15APR68   3/28/69

                                     1           ICTL  01,71,16                                                   00000
                                     2   * /********************************************************************   00010
                                     3   * /*                                                                  *   00020
                                     4   * /*  THIS PROCEDURE DECLARES AN AUTOMATIC AND A STATIC AREA AND GIVES*   00030
                                     5   * /*VALUES TO EACH AREA.  THE AREAS ARE THEN DUMPED BY TESTRAN AT THE *   00040
                                     6   * /*STATEMENT LABELED 'BACK'.                                         *   00050
                                     7   * /*                                                                  *   00060
                                     8   * /********************************************************************   00070
                                     9   *DRIVER:  /*MAIN ENTRY PCINT*/                                            0008C
                                    10   *         PROC OPTICNS(REENTRANT);                                        00090
                                    11           LCLA  &T,&SPN                                         0001    00100
                                    12   .@001    ANOP                                                 0001    00110
000000                              13   DRIVER   CSECT ,                                              0001    00120
000000 90EC D00C          0000C     14            STM   @E,@C,12(@D)                                   0001    00130
000004 05B0                         15            BALR  @B,0                                           0001    00140
000006                              16   @PSTART  DS    0H                                             0001    00150
000006                              17            USING @PSTART+00000,@B                               0001    00160
000006 5800 B0BA          C00C0     18            L     @0,@SIZ001                                     0001    00170
                                    19            GETMAIN  R,LV=(0)                                    0001    00180
C0000A 4510 B008          0000E     20+           BAL   1,*+4 INDICATE GETMAIN
C0000E 0A0A                         21+           SVC   10 ISSUE GETMAIN SVC
000010 18C1                         22            LR    @C,@1                                          0001    00190
000000                              23            USING @DATD+00000,@C                                 0001    00200
000012 D700 C07C C07C 0007C 0C07C   24            XC    @TEMPS(@L),@TEMPS                              0001    00210
000018 50D0 C004          00004     25            ST    @D,@SAV001+4                                   0001    00220
00001C 41F0 C000          0C000     26            LA    @F,@SAV001                                     0001    00230
000020 50F0 D008          00008     27            ST    @F,8(0,@D)                                     0001    00240
000024 18DF                         28            LR    @D,@F                                          0001    00250
                                    29   *         DCL   /*AUTOMATIC STRUCTURE TO BE DUMPED BY TESTRAN*/           00260
                                    30   *               1 S(2) AUTOMATIC,      /*STRUCTURE NAME*/                 00270
                                    31   *                 2 FLAG BIT(8),       /*ARBITRARY ELEMENT USED AS AN     00280
                                    32   *                                        EXAMPLE*/                        00290
                                    33   *                 2 LN PTR(8),         /*ARBITRARY ELEMENT USED AS AN     00300
                                    34   *                                        EXAMPLE*/                        00310
                                    35   *                 2 DIT PTR(8),        /*ARBITRARY ELEMENT USED AS AN     00320
                                    36   *                                        EXAMPLE*/                        00330
                                    37   *                 2 OD PTR(8),         /*ARBITRARY ELEMENT USED AS AN     00340
                                    38   *                                        EXAMPLE*/                        00350
                                    39   *                 2 LOC PTR(31),       /*ARBITRARY ELEMENT USED AS AN     00360
                                    40   *                                        EXAMPLE*/                        00370
                                    41   *                 2 DVS FIXED(31),     /*ARBITRARY ELEMENT USED AS AN     00380
                                    42   *                                        EXAMPLE*/                        00390
                                    43   *                 2 DICTOFF PTR(16),   /*ARBITRARY ELEMENT USED AS AN     00400
                                    44   *                                        EXAMPLE*/                        00410
                                    45   *                 2 MCFF PTR(16),      /*ARBITRARY ELEMENT USED AS AN     00420
                                    46   *                                        EXAMPLE*/                        00430
                                    47   *                 2 SIZECFF FIXED(31), /*ARBITRARY ELEMENT USED AS AN     00440
                                    48   *                                        EXAMPLE*/                        00450
                                    49   *                 2 BOUNCT PTR(31);    /*ARBITRARY ELEMENT USED AS AN     00460
                                    50   *                                        EXAMPLE*/                        00470
                                    51   *         DCL   BACK LABEL LOCAL EXTERNAL; /*TESTRAN TEST POINT*/         00480
                                    52   *         DCL   A(6) CHAR(10) STATIC EXT INIT((6)'AAAAAAAAAA'); /*STATI   00490
                                    53   *                                        ARRAY TO BE DUMPED BY            00500
                                    54   *                                        TESTRAN*/                        00510
                                    55   *         DCL   R5 REG(5) PTR;         /* POINTER TO S*/                  00520
```

• Example 7. Use of OS TESTRAN (Part 11 of 17)



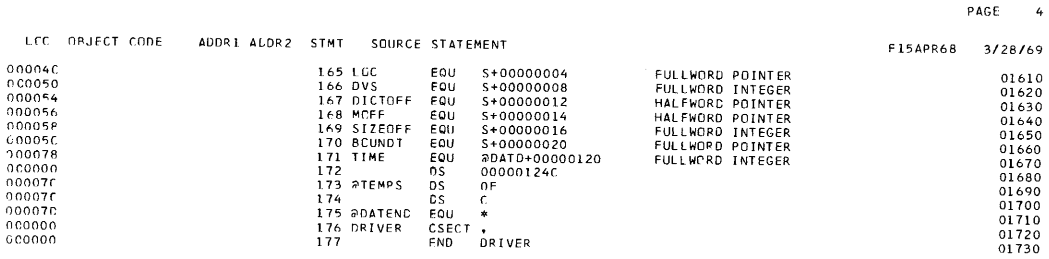
```
                                                                                                      PAGE    2

 LCC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                   F15APR68    3/28/69

                                    56 *                                                                  00530
                                    57 *          FLAG(1)='00000010'B:          /*GIVE ELEMENT A VALUE*/        00540
000026 9202 C048       00048        58           MVI    S,B'00000010'                                  0006   00550
                                    59 *          FLAG(2)='00001111'B:          /*GIVE ELEMENT A VALUE*/        00560
00002A 920F C060       00060        60           MVI    S+24,B'00001111'                               0007   00570
                                    61 *          LN(1)=0:                      /*GIVE ELEMENT A VALUE*/        00580
00002E 1BFF                         62           SR     @F,@F                                          0008   00590
000030 42F0 C049             00049  63           STC    @F,S+1                                         0008   00600
                                    64 *          LN(2)=0:                      /*GIVE ELEMENT A VALUE*/        00610
000034 42F0 C061             0C061  65           STC    @F,S+25                                        0009   00620
                                    66 *          DIT(1)=0:                     /*GIVE ELEMENT A VALUE*/        00630
000038 42F0 C04A             0C04A  67           STC    @F,S+2                                         0010   00640
                                    68 *          DIT(2)=0:                     /*GIVE ELEMENT A VALUE*/        00650
00003C 42F0 CC62             00062  69           STC    @F,S+26                                        0011   00660
                                    70 *          OD(1)=2:                      /*GIVE ELEMENT A VALUE*/        00670
000040 41F0 0C02             00002  71           LA     @F,2                                           0012   00680
000044 42F0 C04B             0C04B  72           STC    @F,S+3                                         0012   00690
                                    73 *          OD(2)=3:                      /*GIVE ELEMENT A VALUE*/        00700
000048 41F0 0003             0C003  74           LA     @F,3                                           0013   00710
00004C 42F0 C063             00063  75           STC    @F,S+27                                        0013   00720
                                    76 *          LCC(1)=1:                     /*GIVE ELEMENT A VALUE*/        00730
000050 41FC 0001             CC001  77           LA     @F,1                                           0014   00740
000054 50F0 C04C             0004C  78           ST     @F,S+4                                         0014   00750
                                    79 *          LCC(2)=3:                     /*GIVE ELEMENT A VALUE*/        00760
000058 41F0 0C03             0C003  80           LA     @F,3                                           0015   00770
00005C 50F0 C064             0C064  81           ST     @F,S+28                                        0015   00780
                                    82 *          DVS(1)=5:                     /*GIVE ELEMENT A VALUE*/        00790
0C0060 41F0 0005             CC005  83           LA     @F,5                                           0016   00800
000064 50F0 C050             0C050  84           ST     @F,S+8                                         0016   00810
                                    85 *          DVS(2)=7:                     /*GIVE ELEMENT A VALUE*/        00820
C00068 41F0 0007             0C007  86           LA     @F,7                                           0017   00830
00006C 50F0 C068             0C068  87           ST     @F,S+32                                        0017   00840
                                    88 *          DICTOFF(1)=0:                 /*GIVE ELEMENT A VALUE*/        00850
0C0070 1BFF                         89           SR     @F,@F                                          0018   00860
000072 40F0 CC54             00054  90           STH    @F,S+12                                        0018   00870
                                    91 *          DICTOFF(2)=0:                 /*GIVE ELEMENT A VALUE*/        00880
000076 40F0 CC6C             0C06C  92           STH    @F,S+36                                        0019   00890
                                    93 *          MOFF(1)=4:                    /*GIVE ELEMENT A VALUE*/        00900
C0007A 41F0 0C04             00004  94           LA     @F,4                                           0020   00910
00007E 40F0 CC56             0C056  95           STH    @F,S+14                                        0020   00920
                                    96 *          MOFF(2)=4:                    /*GIVE ELEMENT A VALUE*/        00930
000082 40F0 CC6E             0C06F  97           STH    @F,S+38                                        0021   00940
                                    98 *          SIZEOFF(1)=0:                 /*GIVE ELEMENT A VALUE*/        00950
000086 1BFF                         99           SR     @F,@F                                          0022   00960
000088 50F0 C058             00058 100           ST     @F,S+16                                        0022   00970
                                   101 *          SIZEOFF(2)=0:                 /*GIVE ELEMENT A VALUE*/        00980
00008C 50F0 C070             0C070 102           ST     @F,S+40                                        0023   00990
                                   103 *          BOUNDT(1)=0:                  /*GIVE ELEMENT A VALUE*/        01000
0C0090 50F0 C05C             0005C 1C4           ST     @F,S+20                                        0024   01010
                                   105 *          BOUNDT(2)=0:                  /*GIVE ELEMENT A VALUE*/        01020
0C0094 50F0 C074             00074 106           ST     @F,S+44                                        0025   01030
                                   1C7 *          RESTRICT(5):                  /*REGISTER 5 POINTS TO S*/      01040
                                   108 *          TIME=2:                       /*SET TIME*/                    01050
000098 41F0 0002             0C002 109           LA     @F,2                                           0027   01060
00009C 50F0 C078             00078 110           ST     @F,TIME                                        0027   01070
```

• Example 7. Use of OS TESTRAN (Part 12 of 17)



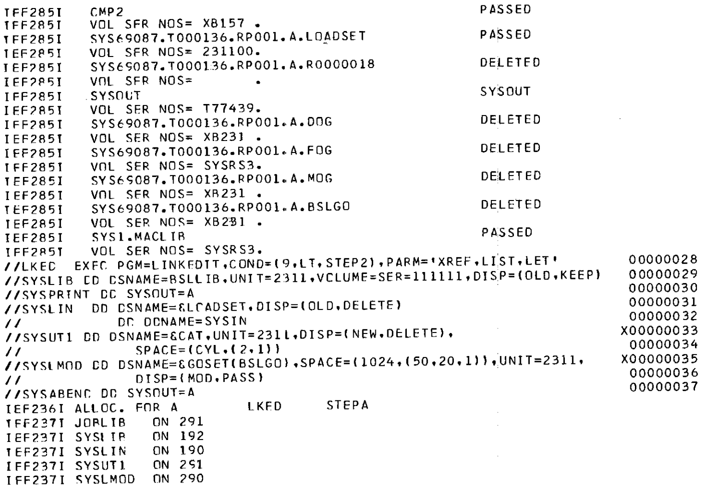
```
                                                                                                    PAGE    3

 LCC  ORJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT                                F15APR68   3/28/69

                                    111 *L1:       /*SET REGISTER 5*/                                       01080
                                    112 *           R5=ADDR(S(1));                  /*TESTRAN POINTER TO S*/   01090
0000A0 4150 C048            00048   113 L1        LA    @5,S                                        0028   01100
                                    114 *BACK:     /*TAKE TESTRAN DUMP*/                                   01110
                                    115 *           TIME=1;                         /*RESET TIME*/          01120
0000A4 41F0 0001            00001   116 BACK      LA    @F,1                                        0029   01130
0000A8 50F0 C078            0C078   117           ST    @F,TIME                                     0029   01140
                                    118 *          END DRIVER;                      /*RETURN TO CALLING PROC- END O   01150
                                    119 *                                             PROC*/               01160
0000AC 58D0 D004            00004   120 @EL01     L     @D,4(0,@D)                                  0030   01170
0000B0 181C                         121           LR    @1,@C                                       0030   01180
0000B2 5800 BOBA            0C0C0   122           L     @0,@SIZ001                                  0030   01190
                                    123           FREEMAIN R,LV=(0),A=(1)                           0030   01200
0000P6 0A0A                         124+          SVC   10 ISSUE FREEMAIN SVC
0000B8 98EC D00C            0000C   125           LM    @E,@C,12(@C)                                0030   01210
0000BC 07FE                         126           BCR   15,@E                                       0030   01220
0000BE                              127 @CATA1    EQU   *                                                  01230
0C0000                              128 @0        EQU   00                   EQUATES FOR REGISTERS 0-15    01240
000001                              129 @1        EQU   01                                                 01250
000002                              130 @2        EQU   02                                                 01260
000003                              131 @3        EQU   03                                                 01270
000004                              132 @4        EQU   04                                                 01280
000005                              133 @5        EQU   05                                                 01290
000006                              134 @6        EQU   06                                                 01300
000007                              135 @7        EQU   07                                                 01310
000008                              136 @8        EQU   08                                                 01320
000009                              137 @9        EQU   09                                                 01330
00000A                              138 @A        EQU   10                                                 01340
00000B                              139 @B        EQU   11                                                 01350
00000C                              140 @C        EQU   12                                                 01360
00000D                              141 @D        EQU   13                                                 01370
00000E                              142 @E        EQU   14                                                 01380
00000F                              143 @F        EQU   15                                                 01390
0000C0                              144           DS    0F                                                 01400
                                    145 @SIZ001   DC    FL1'&SPN'                                          01410
0000C0 00                               @SIZ001   DC    FL1'0'
0000C1 00007D                       146           DC    AL3(@DATEND-@DATD)                                 01420
0000C4                              147           DS    0F                                                 01430
0000C8                              148           DS    0D                                                 01440
0000C8                              149 @DATA     EQU   *                                                  01450
                                    150           ENTRY BACK                                               01460
                                    151           ENTRY A                                                  01470
0000C8                              152 A         EQU   *                    6*10 BYTE(S)                  01480
0000C8 C1C1C1C1C1C1C1C1             153           DC    00006C'AAAAAAAAAA'                                 01490
C00005                              154 R5        EQU   00000005             FULLWORD POINTER REGISTER     01500
0000C8                              155           ORG   @DATA                                              01510
0000C8                              156           DS    00000060C                                          01520
000001                              157 @L        EQU   1                                                  01530
0C0000                              158 @DATD     DSECT                                                    01540
000000                              159 @SAV001   EQU   @DATD+00000000       72 BYTE(S) ON WORD            01550
000048                              160 S         EQU   @DATD+00000072       2*24 BYTE(S) ON WORD          01560
000048                              161 FLAG      EQU   S+00000000           8 BIT(S)                      01570
000049                              162 LN        EQU   S+00000001           1  BYTE  POINTER              01580
00004A                              163 DIT       EQU   S+00000002           1  BYTE  POINTER              01590
00004B                              164 OD        EQU   S+00000003           1  BYTE  POINTER              01600
```

• Example 7. Use of OS TESTRAN (Part 13 of 17)



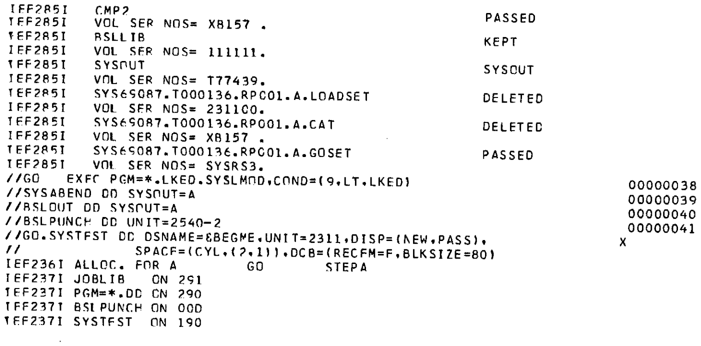
```
                                                                                                    PAGE    4

  LCC  ORJECT CODE     ADDR1 ALDR2  STMT   SOURCE STATEMENT                                F15APR68   3/28/69

000040                              165 LCC       EQU   S+00000004          FULLWORD POINTER              01610
0C0050                              166 DVS       EQU   S+00000008          FULLWORD INTEGER              01620
000054                              167 DICTOFF   EQU   S+00000012          HALFWORD POINTER              01630
000056                              168 MCFF      EQU   S+00000014          HALFWORD POINTER              01640
00005B                              169 SIZEOFF   EQU   S+00000016          FULLWORD INTEGER              01650
C0005C                              170 BCUNDT    EQU   S+00000020          FULLWORD POINTER              01660
000078                              171 TIME      EQU   @DATD+00000120      FULLWORD INTEGER              01670
0C0000                              172           DS    00000124C                                         01680
00007C                              173 @TEMPS    DS    0F                                                01690
00007C                              174           DS    C                                                 01700
00007D                              175 @DATEND   EQU   *                                                 01710
0C0000                              176 DRIVER    CSECT ,                                                 01720
0C0000                              177           END   DRIVER                                            01730
```

• Example 7. Use of OS TESTRAN (Part 14 of 17)



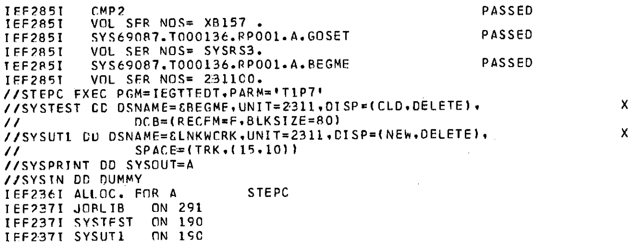
```
TFF285I   CMP2                                              PASSED
IFF285I   VOL SFR NOS= XB157 .
TFF285I   SYS69087.T000136.RP001.A.LOADSET                  PASSED
IEF285I   VOL SFR NOS= 231100.
IEF285I   SYS69087.T000136.RP001.A.R0000018                 DELETED
IEF285I   VOL SFR NOS=        .
IFF285I   SYSOUT                                            SYSOUT
IEF285I   VOL SER NOS= T77439.
IFF285I   SYS69087.T000136.RP001.A.DOG                      DELETED
IEF285I   VOL SER NOS= XB231 .
IFF285I   SYS69087.T000136.RP001.A.FOG                      DELETED
IFF285I   VOL SER NOS= SYSRS3.
TEF285I   SYS69087.T000136.RP001.A.MOG                      DELETED
IEF285I   VOL SER NOS= XB231 .
TEF285I   SYS69087.T000136.RP001.A.BSLGO                    DELETED
IEF285I   VOL SER NOS= XB231 .
IEF285I   SYS1.MACLIB                                       PASSED
IFF285I   VOL SER NOS= SYSRS3.
//LKED  EXEC PGM=LINKEDIT,COND=(9,LT,STEP2),PARM='XREF,LIST,LET'          00000028
//SYSLIB DD DSNAME=BSLLIB,UNIT=2311,VCLUME=SER=111111,DISP=(OLD,KEEP)     00000029
//SYSPRINT DD SYSOUT=A                                                     00000030
//SYSLIN  DD DSNAME=&LOADSET,DISP=(OLD,DELETE)                             00000031
//           DD DDNAME=SYSIN                                               00000032
//SYSUT1 DD DSNAME=&CAT,UNIT=2311,DISP=(NEW,DELETE),                      X00000033
//              SPACE=(CYL,(2,1))                                          00000034
//SYSLMOD DD DSNAME=&GOSET(BSLGO),SPACE=(1024,(50,20,1)),UNIT=2311,       X00000035
//              DISP=(MOD,PASS)                                            00000036
//SYSABEND DD SYSOUT=A                                                     00000037
IEF236I ALLOC. FOR A        LKED     STEPA
IFF237I JOBLIB    ON 291
IEF237I SYSLIB    ON 192
TEF237I SYSLIN    ON 190
IFF237I SYSUT1    ON 291
IFF237I SYSLMOD   ON 290
```

• Example 7. Use of OS TESTRAN (Part 15 of 17)



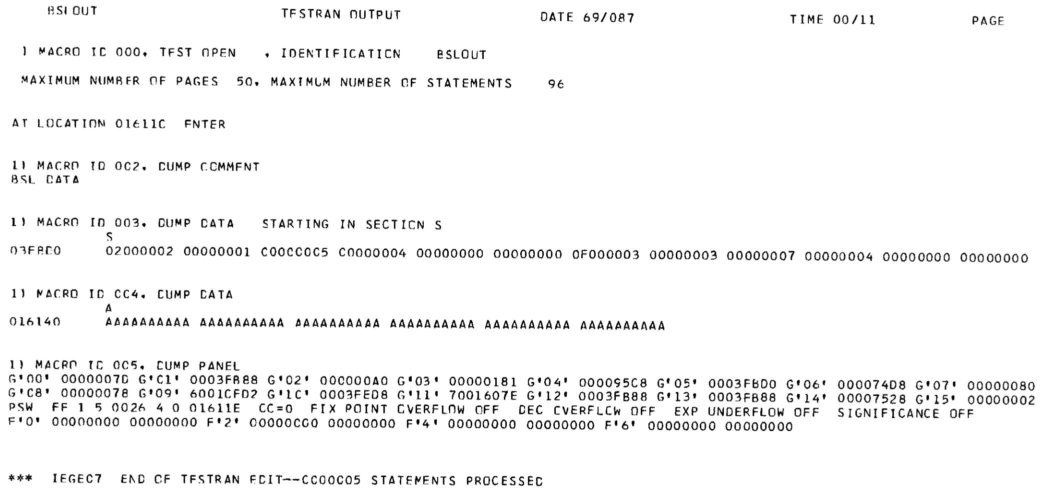
```
IEF285I   CMP2                                          PASSED
IEF285I   VOL SER NOS= XB157 .
IEF285I   BSLLIB                                        KEPT
IEF285I   VOL SER NOS= 111111.
IEF285I   SYSOUT                                        SYSOUT
IEF285I   VOL SER NOS= T77439.
IEF285I   SYS69087.T000136.RPC01.A.LOADSET              DELETED
IEF285I   VOL SER NOS= 231100.
IEF285I   SYS69087.T000136.RP001.A.CAT                  DELETED
IEF285I   VOL SER NOS= XB157 .
IEF285I   SYS69087.T000136.RPC01.A.GOSET                PASSED
IEF285I   VOL SER NOS= SYSRS3.
//GO   EXEC PGM=*.LKED.SYSLMOD,COND=(9,LT,LKED)                         00000038
//SYSABEND DD SYSOUT=A                                                  00000039
//BSLOUT DD SYSOUT=A                                                    00000040
//BSLPUNCH DD UNIT=2540-2                                               00000041
//GO.SYSTEST DD DSNAME=&BEGME,UNIT=2311,DISP=(NEW,PASS),               X
//             SPACE=(CYL,(2,1)),DCB=(RECFM=F,BLKSIZE=80)
IEF236I ALLOC. FOR A        GO       STEPA
IEF237I JOBLIB   ON 291
IEF237I PGM=*.DD ON 290
IEF237I BSLPUNCH ON 00D
IEF237I SYSTEST  ON 190
```

• Example 7. Use of OS TESTRAN (Part 16 of 17)



```
IEF285I   CMP2                                           PASSED
IEF285I   VOL SER NOS= XB157 .
IEF285I   SYS69087.T000136.RP001.A.GOSET                 PASSED
IEF285I   VOL SER NOS= SYSRS3.
IEF285I   SYS69087.T000136.RP001.A.BEGME                 PASSED
IEF285I   VOL SER NOS= 231100.
//STEPC FXEC PGM=IEGTTEDT,PARM='T1P7'
//SYSTEST CC DSNAME=&BEGME,UNIT=2311,DISP=(CLD,DELETE),           X
//             DCB=(RECFM=F,BLKSIZE=80)
//SYSUT1 DD DSNAME=&LNKWCRK,UNIT=2311,DISP=(NEW,DELETE),          X
//             SPACE=(TRK,(15,10))
//SYSPRINT DD SYSOUT=A
//SYSIN DD DUMMY
IEF236I ALLOC. FOR A        STEPC
IEF237I JOBLIB   ON 291
IEF237I SYSTEST  ON 190
IEF237I SYSUT1   ON 19C
```

• Example 7. Use of OS TESTRAN (Part 17 of 17)

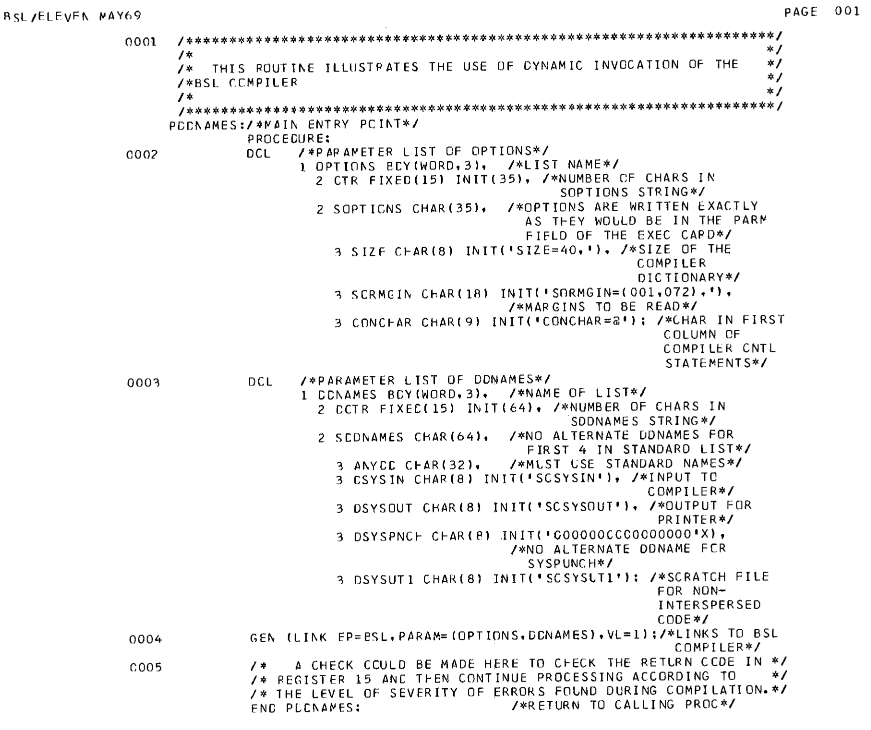


```
    BSI OUT                      TFSTRAN OUTPUT                 DATE 69/087                 TIME 00/11          PAGE    1

 1 MACRO IC 000. TEST OPEN   . IDENTIFICATICN    BSLOUT

 MAXIMUM NUMBER OF PAGES  50. MAXIMUM NUMBER OF STATEMENTS    96


AT LOCATION 01611C  FNTER


1) MACRO ID 0C2. CUMP CCMMENT
BSL CATA


1) MACRO ID 003. CUMP CATA   STARTING IN SECTICN S
           S
03FBC0     02000002 00000001 C00CC0C5 C0000004 00000000 00000000 0F000003 00000003 00000007 00000004 00000000 00000000


1) MACRO IC CC4. CUMP CATA
           A
016140     AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA AAAAAAAAAA


1) MACRO IC 0C5. CUMP PANEL
G'00' 0000007D G'C1' 0003FB88 G'02' 000000A0 G'03' 00000181 G'04' 000095C8 G'05' 0003FBD0 G'06' 000074D8 G'07' 00000080
G'C8' 00000078 G'09' 6001CFD2 G'1C' 0003FED8 G'11' 7001607E G'12' 0003FB88 G'13' 0003FB88 G'14' 00007528 G'15' 00000002
PSW  FF 1 5 0026 4 0 01611E  CC=0  FIX POINT CVERFLOW OFF  DEC CVERFLCW OFF  EXP UNDERFLOW OFF  SIGNIFICANCE OFF
F'0' 00000000 00000000 F'2' 00000CG0 00000000 F'4' 00000000 00000000 F'6' 00000000 00000000


***  IEGEC7  END CF TFSTRAN ECIT--CC00C05 STATEMENTS PROCESSEC
```

Appendix D: Example of Dynamic Invocation of Compiler

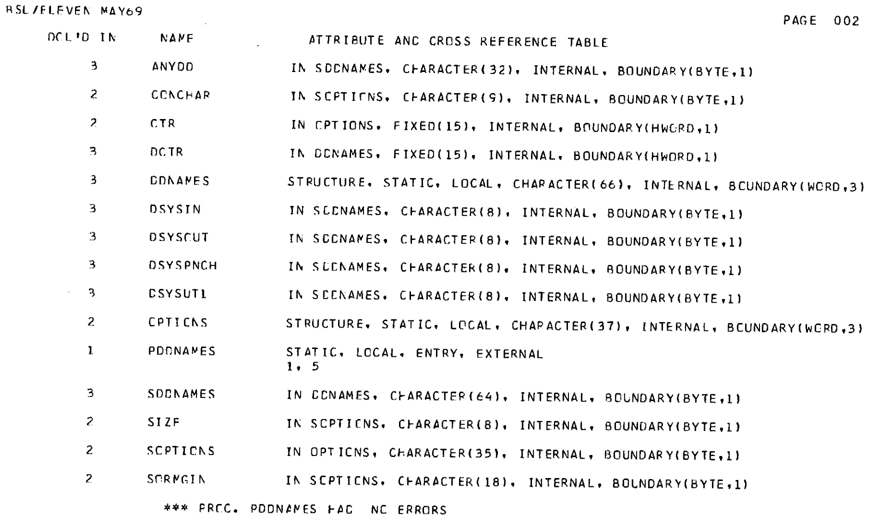
Example 8 is an example of how the BSL compiler can be invoked during execution of a problem program.

• Example 8. Use of Dynamic Invocation of Compiler (Part 1 of 10)



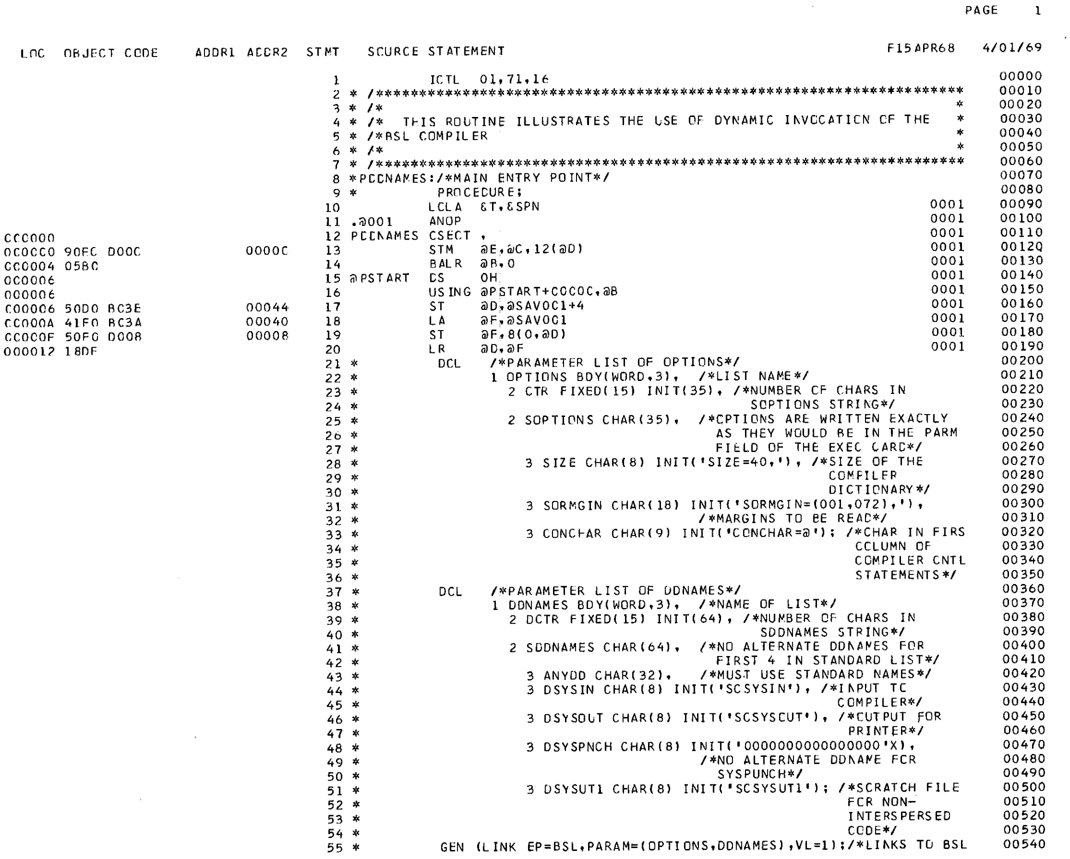
```
BSL/ELEVEN MAY69                                                                    PAGE  001
         0001  /*********************************************************************/
               /*                                                                   */
               /*  THIS ROUTINE ILLUSTRATES THE USE OF DYNAMIC INVOCATION OF THE    */
               /*BSL COMPILER                                                       */
               /*                                                                   */
               /*********************************************************************/
              PCDNAMES:/*MAIN ENTRY POINT*/
                         PROCEDURE;
         0002            DCL   /*PARAMETER LIST OF OPTIONS*/
                               1 OPTIONS BDY(WORD,3),   /*LIST NAME*/
                                 2 CTR FIXED(15) INIT(35), /*NUMBER OF CHARS IN
                                                            SOPTIONS STRING*/
                                 2 SOPTIONS CHAR(35),   /*OPTIONS ARE WRITTEN EXACTLY
                                                         AS THEY WOULD BE IN THE PARM
                                                         FIELD OF THE EXEC CARD*/
                                   3 SIZE CHAR(8) INIT('SIZE=40,'), /*SIZE OF THE
                                                                     COMPILER
                                                                     DICTIONARY*/
                                   3 SORMGIN CHAR(18) INIT('SORMGIN=(001,072),'),
                                                      /*MARGINS TO BE READ*/
                                   3 CONCHAR CHAR(9) INIT('CONCHAR=@'); /*CHAR IN FIRST
                                                                         COLUMN OF
                                                                         COMPILER CNTL
                                                                         STATEMENTS*/
         0003            DCL   /*PARAMETER LIST OF DDNAMES*/
                               1 DDNAMES BDY(WORD,3),   /*NAME OF LIST*/
                                 2 DCTR FIXED(15) INIT(64), /*NUMBER OF CHARS IN
                                                             SDDNAMES STRING*/
                                 2 SDDNAMES CHAR(64),   /*NO ALTERNATE DDNAMES FOR
                                                         FIRST 4 IN STANDARD LIST*/
                                   3 ANYDD CHAR(32),    /*MUST USE STANDARD NAMES*/
                                   3 DSYSIN CHAR(8) INIT('SCSYSIN'), /*INPUT TO
                                                                      COMPILER*/
                                   3 DSYSOUT CHAR(8) INIT('SCSYSOUT'), /*OUTPUT FOR
                                                                        PRINTER*/
                                   3 DSYSPNCH CHAR(8) INIT('0000000000000000'X),
                                                      /*NO ALTERNATE DDNAME FOR
                                                        SYSPUNCH*/
                                   3 DSYSUT1 CHAR(8) INIT('SCSYSUT1'); /*SCRATCH FILE
                                                                        FOR NON-
                                                                        INTERSPERSED
                                                                        CODE*/
         0004            GEN (LINK EP=BSL,PARAM=(OPTIONS,DDNAMES),VL=1);/*LINKS TO BSL
                                                                          COMPILER*/
         0005            /*   A CHECK COULD BE MADE HERE TO CHECK THE RETURN CODE IN */
                         /* REGISTER 15 AND THEN CONTINUE PROCESSING ACCORDING TO    */
                         /* THE LEVEL OF SEVERITY OF ERRORS FOUND DURING COMPILATION.*/
                         END PCDNAMES;                 /*RETURN TO CALLING PROC*/
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 2 of 10)



```
BSL/ELEVEN MAY69
                                                                        PAGE  002
     DCL'D IN      NAME               ATTRIBUTE AND CROSS REFERENCE TABLE

          3       ANYDD           IN SDDNAMES, CHARACTER(32), INTERNAL, BOUNDARY(BYTE,1)

          2       CONCHAR         IN SOPTIONS, CHARACTER(9), INTERNAL, BOUNDARY(BYTE,1)

          2       CTR             IN OPTIONS, FIXED(15), INTERNAL, BOUNDARY(HWORD,1)

          3       DCTR            IN DDNAMES, FIXED(15), INTERNAL, BOUNDARY(HWORD,1)

          3       DDNAMES         STRUCTURE, STATIC, LOCAL, CHARACTER(66), INTERNAL, BOUNDARY(WORD,3)

          3       DSYSIN          IN SDDNAMES, CHARACTER(8), INTERNAL, BOUNDARY(BYTE,1)

          3       DSYSOUT         IN SDDNAMES, CHARACTER(8), INTERNAL, BOUNDARY(BYTE,1)

          3       DSYSPNCH        IN SDDNAMES, CHARACTER(8), INTERNAL, BOUNDARY(BYTE,1)

          3       DSYSUT1         IN SDDNAMES, CHARACTER(8), INTERNAL, BOUNDARY(BYTE,1)

          2       OPTIONS         STRUCTURE, STATIC, LOCAL, CHARACTER(37), INTERNAL, BOUNDARY(WORD,3)

          1       PDDNAMES        STATIC, LOCAL, ENTRY, EXTERNAL
                                  1, 5

          3       SDDNAMES        IN DDNAMES, CHARACTER(64), INTERNAL, BOUNDARY(BYTE,1)

          2       SIZE            IN SOPTIONS, CHARACTER(8), INTERNAL, BOUNDARY(BYTE,1)

          2       SOPTIONS        IN OPTIONS, CHARACTER(35), INTERNAL, BOUNDARY(BYTE,1)

          2       SORMGIN         IN SOPTIONS, CHARACTER(18), INTERNAL, BOUNDARY(BYTE,1)

                 *** PROC. PDDNAMES HAD  NO ERRORS
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 3 of 10)

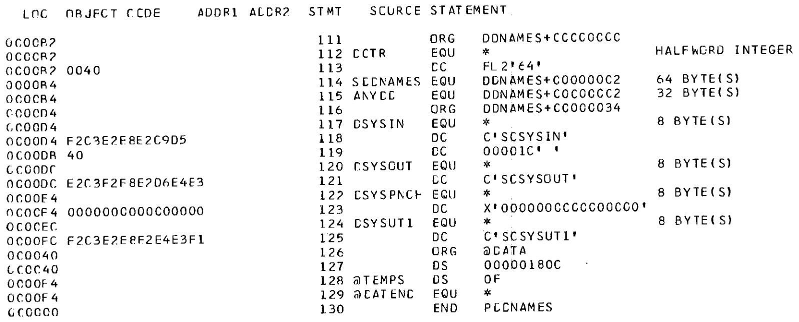


```
                                                                                                     PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SCURCE STATEMENT                                   F15APR68   4/01/69

                                    1             ICTL  01,71,16                                        00000
                                    2 * /*********************************************************************  00010
                                    3 * /*                                                             *  00020
                                    4 * /*  THIS ROUTINE ILLUSTRATES THE USE OF DYNAMIC INVCCATICN CF THE   *  00030
                                    5 * /*BSL COMPILER                                                   *  00040
                                    6 * /*                                                             *  00050
                                    7 * /*********************************************************************  00060
                                    8 *PCCNAMES:/*MAIN ENTRY POINT*/                                         00070
                                    9 *             PROCEDURE;                                              00080
                                   10             LCLA  &T,&SPN                                  0001   00090
                                   11 .@001       ANOP                                           0001   00100
CC0000                             12 PCCNAMES  CSECT ,                                         0001   00110
0C0CC0 90EC D00C          0000C    13             STM   @E,@C,12(@D)                             0001   0012Q
CC0004 05BC                        14             BALR  @B,0                                     0001   00130
0C0006                             15 @PSTART    CS    OH                                       0001   00140
000006                             16             USING @PSTART+COCOC,@B                         0001   00150
C00006 50D0 BC3E          00044    17             ST    @D,@SAVOC1+4                             0001   00160
CC000A 41F0 BC3A          00040    18             LA    @F,@SAVOC1                               0001   00170
CC0C0F 50F0 D008          00008    19             ST    @F,8(0,@D)                               0001   00180
000012 18DF                        20             LR    @D,@F                                    0001   00190
                                   21 *            DCL    /*PARAMETER LIST OF OPTIONS*/                     00200
                                   22 *                   1 OPTIONS BDY(WORD,3),  /*LIST NAME*/              00210
                                   23 *                     2 CTR FIXED(15) INIT(35), /*NUMBER CF CHARS IN   00220
                                   24 *                                            SCPTIONS STRING*/         00230
                                   25 *                     2 SOPTIONS CHAR(35),  /*CPTIONS ARE WRITTEN EXACTLY  00240
                                   26 *                                         AS THEY WOULD BE IN THE PARM  00250
                                   27 *                                         FIELD OF THE EXEC CARD*/    00260
                                   28 *                       3 SIZE CHAR(8) INIT('SIZE=40,'), /*SIZE OF THE  00270
                                   29 *                                                   COMPILER         00280
                                   30 *                                                   DICTIONARY*/     00290
                                   31 *                       3 SORMGIN CHAR(18) INIT('SORMGIN=(001,072),'),  00300
                                   32 *                                       /*MARGINS TO BE READ*/        00310
                                   33 *                       3 CONCHAR CHAR(9) INIT('CONCHAR=@'); /*CHAR IN FIRS  00320
                                   34 *                                                     CCLUMN OF      00330
                                   35 *                                                     COMPILER CNTL  00340
                                   36 *                                                     STATEMENTS*/   00350
                                   37 *            DCL    /*PARAMETER LIST OF DDNAMES*/                     00360
                                   38 *                   1 DDNAMES BDY(WORD,3),  /*NAME OF LIST*/          00370
                                   39 *                     2 DCTR FIXED(15) INIT(64), /*NUMBER OF CHARS IN  00380
                                   40 *                                             SDDNAMES STRING*/       00390
                                   41 *                     2 SDDNAMES CHAR(64),  /*NO ALTERNATE DDNAMES FOR  00400
                                   42 *                                         FIRST 4 IN STANDARD LIST*/  00410
                                   43 *                       3 ANYDD CHAR(32),   /*MUST USE STANDARD NAMES*/  00420
                                   44 *                       3 DSYSIN CHAR(8) INIT('SCSYSIN'), /*INPUT TC    00430
                                   45 *                                                   COMPILER*/       00440
                                   46 *                       3 DSYSOUT CHAR(8) INIT('SCSYSCUT'), /*CUTPUT FOR  00450
                                   47 *                                                    PRINTER*/       00460
                                   48 *                       3 DSYSPNCH CHAR(8) INIT('0000000000000000'X),  00470
                                   49 *                                       /*NO ALTERNATE DDNAME FCR      00480
                                   50 *                                         SYSPUNCH*/                  00490
                                   51 *                       3 DSYSUT1 CHAR(8) INIT('SCSYSUT1'); /*SCRATCH FILE  00500
                                   52 *                                                    FCR NON-        00510
                                   53 *                                                    INTERSPERSED    00520
                                   54 *                                                    CODE*/          00530
                                   55 *            GEN (LINK EP=BSL,PARAM=(OPTIONS,DDNAMES),VL=1);/*LINKS TO BSL  00540
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 4 of 10)

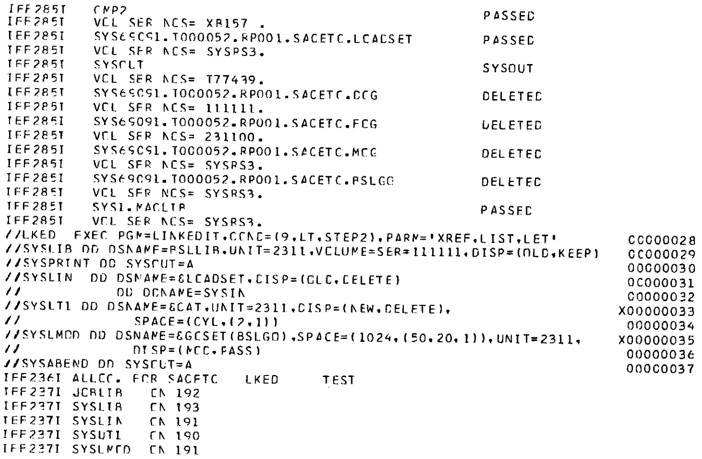
```
                                                                                                        PAGE    2

  LCC  OBJECT CCDE     ADDR1 ALCR2  STMT    SCURCE STATEMENT                                    F15APR68   4/01/69

                                     56 *                                                      COMPILER*/      00550
                                     57               LINK  EP=BSL,PARAM=(OPTIONS,DDNAMES),VL=1                 00560
0C0014                               58+              CNOP  0,4
0CCC14 4510 B01A             00020   59+              BAL   1,IHB0002A LOAD LIST ADDR IN REG1
0C0018                               60+IHB0002       EQU   *
CCOC18 0000CC8A                      61+              DC    A(OPTIONS) PROB.PROG.PARAMETER
0C001C 80                            62+              DC    B'1000000C' SET VL SWITCH BIT
0C001D 0000B2                        63+              CC    AL3(CDNAMES) PRCB. PROG. PARAMETER
0C0020                               64+IHB0002A      EQU   *
0C0020                               65+              CNOP  0,4
CC0020 45F0 BC2E             00034   66+              BAL   15,*+20 LOAD SUP.PARAMLIST ADR
000024 0000002C                      67+              CC    A(*+8) ADDR OF EP PARAMETER
0C0C28 0CCC0C00                      68+              DC    A(C) OCB ADDRESS PARAMETER        LCOA
0C002C C2E2D340404C4040              69+              DC    CL8'BSL' EP PARAMETER
CC0034 0A06                          70+              SVC   6 ISSUE LINK SVC
000036                               71               DS    0H                                                00570
                                     72 *              /*   A CHECK CGULD BE MADE HERE TO CHECK THE RETURN CCDE IN *  00580
                                     73 *              /* REGISTER 15 AND THEN CONTINUE PROCESSING ACCCRDING TC    *  00590
                                     74 *              /* THE LEVEL OF SEVERITY CF ERRCRS FCUND DURING CCMPILATION.*  00600
                                     75 *              END PDDNAMES:                   /*RETURN TO CALLING PRCC*/      00610
C0003E 58DC D004             00004   76 @EL01         L     @D,4(0,@C)                                  0005  00620
0C003A 98EC D00C             0000C   77               LM    @E,@C,12(@D)                                0005  00630
0C003F 07FE                          78               BCR   15,@E                                       0005  00640
0C0040                               79 @CATA1        EQU   *                                                 00650
0C0000                               80 @0            ECU   00                  EQUATES FOR REGISTERS 0-15    00660
CCOC01                               81 @1            EQU   01                                                00670
CC00C2                               82 @2            FCU   02                                                00680
0C0CC3                               83 @3            EQU   03                                                00690
C000C4                               84 @4            EQU   04                                                00700
0C00C5                               85 @5            EQU   05                                                00710
C000C6                               86 @6            EQU   06                                                00720
CC0CC7                               87 @7            EQU   07                                                00730
CCC0C8                               88 @8            EQU   08                                                00740
CC0CC9                               89 @9            EQU   09                                                00750
0C000A                               90 @A            EQU   10                                                00760
0C000B                               91 @E            FQU   11                                                00770
CC000C                               92 @C            EQU   12                                                00780
CC0C0D                               93 @C            EQU   13                                                00790
0C000E                               94 @E            ECU   14                                                00800
CC0C0F                               95 @F            FOU   15                                                00810
0C0040                               96               DS    0C                                                00820
0C0040                               97 @CATA         EQU   *                                                 00830
CC0C40                               98 @SAV001       EQU   @DATA+00CCCCCC      72 BYTE(S) CN WCRD            00840
CC008A                               99 CPTICNS       EQU   @DATA+COC0C074      37 BYTE(S) CN WORD+2          00850
0C008A                              100               ORG   OPTIONS+CCCCCCCC                                  00860
0C0C8A                              101 CTR           EQU   *                   HALFWORD INTEGER              00870
0C0C8A 0023                         102               CC    FL2'35'                                           00880
0C0C8C                              103 SCPTICNS      ECU   OPTIONS+CC000CC2    35 BYTE(S)                    00890
0C008C                              104 SIZE          EQU   *                   8 BYTE(S)                     00900
0C008C E2C9E9C57EF4F06B             105               DC    C'SIZE=4C,'                                       00910
0CC094                              106 SCRMGIN       EQU   *                   18 BYTE(S)                    00920
CCC094 E2D6D9D4C7C9D57E             107               CC    C'SORMGIN=(C01,072),'                             00930
0C0CA6                              108 CCNCHAR       EQU   *                   9 BYTE(S)                     00940
0000A6 C3D6D5C3C8C1D97F             109               DC    C'CONCHAR=@'                                      00950
0C0CB2                              110 DCNAMES       EQU   @CATA+C000C114      66 BYTE(S) CN WCRD+2          00960
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 5 of 10)



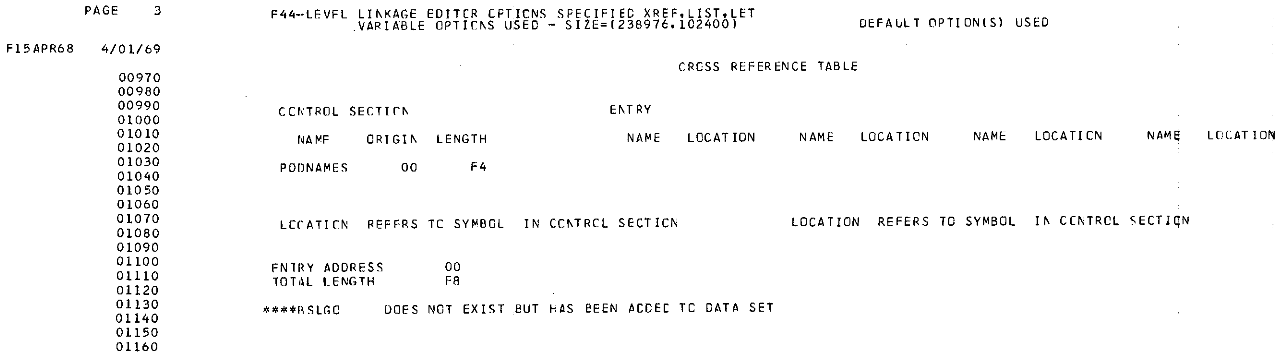
```
  LOC  OBJFCT CCDE     ADDR1 ALCR2  STMT   SCURCE STATEMENT

0C00B2                               111               ORG    DDNAMES+CCCCOCCC
0CCCB2                               112 CCTR          EQU    *                      HALFWORD INTEGER
0C00B2 0040                          113               CC     FL2'64'
0000B4                               114 SCCNAMES      EQU    DDNAMES+C00000C2       64 BYTE(S)
0C0CB4                               115 ANYCC         EQU    DDNAMES+COCOCCC2       32 BYTE(S)
C00CD4                               116               ORG    DDNAMES+CC000034
CC00D4                               117 CSYSIN        EQU    *                      8 BYTE(S)
0C00D4 F2C3E2E8E2C9D5                118               DC     C'SCSYSIN'
0C00DB 40                            119               DC     00001C' '
CC00DC                               120 CSYSCUT       EQU    *                      8 BYTE(S)
0C00DC E2C3F2F8E2D6E4E3              121               CC     C'SCSYSOUT'
0C00F4                               122 CSYSPNCH      EQU    *                      8 BYTE(S)
0C0CF4 000000C000C00000              123               DC     X'000000CCCCC00CC0'
0C0CEC                               124 CSYSUT1       EQU    *                      8 BYTE(S)
0C00FC F2C3E2E8F2E4E3F1              125               DC     C'SCSYSUT1'
0C0040                               126               ORG    @CATA
CCCC40                               127               DS     00000180C
0C00F4                               128 @TEMPS        DS     OF
0C00F4                               129 @CATENC       EQU    *
GC0CC0                               130               END    PCCNAMES
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 6 of 10)



```
IFF285I   CNP2
IFF285I   VCL SER NCS= XR157 .                       PASSED
IEF285I   SYS69C91.T000052.RP001.SACETC.LCADSET      PASSED
IFF285I   VCL SER NCS= SYSPS3.
IFF285I   SYSCUT
IFF285I   VCL SER NCS= T77439.                       SYSOUT
IFF285I   SYS69C91.TOC0052.RP001.SACETC.DCG          DELETEC
IFF285I   VCL SFR NCS= 111111.
IEF285I   SYS69091.T000052.RP001.SACETC.FCG          DELETED
IFF285I   VCL SER NCS= 231100.
IEF285I   SYS69C91.T000052.RP001.SACETC.MCG          DELETEC
IFF285I   VCL SFR NCS= SYSRS3.
IFF285I   SYS69C91.T000052.RP001.SACETC.PSLGC        DELETEC
IFF285I   VCL SFR NCS= SYSRS3.
IFF285I   SYS1.MACLIB                                PASSEC
IFF285I   VCL SER NCS= SYSRS3.
//LKED  FXEC PGM=LINKEDIT,CCNC=(9,LT,STEP2),PARM='XREF,LIST,LET'          CC000028
//SYSLIB DD DSNAMF=PSLLIB,UNIT=2311,VCLUME=SER=111111,DISP=(OLD,KEEP)     CC000029
//SYSPRINT DD SYSCUT=A                                                    00C00030
//SYSLIN  DD DSNAME=&LCADSET,CISP=(OLC,CELETE)                            0C000031
//            DD DCNAME=SYSIN                                             C0000032
//SYSLT1 DD DSNAME=&CAT,UNIT=2311,CISP=(NEW,CELETE),                     X00000033
//              SPACE=(CYL,(2,1))                                         00000034
//SYSLMOD DD DSNAME=&GCSET(BSLGD),SPACE=(1024,(50,20,1)),UNIT=2311,      X00000035
//              DISP=(MCC,PASS)                                           00000036
//SYSABEND DD SYSCUT=A                                                    000C0037
IFF236I ALLCC. FCR SACFTC   LKED      TEST
IFF237I JCBLIB   CN 192
IFF237I SYSLIB   CN 193
IEF237I SYSLIN   CN 191
IFF237I SYSUT1   CN 190
IFF237I SYSLMCD  CN 191
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 7 of 10)

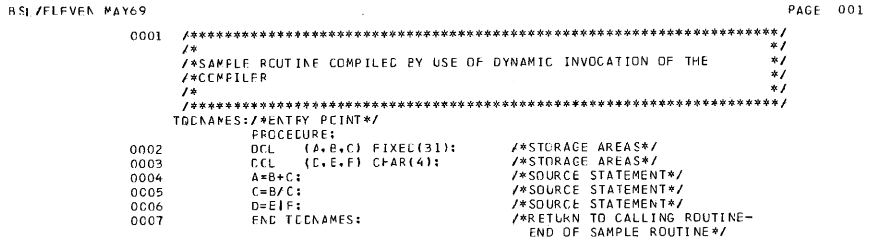


```
        PAGE    3          F44-LEVEL LINKAGE EDITCR CPTICNS SPECIFIEC XREF,LIST,LET
                                   VARIABLE OPTICNS USED - SIZE=(238976,102400)          DEFAULT OPTION(S) USED

F15APR68   4/01/69
                                                                 CRCSS REFERENCE TABLE
            00970
            00980
            00990        CCNTROL SECTICN                    ENTRY
            01000
            01010          NAME     ORIGIN  LENGTH            NAME   LOCATION     NAME   LOCATICN     NAME   LOCATICN     NAME   LOCATION
            01020
            01030        PODNAMES      00      F4
            01040
            01050
            01060
            01070        LCCATICN  REFERS TC SYMBOL  IN CCNTRCL SECTICN             LOCATION  REFERS TO SYMBOL  IN CCNTRCL SECTICN
            01080
            01090
            01100        ENTRY ADDRESS        00
            01110        TOTAL LENGTH         F8
            01120
            01130      ****RSLGO     DOES NOT EXIST BUT HAS BEEN ACDEC TC DATA SET
            01140
            01150
            01160
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 8 of 10)

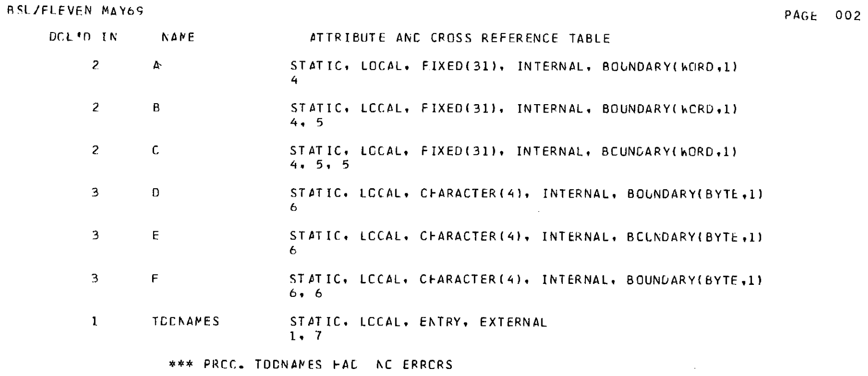
```
IFF2P5I    CNP2                                           PASSEC
IFF285I    VCL SFK NCS= XB157 .
IFF2P5I    BSLLIB                                         KEPT
IFF2P5I    VCL SER NCS= 111111.
IFF2P5I    SYSCLT                                         SYSOLT
IFF285I    VCL SER NCS= T77439.
IEF2P5I    SYS69091.T000052.RP001.SACETC.LCADSFT          DELETEC
IFF285I    VCL SER NCS= SYSRS3.
IEF2P5I    SYS69091.T000052.RP001.SACETC.CAT              CELETEC
IFF2P5I    VCL SFR NCS= 231100.
IFF285I    SYS69091.T000052.RP001.SACETC.GCSET            PASSED
IEF285I    VCL SFP NCS= SYSRS3.
//GC   FXEC PGM=*.LKEC.SYSLMCC,CCNC=(9,LT,LKEC)                         00000038
//SYSABEND DD SYSCUT=A                                                  00000039
//BSLOLT DD SYSCUT=A                                                    CC000040
//BSLPUNCH DD UNIT=2540-2                                               00000041
//GC.SCSYSLT1  DD DSNAME=&CAT,UNIT=2311,DISP=(NEW,DELETE),            X
//               SPACE=(CYL,(2,1))
//GO.SCSYSOUT   DD   SYSCUT=A
//GP.SCSYSIN  DC  *
IEF236I ALLCC. FCR SACETC   GC       TEST
IEF237I JCBLIB   CN 192
IFF237I PGM=*.DD CN 191
IFF237I BSLPUNCH CN 00C
IFF237I SCSYSLT1 CN 193
IEF237I SCSYSIN  CN 00C
```

Example 8. Use of Dynamic Invocation of Compiler (Part 9 of 10)



```
BSL/ELEVEN MAY69                                                                      PAGE  001
          0001  /*********************************************************************/
                /*                                                                    */
                /*SAMPLE ROUTINE COMPILED BY USE OF DYNAMIC INVOCATION OF THE         */
                /*COMPILER                                                            */
                /*                                                                    */
                /*********************************************************************/
               TOCNAMES:/*ENTRY POINT*/
                      PROCEDURE;
          0002        DCL    (A,B,C) FIXED(31):         /*STORAGE AREAS*/
          0003        DCL    (D,E,F) CHAR(4):           /*STORAGE AREAS*/
          0004        A=B+C:                            /*SOURCE STATEMENT*/
          0005        C=B/C:                            /*SOURCE STATEMENT*/
          0006        D=E|F:                            /*SOURCE STATEMENT*/
          0007        END TOCNAMES:                     /*RETURN TO CALLING ROUTINE-
                                                          END OF SAMPLE ROUTINE*/
```

• Example 8. Use of Dynamic Invocation of Compiler (Part 10 of 10)



```
BSL/ELEVEN MAY69                                                            PAGE  002
     DCL'D IN      NAME               ATTRIBUTE AND CROSS REFERENCE TABLE

         2         A               STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                   4

         2         B               STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                   4, 5

         2         C               STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                   4, 5, 5

         3         D               STATIC, LOCAL, CHARACTER(4), INTERNAL, BOUNDARY(BYTE,1)
                                   6

         3         E               STATIC, LOCAL, CHARACTER(4), INTERNAL, BOUNDARY(BYTE,1)
                                   6

         3         F               STATIC, LOCAL, CHARACTER(4), INTERNAL, BOUNDARY(BYTE,1)
                                   6, 6

         1         TDDNAMES        STATIC, LOCAL, ENTRY, EXTERNAL
                                   1, 7

              *** PROC. TDDNAMES HAD  NO ERRORS
```

Appendix E: Diagnostic Messages and Codes

When an incorrect statement is encountered in your BSL program, the compiler flags the statement and writes a diagnostic message. At the point of error, a statement flag is printed in the form:

```
*** ERROR message-number ***
```

Following the attribute and cross-reference table, the related diagnostic message is printed in the form:

```
statement   message   message text
number      number
```

Message Numbers and Severity Levels

Each of the compiler's diagnostic messages is numbered in the form (M)snn where:

- M indicates that the error occurred during the compiler's macro phase. The absence of an M indicates that the error occurred during the compile phase.
- s is an alphabetic character that indicates the severity level of the error that occurred.
- nn is an integer that numbers the message within its severity group.

The compiler recognizes four severity levels. These levels and their meanings are summarized in Table 11 (for the macro phase) and Table 12 (for the compile phase).

Table 11. Severity Levels of Diagnostic Messages (Macro Phase) (Part 1 of 2)

Level	Explanation
MW	is an abbreviation for a macro phase "Warning Message." This prefix indicates that either: • The statement had an error that was repaired by the compiler's macro phase. • The statement is not in error, but it may produce unexpected results when the program is compiled. This severity level is intended to draw your attention to potential errors.
ME	is a abbreviation for a macro phase "Error Message." This prefix indicates that the flagged statement is definitely incorrect, and compile-time processing of the statement is terminated. This severity level is used when the error is localized to a single statement. Processing continues from the next statement.

(Part 1 of 2)

MSG

Table 11. Severity Levels of Diagnostic Messages (Macro Phase) (Part 2 of 2)

Level	Explanation
MS	is an abbreviation for a macro phase "Serious Error Message." This prefix indicates that the flagged statement is definitely incorrect. The flagged statement is not processed. This severity level is used when the error cannot be localized to a single statement and, therefore, the error may affect the remaining statements. Processing continues from the next statement.
MD	is an abbreviation for a macro phase "Disastrous Error Message." This prefix indicates an error of the highest severity, and processing is terminated at the point of error.

Table 12. Severity Levels of Diagnostic Messages (Compile Phase)

Level	Explanation
W	is an abbreviation for a "Warning Message." This prefix indicates that either: • The statement had an error that was repaired by the BSL compiler. • The statement is not in error, but it may produce unexpected results when the program is executed. This severity level is intended to draw your attention to potential errors.
E	is an abbreviation for an "Error Message." This prefix indicates that the flagged statement is definitely incorrect. The compiler does not generate code for the flagged statement. This severity level is used when the error is localized to a single statement. Compilation continues from the next statement, and assembler text is produced.
S	is an abbreviation for a "Serious Error Message." This prefix indicates that the flagged statement is definitely incorrect. The compiler does not generate code for the flagged statement. This severity level is used when the error cannot be localized to a single statement and, therefore, the error may affect other statements in the compilation. Compilation continues from the next statement, but assembler text is not produced.
D	is an abbreviation for a "Disastrous Error Message." This prefix indicates an error of the highest severity, and compilation is terminated at the point of error. Assembler text is not produced.

Compiler Return Codes

When the compilation is completed or terminated, the compiler places a return code value in register 15. These return codes, which indicate the error status of the compilation, are:

Return Code	Error Status
0	No errors.
4	W-type errors were the highest level that occurred.
8	E-type errors were the highest level that occurred.
12	S-type errors were the highest level that occurred.
16	D-type errors were the highest level that occurred.

If the return code is 12 or 16, the compiler does not produce assembler text. If you assemble under OS, the cataloged procedures BSLASM and BSLALG test the compiler return codes to determine whether or not to attempt assembly.

The macro phase does not set a return code that can be tested by the user. If an error above the warning level occurs during the macro phase, the compile phase (when it gets control) sets the return code to 16 and returns to the system without attempting compilation.

Console Error Messages (OS)

If the BSL compiler cannot open one of its required data sets, the compiler terminates with a user code of 016. A message is written on the console device. This message is:

```
UNABLE TO OPEN xxxxxx
```

where xxxxxx is the ddname.

If invalid parameters are passed to the compiler when it is dynamically invoked, a compiler error may occur while trying to process the parameters. The following disastrous error message is written on the console device:

```
INCORRECT INVOCATION OF THE BSL COMPILER
```

Compilation is terminated at the point of error and no assembly text is produced. (See also diagnostic message D19 explanation.)

Message Explanations (Macro Phase)

In the following listings, you will find an explanation for each of the diagnostic messages that may occur during the macro phase. These explanations will assist you in interpreting the messages and correcting the errors. There is a separate listing for messages of each severity level (W, E, S, and D). Within each severity level, the messages are listed numerically by message number.

Macro Warning Messages

MW01 xxxxxxxx HAS NOT BEEN SET. REPLACEMENT CANNOT OCCUR.

Explanation: The named macro variable has been activated, but a value has not been assigned to it. For example:

```
%DCL A CHAR;
B = A;  /* MACRO ASSIGNMENT HAS NOT
           APPEARED */
%A = 'XYZ';
```

Compiler Action: The macro variable is not replaced by an assigned value.

MW02 xxxxxxxx APPEARS AS A COMPILE-TIME LABEL, BUT HAS BEEN DEFINED PREVIOUSLY. IT IS IGNORED.

Explanation: The named item is a label on a compile-time statement. It has also appeared as a label on a previous compile-time statement, or has been declared FIXED or CHAR in a compile-time DECLARE statement.

Compiler Action: The label is ignored.

MW03 xxxxxxxx IS MULTIPLY DEFINED. THE SECOND DEFINITION IS IGNORED.

Explanation: The named item has appeared as a label on a previous compile-time statement, or has been declared in a previous compile-time DECLARE statement.

Compiler Action: The second definition is ignored.

MW04 xxxxxxxx CANNOT BE ACTIVATED/ DEACTIVATED BECAUSE IT IS UNDEFINED OR IS A COMPILE-TIME LABEL.

Explanation: The named item is not the type of item that can be activated or deactivated. If the item is undefined, xxxxxxxx will be replaced by the word VARIABLE. If the item is a compile-time label, xxxxxxxx will be replaced by the label name.

Compiler Action: The item in the ACTIVATE or DEACTIVATE statement is ignored.

MW05 xxxxxxxx IS A COMPILE-TIME KEYWORD BUT APPEARS AS A COMPILE-TIME LABEL. THE LABEL IS IGNORED.

Explanation: A keyword is used as a compile-time label. For example:

```
%IF:  GO TO L1; /* IF IS A KEYWORD */
```

Compiler Action: The label on the compile-time statement is ignored.

MW06 MORE THAN ONE PERCENT APPEARS FOLLOWING A COMPILE-TIME THEN OR ELSE. IT IS IGNORED.

Explanation: A THEN or ELSE is followed by more than one % sign. For example:

```
%X = 10;
%IF X = 5 %THEN %A = %Y = 6;
    /* TWO % FOLLOW THEN */
```

Compiler Action: The additional macro expressions are ignored.

MW07 THERE IS A MIXTURE OF COMPILE-TIME AND NON COMPILE-TIME STATEMENTS ON A CARD. PERCENT INSERTED.

Explanation: A compile-time statement is followed by a non compile-time statement on the same card or a percent is missing where one is expected. For example:

```
%X = 10; Y = 5; /* Y = 5 DOES NOT
HAVE % PRECEDING IT */
%IF A = 5 THEN % B = 10; /* A % WAS
EXPECTED BEFORE THEN */
```

Compiler Action: Percent assumed.

MW08 THERE IS A MIXTURE OF NON COMPILE-TIME AND COMPILE-TIME STATEMENTS ON A CARD. PERCENT IGNORED.

Explanation: A non compile-time statement is followed by a compile-time statement on the same card. For example:

```
NAMELST='OF'X;  % Y=10;  /* %APPEARS
BEFORE Y=10 */
```

Compiler Action: Percent ignored.

Macro Error Messages

ME01 A COMPILE-TIME REPLACEMENT LENGTH IS GREATER THAN 1000 BYTES. IT IS TRUNCATED.

Explanation: A string is specified to replace a compile-time variable name, and the string is greater than 1000 bytes.

Compiler Action: The first 1000 bytes are used.

| ME02 (Unassigned)

ME03 A COMPILE-TIME VARIABLE CONTAINS AN UNBALANCED QUOTE OR COMMENT.

Explanation: The replacement text for a compile-time variable contains an unbalanced quote or comment. For example:

```
%DCL (A, B) CHAR;
%A ='/*THIS IS';
```

will cause A to be as an unbalanced comment after substitution is made.

Compiler Action: The condition is accepted. The assignment is made as specified.

ME04 A COMPILE-TIME THEN APPEARS IN AN INCORRECT POSITION. THE COMPILE-TIME STATEMENT IS SKIPPED.

Explanation: Self-explanatory.

Compiler Action: Everything up to and including the next semicolon (;) is ignored.

ME05 A COMPILE-TIME DECLARE IS NOT TERMINATED BY A SEMICOLON. SCANNING RESUMES AFTER NEXT SEMICOLON.

Explanation: Self-explanatory.

ME06 A DEACTIVATE OR ACTIVATE LIST HAS AN ILLEGAL DELIMITER. THE REST OF THE STATEMENT IS IGNORED.

Explanation: An illegal delimiter is contained in an ACTIVATE or DEACTIVATE list. For example:

```
%DEACT A,B: C; /*ILLEGAL DELIMITER*/
%DEACT A,B,C? /*ILLEGAL DELIMITER*/
```

Compiler Action: The statement is ignored from the point of the illegal delimiter.

ME07 xxxxxxxx WAS THE TARGET OF A COMPILE-TIME GOTO, BUT NOW APPEARS IN A NON-LABEL POSITION. IT IS IGNORED.

Explanation: The named item is used in a non-label position, but was previously the target of a GOTO statement. For example:

```
%GOTO L1;
%A = L1; /* CONFLICTING USAGE */
```

Compiler Action: The statement is ignored.

ME08 THE LENGTH OF A COMPILE-TIME VARIABLE EXCEEDS 1000 BYTES. THE LENGTH IS RESET TO 0.

Explanation: The concatenation of a compile-time string variable resulted in a string length of more than 1000 bytes.

Compiler Action: The variable receives a length of 0.

ME09 CONCATENATING A STRING CONSTANT HAS CAUSED THE LENGTH OF A STRING VARIABLE TO EXCEED 1000 BYTES.

Explanation: The concatenation of a compile-time string constant resulted in a string length of more than 1000 bytes.

Compiler Action: The string variable receives a length of 0.

ME10 THE NUMBER OF COMPILE-TIME ERRORS EXCEEDS 99. THE REMAINING ERRORS ARE NOT LISTED.

Explanation: The compiler lists a maximum of 99 compile-time error messages.

Macro Serious Error Messages

MS01 AN ILLEGAL CHARACTER APPEARS IN A COMPILE-TIME STATEMENT. THE REST OF THE STATEMENT IS IGNORED.

Explanation: The first character following the % is not an alphabetic character, a semicolon, or a blank.

MS02 THE RECEIVER IN A COMPILE-TIME ASSIGNMENT STATEMENT HAS NOT BEEN PREVIOUSLY DECLARED. THE STATEMENT IS IGNORED.

Explanation: An assignment was made to an item that was not previously declared in a compile-time DECLARE statement.

MS03 THE RECEIVER ON A COMPILE-TIME ASSIGNMENT STATEMENT IS A LABEL. THE STATEMENT IS IGNORED.

Explanation: A compile-time assignment was made to an item that was previously used or declared as a label.

MS04 ILLEGAL NAME IN COMPILE-TIME DECLARE. THE DECLARE IS IGNORED.

Explanation: The compile-time DECLARE statement contains an illegal name. For example:

```
%DECLARE ?   ; /* ILLEGAL NAME */
```

MS05 A , OR) APPEARS IN A COMPILE-TIME DECLARE THAT HAS NO FACTORING. THE STATEMENT IS IGNORED.

Explanation: A comma or a right parenthesis has been encountered in a compile-time DECLARE statement, but the statement contains no factored items.

MS06 A COMPILE-TIME DECLARE STATEMENT HAS AN ILLEGAL ATTRIBUTE. THE STATEMENT IS IGNORED.

Explanation: A compile-time DECLARE statement contains an illegal or unidentifiable attribute. For example:

```
%DCL (A, B, C) GLOM ;
    /* NO SUCH ATTRIBUTE */
```

MS07 AN UNKNOWN RELATIONAL OPERATOR HAS BEEN ENCOUNTERED. THE STATEMENT IS IGNORED.

Explanation: A relational operator is illegal. For example:

```
%IF A ?= B %THEN %GOTO L1;
    /* ?= IS ILLEGAL */
```

MS08 THERE ARE UNEQUAL LENGTHS ON A STRING COMPARISON. THE STATEMENT IS IGNORED.

Explanation: Both operands of a compile-time comparison expression are not the same length. For example:

```
%DCL (A,B) CHAR;
%A = 'X';
%B = 'XYZ';
%IF A = B %THEN %GOTO L1;
   /*A AND B ARE DIFFERENT LENGTHS*/
```

MS09 THE TARGET OF A COMPILE-TIME GCTO DOESN'T START WITH AN ALPHABETIC. THE GOTO IS IGNORED.

Explanation: An illegal identifier is the target of a compile-time GOTO. For example:

```
%GOTO 1C; /* ILLEGAL IDENTIFIER */
```

MS10 (Unassigned)

MS11 xxxxxxxx HAS APPEARED PREVIOUSLY IN TEXT, BUT IT IS THE TARGET OF A COMPILE-TIME GOTO. THE GOTO IS IGNORED.

Explanation: The compile-time facility does not allow a branch backward to a label that previously appeared.

MS12 AN ILLEGAL OPERAND IS FOUND IN A COMPILE-TIME STATEMENT. THE REST OF THE STATEMENT IS IGNORED.

Explanation: The operand of a compile-time statement is not a name, a decimal number, or a character string.

MS13 A NAME OR NUMBER IN A COMPILE-TIME STATEMENT IS LONGER THAN 8 CHARACTERS. THE STATEMENT IS IGNORED.

Explanation: An identifier or a number in a compile-time statement is illegal because it has more than eight characters.

MS14 xxxxxxxx IS IN A COMPILE-TIME EXPRESSION BUT IS A LABEL, KEYWORD, OR UNDEFINED. THE STATEMENT IS IGNORED.

Explanation: The named item is used illegally in a compile-time expression. For example:

```
%A = B + GOTO; /*GOTO IS A KEYWORD*/
```

MS15 xxxxxxxx FOLLOWS A COMPILE-TIME IF, BUT IT IS A KEYWORD. THE STATEMENT IS IGNORED.

Explanation: The named item is a keyword and is used illegally following a compile-time IF.

MS16 A COMPILE-TIME ASSIGNMENT STATEMENT DOES NOT HAVE AN '=' OPERATOR. THE STATEMENT IS IGNORED.

Explanation: The = operator is missing from a compile-time assignment statement.

MS17 THERE ARE MORE THAN 2 TERMS IN A COMPILE-TIME ARITHMETIC EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: A maximum of two terms is allowed in a compile-time arithmetic expression. For example:

```
%A = B + C + D; /*TOO MANY TERMS*/
```

MS18 THERE IS AN ILLEGAL OPERATOR IN A COMPILE-TIME EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: Self-explanatory. For example:

```
%A = B ** C;  /* ILLEGAL OPERATOR */
%X = Y & Z;  /* ILLEGAL OPERATOR */
```

MS19 A COMPILE-TIME IF CLAUSE IS NOT FOLLOWED BY A THEN CLAUSE. THE STATEMENT IS IGNORED.

Explanation: A compile-time IF statement is incomplete because the IF clause is not followed by a THEN clause.

MS20 A NUMBER APPEARS AS AN OPERAND OUTSIDE OF A COMPILE-TIME ARITHMETIC EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: The compile-time expression is not arithmetic, but has a number as an operand. For example:

```
%DCL C CHAR;
%C=1; /*1 IS ILLEGAL OPERAND*/
```

MS21 xxxxxxxx IS A COMPILE-TIME VARIABLE THAT DOES NOT HAVE A VALUE, BUT IS BEING USED IN A COMPILE-TIME EXPRESSION.

Explanation: A compile-time variable cannot appear in a compile-time expression unless it has been previously assigned a value.

Compiler Action: The statement is ignored.

MS22 THE RESULT OF A COMPILE-TIME ARITHMETIC EXPRESSION IS MORE THAN 8 DIGITS. THE STATEMENT IS IGNORED.

Explanation: Self-explanatory. For example:

```
%DCL (A, B, C) FIXED;
%A = 99999999;
%B = 2;
%C = A + B; /* RESULT IS MORE THAN 8
                 DIGITS */
```

MS23 AN ILLEGAL NUMBER APPEARS IN A COMPILE-TIME EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: Self-explanatory. For example:

```
%A=B+28X; /*28X IS ILLEGAL NUMBER*/
```

MS24 AN OPERATOR OTHER THAN CONCATENATION IS IN A COMPILE-TIME STRING EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: The concatenation operator is the only operator allowed in a compile-time string expression. For example:

```
%DCL C CHAR;
%C = 'XYZ' || C + 3;
    /* OPERATOR + IS ILLEGAL */
```

MS25 A CHARACTER STRING CONSTANT APPEARS OUTSIDE OF A COMPILE-TIME STRING EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: A character string constant can only be used in a compile-time expression. For example:

```
%A = B + 'XYZ';
    /*ILLEGAL ARITH EXPRESSION*/
```

MS26 xxxxxxxx IS AN OPERAND OF A COMPILE-TIME ARITHMETIC STATEMENT BUT IS NOT ARITHMETIC. THE STATEMENT IS IGNORED.

Explanation: The named item is illegally used as an operand of a compile-time arithmetic statement. For example:

```
%DCL A FIXED, C CHAR;
%A = A + C; /*C IS NOT ARITHMETIC*/
```

MS27 xxxxxxxx IS AN OPERAND OF A COMPILE-TIME STRING STATEMENT, BUT IS NOT A STRING. THE STATEMENT IS IGNORED.

Explanation: The named item is illegally used as an operand of a compile-time string statement. For example:

```
%DCL C CHAR, A FIXED;
%C='YXZ' || A; /*A IS NOT STRING*/
```

MS28 THERE IS A DIVISION BY 0 IN A COMPILE-TIME ARITHMETIC EXPRESSION. THE STATEMENT IS IGNORED.

Explanation: Division by zero is illegal. For example:

```
%DCL (A, B) FIXED;
%A = 0; %B = 6;
%B = B/A;  /* DIVISION BY ZERO */
```

Macro Disastrous Error Messages

MD01 AN END-OF-FILE IS ENCOUNTERED BEFORE NORMAL END OF TEXT. COMPILE-TIME PROCESSING IS TERMINATED.

Explanation: End-of-file is found before the normal end of text. For example:

- In the middle of a compile-time statement.
- In the middle of a string.
- In the middle of a comment.
- While attempting to execute %GOTO, but before reaching the target label.
- In the middle of a BSL source statement.

MD02 THE COMPILE-TIME DICTIONARY OR WORK AREA HAS OVERFLOWED. ALL COMPILE-TIME PROCESSING IS TERMINATED.

Explanation: There is not enough space for compile-time processing because the number of compile-time variables and labels exceeds 500, or the total length of compile-time character strings exceeds 45,000 bytes.

MD03 A LOOP HAS OCCURRED IN THE COMPILE-TIME RESCAN. ALL COMPILE-TIME PROCESSING IS TERMINATED.

Explanation: Compile-time processing allows a maximum of 50 rescans, and then assumes that a loop has occurred.

MD04 THE INCLUDE STATEMENT IS INCORRECTLY WRITTEN.

Explanation: The ddname or member name is too long, or there are blanks between ddname(member).

MD05 THE INCLUDE STATEMENT ASKS FOR A LIBRARY MEMBER THAT CANNOT BE FOUND IN THE SPECIFIED LIBRARY.

Explanation: The member name specified in the INCLUDE statement cannot be found in the specified partitioned data set.

User Response: Check the spelling of the member name in the INCLUDE statement, and check the data set name on the DD statement indicated in the INCLUDE statement.

MD06 AN I/O ERROR HAS OCCURRED WHILE SEARCHING FOR THE REQUESTED MEMBER IN THE USER'S LIBRARY.

Explanation: I/O error.

MD07 THE PARTITIONED DATA SET SPECIFIED IN THE INCLUDE HAS A BLKSIZE GREATER THAN 3520 OR A RECFM OTHER THAN F.

Explanation: The partitioned data set may contain blocked records up to a block size of 3520 bytes, and must contain fixed format records.

User Response: Respecify the block size or record format of the data set and run the job again. Make sure that the INCLUDE statement specifies the correct data set.

MD08 AN INCLUDE STATEMENT APPEARS WITHIN INCLUDED TEXT.

Explanation: Included text may not contain an INCLUDE statement.

MD09 A COMPILER ERROR HAS OCCURRED IN THE MACRO PHASE. SUBMIT A TROUBLE REPORT TO DEPT. D76, POUGHKEEPSIE, N.Y.

Explanation: Self-explanatory.

Message Explanations (Compile Phase)

In the following listings, you will find an explanation for each of the diagnostic messages that may occur during the compile phase. These explanations will assist you in interpreting the messages and correcting the errors. There is a separate listing for messages of each severity level (W, E, S, and D). Within each severity level, the messages are listed numerically by message number.

Warning Messages

W01 VARIABLE HAS ILLEGAL PRECISION OR LENGTH. THE DEFAULT PRECISION OR LENGTH HAS BEEN USED.

Explanation: The declaration contains one of the following illegal precisions or length:

- A FIXED variable is declared with a precision other than 15 or 31. For example:

```
DCL A FIXED(25);  /* ILLEGAL */
```

- A POINTER variable is declared with a precision other than 8, 15, 16, 24, 31, or 32. For example:

```
DCL B PTR(17);  /* ILLEGAL */
```

- A BIT or CHAR variable is declared with no specified length. For example:

```
DCL C BIT;     /* ILLEGAL */
DCL D CHAR;   /* ILLEGAL */
```

Compiler Action: The default precision or length is used for the declared variable, as follows:

- A precision of 31 for a FIXED variable.
- A precision of 31 for a POINTER variable.
- A length of 1 for a BIT or CHAR variable.

W02 xxxxxxxx HAS A BOUNDARY GREATER THAN THAT OF ITS CONTAINING STRUCTURE. A BYTE BOUNDARY WAS USED.

Explanation: The boundary of the named variable is greater than the boundary of its containing structure, where DWORD>WORD>HWORD>BYTE. For example:

```
DCL 1A FIXED(31) BDY HWORD,
      2B CHAR(2),
      2C FIXED(31);  /* BDY GREATER
         THAN HWORD */
```

Compiler Action: The named variable is put on the next byte boundary.

W03 xxxxxxxx HAS MORE INITIAL VALUES THAN THERE ARE ELEMENTS TO BE INITIALIZED. THE EXTRA VALUES WERE IGNORED.

Explanation: One of the following:

- The named item is an array with a replication factor that specifies too many initial values. For example:

```
DCL A(10) INIT(1,2,(9)3);
   /*TOO MUCH REPLICATION*/
```

- The named item is an array with too many initial values. For example:

```
DCL B(3) INIT(1,2,3,4);
  /*TOO MANY VALUES*/
```

- The named item is not dimensioned, but contains a replication factor.

Compiler Action: All elements of the array are initialized, and the remaining initial values are not used.

W04 xxxxxxxx HAS A CONSTANT SUBSCRIPT THAT IS GREATER THAN THE NUMBER OF ELEMENTS DECLARED FOR THE ARRAY.

Explanation: The named item, which was declared to be an array, is being referred to with a constant subscript that is larger than its dimension. For example:

```
DCL A(10);
A(12) = B;  /*SUBSCRIPT TOO LARGE*/
```

W05 A DECLARED REGISTER IS ONE OF THE REGISTERS WHICH MAY BE REQUIRED FOR USE BY THE COMPILER.

Explanation: A variable is declared with REGISTER storage class. The register specified is register 0, 13, 14, or 15; or a CODEREG or DATAREG; or register 1 when there are formal parameters.

W06 xxxxxxxx IS A DIMENSIONED ITEM THAT APPEARS WITHOUT A SUBSCRIPT.

Explanation: The named item was declared to be an array, but is being referred to without a subscript. For example:

```
DCL A (10);
A = B; /* COMPILED AS A(1)=B */
```

Compiler Action: A subscript of 1 is assumed.

W07 (Unassigned)

W08 xxxxxxxx SHOULD NOT BE USED AS A LABEL. IT HAS BEEN PREVIOUSLY USED OR DECLARED AS OTHER THAN LOCAL LABEL.

Explanation: The named item is being used as a label, but was previously used or declared as other than a LOCAL label. For example:

```
    DCL P PTR, L LABEL BASED(P);

L:  CALL Q;  /* L IS A BASED LABEL AND
                SHOULD NOT APPEAR AS
                LABEL OF A STATEMENT */

L1:  A = B + C;

L1:  X = Y;  /* L1 HAS BEEN PREVIOUSLY
                USED AS A LABEL */
```

Compiler Action: The named item is not used.

W09 PROCEDURE STATEMENT WAS NOT FOUND. A CSECT WITH NO NAME WAS PRODUCED.

Explanation: The first statement in a compilation is neither a GENERATE statement nor a PROCEDURE statement.

Compiler Action: An unnamed procedure is assumed.

W10 (Unassigned)

W11 COMMENT TERMINATOR MAY BE MISSING. A SEMICOLON OR A /* APPEARS WITHIN THE COMMENT.

Explanation: A semicolon or a /* appears within a comment. For example:

```
/* A BSL STATEMENT ENDS WITH A; */

/* THE COMBINATION OF OPERATORS /*
SHOULD NOT START IN COLUMN 1 */
```

W12 VARIABLE HAS BEEN TRUNCATED TO EIGHT CHARACTERS.

Explanation: An identifier has more than eight characters.

Compiler Action: The identifier is shortened to eight characters by using the eight leftmost characters.

W13 xxxxxxxx MAY NOT BE ADDRESSABLE.

Explanation: The generated code may not correctly address the named item. For example:

```
DCL (B(10),A) CHAR(256);
A = B(I);
```

might be compiled as:

```
L 8,I
SLL 8,8
LA 10,B-256(8)
MVC A(256),0(10)
```

which would be incorrect if the array began less than 256 bytes from the start of addressability.

W14 VARIABLE IS ARITHMETIC, AND MAY RESULT IN AN ERROR WHEN USED IN THIS CONTEXT.

Explanation: The result of the assignment may not be what is expected. For example:

```
DCL C CHAR(1), V INIT(1);
C = V;
```

IS COMPILED AS MVC C(1),V which sets C to '00'X.

W15 xxxxxxxx HAS PREVIOUSLY RECEIVED DIFFERENT ATTRIBUTES. THESE ATTRIBUTES ARE OVERRIDDEN AT THIS POINT.

Explanation: The label or entry name was previously declared with different attributes. For example:

```
   P = ADDR(L);  /* L DEFAULTS TO
            FIXED(31) */
     .
     .
     .

L:  A = A + 1;
    /* L GETS AN OVERRIDDING ATTRIBUTE
    OF LABEL */
```

Compiler Action: The previous attribute is overridden at this point, and the result may be incorrect or inefficient.

W16 xxxxxxxx HAS SIGNIFICANT HIGH ORDER BIT, BUT CODE MAY HAVE BEEN PRODUCED THAT ASSUMES ITS HIGH ORDER BIT IS 0.

Explanation: Code may have been produced assuming that the high-order bit of the named item is 0. For example:

```
DCL P32 PTR(32), A FIXED;
IF P32 = A THEN GOTO L1;
    /* COMPARISON MAY BE WRONG */
```

W17 THE USE OF REGISTER 1 MAY BE INCORRECT. IT IS REQUIRED FOR USE AS A PARAMETER LIST OR ARGUMENT LIST POINTER.

Explanation: The use of register 1 may be incorrect for one of the following reasons:

- A CALL statement with arguments appears in a PROCEDURE with formal parameters and register 1 was not saved. The user must save and restore register 1 around the CALL statement before making another reference to a parameter.
- A PROCEDURE statement has formal parameters and register 1 has not been saved.

W18 xxxxxxxx APPEARS ON A PROCEDURE END STATEMENT, BUT IS NOT THE NAME OF THE PROCEDURE.

Explanation: The named item is an identifier that follows a procedure END statement, but is not the name of the procedure that the END statement will close. For example:

```
A:  PROC;
    .
    .
    .
   END B; /* NOT PROCEDURE NAME */
```

W19 AUTOMATIC ATTRIBUTE CANNOT BE SPECIFIED IN A NON-REENTRANT PROCEDURE. AUTOMATIC HAS BEEN REPLACED BY STATIC.

Explanation: The AUTOMATIC attribute is specified for an item, but the external procedure does not specify the REENTRANT option.

Compiler Action: The item is given the STATIC attribute.

W20 xxxxxxxx IS NOT IN THE SCOPE OF THIS PROCEDURE.

Explanation: The named item was declared in a disjoint procedure. For example:

```
A:  PROC;
B:  PROC;
   DCL X;
   END B;
    /*PROC B IS DISJOINT FROM C*/
C:  PROC;
   X = Z; /*X IS DECLARED IN PROC B*/
   END C;
   END;
```

W21 VARIABLE HAS AN ILLEGAL DIMENSION. A DIMENSION OF ONE HAS BEEN USED.

Explanation: The value specified for a dimension is not a decimal constant between 1 and 32767.

Compiler Action: A constant of one is used for the dimension.

W22 xxxxxxxx HAS CONFLICTING BASED AND BOUNDARY ATTRIBUTES; THE BASE WAS ADJUSTED TO AGREE WITH THE BOUNDARY.

Explanation: The named item is declared with both BASED and BOUNDARY attributes. The BOUNDARY attribute specifies a boundary that differs from that of the item in the BASED attribute.

Compiler Action: The base is adjusted upward to the next higher boundary that agrees with the boundary in the BOUNDARY attribute. For example:

```
DCL A BASED(3) BDY(WORD);
```

Location 3 is not a word boundary, so A will be based on 4 instead of 3.

W23 TRACE OPTION WAS SPECIFIED FOR A PROCEDURE WITH NO SAVE AREA. TRACE WILL BE DONE USING THE VALUE IN REGISTER 13.

Explanation: The TRACE option is specified, but the procedure has no save area.

Compiler Action: Tracing code is generated assuming that register 13 points to a save area which is at least as large as the specified offset.

W24 PROCEDURE SPECIFIED CODEREG(0) AND TRACE OPTION HAS BEEN USED. USER MUST GENERATE CODE TO INITIALIZE TRACING.

Explanation: The compiler did not generate addressability for the procedure because of the CODEREG(0) option. With no addressability, the compiler cannot generate code to initialize the tracing operation.

Compiler Action: The compiler assumes that the user has provided the code to establish addressability and initialize tracing.

W25 xxxxxxxx HAS BEEN USED OUTSIDE THE SCOPE OF THIS PROCEDURE.

Explanation: The named variable is a label that was referred to by a GOTO statement from outside the scope of this procedure.

W26 A LABEL APPEARS ON A DECLARE, RESTRICT, OR RELEASE STATEMENT. THE LABEL WILL BE APPLIED TO THE NEXT STATEMENT.

Explanation: DECLARE, RESTRICT, and RELEASE statements should not be labeled.

Compiler Action: The label will be used on the next statement that can have a label.

W27 xxxxxxxx IS BASED ON REGISTER ZERO, WHICH CANNOT BE USED DIRECTLY AS A BASE REGISTER.

Explanation: This warning is given on the declare statement where the named item is declared. A register other than zero should be used as a base.

W28 THE OPTION NOSAVEAREA WAS SPECIFIED IN THE CONTAINING PROCEDURE. REGISTER 13 MAY CONTAIN AN INCORRECT ADDRESS.

Explanation: An internal procedure is contained in a procedure that used the NOSAVEAREA procedure option, and the internal procedure has used no procedure options that would suppress save area chaining. For example:

```
A:  PROC OPTIONS(NOSAVEAREA);
          .
          .
          .
      CALL B;
          .
          .
          .
B:  PROC;
          .
          .
          .
      END B;

      END A;
```

Compiler Action: Save area chaining will be done using the current value in register 13.

W29 (Unassigned)

W30 xxxxxxxx IS A REGISTER VARIABLE AND HAS NOT BEEN RESTRICTED. INCORRECT CODE MAY BE GENERATED.

Explanation: The indicated register is still available for use by the compiler as long as it has not been restricted by the programmer. If the programmer needs to maintain a certain value in a register for a particular section of code, that register should be restricted in order to prevent the compiler from allocating it for generated code.

W31 AN END STATEMENT WAS ENCOUNTERED FOR A DO STATEMENT WHICH CONTAINED AN ERROR.

Explanation: The corresponding DO statement for this END statement:

- contained an error.
- was contained in a dangling ELSE clause.

The END statement has been matched with a DO statement for which no code was generated. Thus, no loop return code will be generated for the END statement. When the error in the DO statement or the error that caused the dangling ELSE has been corrected, the END statement will be correctly processed.

W32 xxxxxxxx IS A STRUCTURE WHOSE SPECIFIED SIZE IS LESS THAN THE TOTAL SIZE OF ITS COMPONENTS.

Explanation: The named structure is a major or minor structure with a declared size and the total size of this structure's components exceeds that size.

W33 xxxxxxxx HAS A BOUNDARY REQUIREMENT WHICH CAUSES BYTES TO BE SKIPPED IN THE MAPPING OF THE STRUCTURE.

Explanation: The named variable is in a structure and it has a declared or implied boundary which causes bytes to be skipped in mapping the structure. For example:

```
DCL 1 RECORD,
      2 NAMEFLD CHAR(27),
      2 CODE FIXED(31);
```

There will be a byte skipped between NAMEFLD and CODE because CODE requires a word boundary.

Error Messages

E01 xxxxxxxx HAS BEEN PREVIOUSLY DECLARED. THIS DECLARATION HAS NOT BEEN PROCESSED.

Explanation: The named item was declared previously, either explicitly or by default.

Compiler Action: The declaration is not processed.

E02 MORE THAN 20 LEVELS OF FACTORING ARE USED.

Explanation: The DECLARE statement has more than 20 as yet unmatched left parentheses.

Compiler Action: The statement is processed as if factoring ended after 20 levels.

E03 A DATA NAME WAS EXPECTED BUT NOT FOUND.

Explanation: A DECLARE statement has something other than the name of a data item in a place where a data name is expected. For example:

```
DCL A FIXED, ;
    /* DATA NAME IS EXPECTED FOLLOWING
    THE COMMA */
```

E04 A RIGHT PARENTHESIS WAS EXPECTED BUT NOT FOUND. ONE WAS ASSUMED.

Explanation: Self-explanatory. For example:

```
DCL A FIXED(15;
    /* RIGHT PAREN IS MISSING */
```

Compiler Action: A right parenthesis is assumed.

E05 xxxxxxxx HAS CONFLICTING ATTRIBUTES. THE SECOND ATTRIBUTE WAS IGNORED.

Explanation: The named item has conflicting attributes. For example:

```
DCL A INTERNAL EXTERNAL;
    /* CONFLICTING ATTRIBUTES */
```

Compiler Action: The second attribute is ignored.

E06 A KEYWORD, OPERATOR, OR DELIMITER APPEARS IN AN INCORRECT POSITION.

Explanation: The position of a keyword, operator, or delimiter is syntactically incorrect. For example:

```
A=B+; /*OPERATOR PRECEDES SEMICOLON*/
```

E07 A CONSTANT SUBSCRIPT LARGER THAN 32767 HAS BEEN USED.

Explanation: The maximum value of a subscript constant is 32,767. For example:

```
A(32768) = 0;  /* ILLEGAL SUBSCRIPT */
```

E08 OPTION SPECIFIED WITH AN ENTRY ATTRIBUTE IS INVALID OR INCORRECTLY WRITTEN.

Explanation: The option specified is not 'VLIST' or the format is incorrect. For example:

```
DCL A ENTRY OPTIONS VLIST);
    /*No left paren*/

DCL B ENTRY OPTIONS(DONTSAVE);
    /*Option is not VLIST*/
```

E09 VARIABLE HAS AN ILLEGAL BOUNDARY ATTRIBUTE. THE NORMAL DEFAULT BOUNDARY WAS USED.

Explanation: The BOUNDARY attribute is incorrectly written. For example:

```
DCL A FIXED BDY(HWRD);
    /* SHOULD BE SPELLED HWORD */
```

E10 CALL STATEMENT HAS AN ILLEGAL ARGUMENT -- A REGISTER VARIABLE OR AN EXPRESSION USING A BIT OPERATOR.

Explanation: The following items cannot be used as arguments in a CALL statement:

- Register variables.
- Expressions using only bit operators (&,|,&&).

For example:

```
DCL R3 REG(3), X CHAR(4), Y CHAR(5);
CALL SUBR(X&Y, R3);
    /* ILLEGAL ARGUMENTS */
```

E11 STATEMENT CONTAINS UNBALANCED PARENTHESES.

Explanation: There are more right parentheses than left parentheses. For example:

```
DCL A,B) FIXED;
    /*MISSING LEFT PAREN*/
```

E12 A SEMICOLON WAS ENCOUNTERED BEFORE FACTORING WAS CLOSED OUT. A RIGHT PARENTHESIS WAS ASSUMED AT THAT POINT.

Explanation: In a DECLARE statement, a semicolon was encountered in the list of names having factored attributes. For example:

```
DCL (A, B FIXED;
    /* INCOMPLETE FACTORING */
```

Compiler Action: A right parenthesis is assumed at the point where the semicolon was encountered.

E13 xxxxxxxx CANNOT HAVE AN INITIAL VALUE.

Explanation: The named variable cannot be initialized because it is part of a BASED or AUTOMATIC structure. For example:

```
DCL 1 A BASED,
      2 B INIT(10),
          /*ILLEGAL INITIALIZATION*/
      2 C;
```

E14 VARIABLE DOES NOT HAVE AN ATTRIBUTE WHERE AN ATTRIBUTE IS EXPECTED.

Explanation: A non-attribute appears where an attribute is expected. For example:

```
DCL A FIXED B EXT;
    /* MISSING COMMA BEFORE B */
```

E15 xxxxxxxx HAS AN ILLEGAL INITIAL VALUE.

Explanation: The initial value for the named item is illegal for one of the following reasons:

- The syntax of the initial value is incorrect. For example: ('0101A'B).
- Initial values are not separated by commas.
- Initial values are not terminated by a right parenthesis.
- Initial values contain a semicolon.
- The initial value is a null string.
- An initial string value is longer than the declared length of the string.
- A FIXED or POINTER item is initialized with a character or bit string.
- A CHARACTER or BIT item is initialized with a binary or decimal number.
- A FIXED, CHARACTER, or BIT item is initialized with the ADDR function.
- The syntax of an ADDR function is incorrect.
- In ADDR(name), the name is not a STATIC item.
- A replication factor is not a decimal number.

E16 SUBSTRING NOTATION IS IMPROPERLY WRITTEN OR INCORRECTLY USED.

Explanation: The substring notation is illegal for one of the following reasons:

- A subscript is used with a non-dimensioned LABEL item. For example:

  ```
  DCL L LABEL;
  GO TO L(2); /*ILLEGAL SUBSCRIPT*/
  ```

- A substring is used on a FIXED or POINTER item.
- A variable range or variable offset substring is used in an arithmetic or CALL statement, or in an ADDR function.
- A variable range is used for the assignment of one byte. For example:

  ```
  A(I:J)='A'; /*ILLEGAL RANGE*/
  ```

E17 xxxxxxxx IS A REGISTER USED IN A STRING EXPRESSION OR WITH A SUBSCRIPT OR SUBSTRING.

Explanation: The named variable is a register; therefore, it may not be subscripted, substringed, or used in a string expression. For example:

```
DCL R2 REG(2), A CHAR(4), B CHAR(4);
F31 = R2(2);
    /* USED WITH A SUBSTRING */
A = B & R2;
    /* USED IN STRING EXPRESSION */
```

E18 AN ILLEGAL COMBINATION OF OPERATORS WAS USED. AN OPERAND MAY BE MISSING.

Explanation: The combination of operators is illegal for one of the following reasons:

- An operand is missing. For example:

  ```
  A = B + ; /* MISSING OPERAND */
  ```

- The argument of an ADDR function is a number. For example:

  ```
  P = ADDR(24);
    /* ILLEGAL ADDR ARGUMENT */
  ```

E19 'DO' IS NOT FOLLOWED BY SEMICOLON, EQUAL, OR A SERIES OF POINTERS FOLLOWED BY EQUAL.

Explanation: The DO statement is written incorrectly, the control variable is subscripted or substringed, or is more than four bytes long. For example:

```
DO I 1 TO 10 BY 3;
    /* INCORRECTLY WRITTEN */
DO A(I) = 1 TO 10;
  /*SUBSCRIPT WITH CONTROL VARIABLE*/
DCL C5 CHAR(5);
DO C5=1 TO 10;
  /*VARIABLE MORE THAN FOUR BYTES
    LONG*/
```

E20 A SEMICOLON APPEARS BEFORE THE END OF A FORMAL PARAMETER LIST.

Explanation: A simicolon is encountered before the right parenthesis of a formal parameter list. For example:

```
A:  PROC (X, Y, Z ;
    /* SEMICOLON BEFORE PAREN */
```

E21 'GENERATE' IS NOT FOLLOWED BY A LEFT PAREN OR A SEMICOLON, OR IS NOT ENDED BY A RIGHT PAREN AND A SEMICOLON.

Explanation: The keyword GENERATE must be followed by a left parenthesis (for a simple GENERATE) or a semicolon (for a block GENERATE). A simple GENERATE statement must be ended by a right parenthesis and a semicolon. For example:

```
GEN TRT 0(100,7),TABLE);
    /* MISSING LEFT PAREN */

GEN (TRT 0(100,7),TABLE)
    /* MISSING SEMICOLON */
```

This message may also occur if a simple GENERATE statement covers more than one card.

E22 A PARAMETER IS MISSING.

Explanation: Self-explanatory. For example:

```
X:  PROC (A, ,B);
    /* SUCCESSIVE COMMAS IN PARAMETER
    LIST */
```

E23 THE LEFT SIDE OF A RELATIONAL EXPRESSION CONTAINS BOTH STRING AND ARITHMETIC DATA WITH NO ARITHMETIC OPERATORS.

Explanation: Self-explanatory. For example:

```
DCL A FIXED, B CHAR(2);
IF (A & B) = 10 THEN GOTO L1;
    /* ILLEGAL */
```

E24 (Unassigned)

E25 STATEMENT CONTAINS TOO MANY TERMS OR TOO MANY ARGUMENTS.

Explanation: The statement is illegal for one of the following reasons:

- A statement contains too many terms.
- A CALL statement contains more than 25 arguments.
- An IF statement requires more than 24 true/false branches.

E26 AN INVALID CHARACTER WAS FOUND IN THIS STATEMENT. THE COMPILER WILL SKIP TO THE NEXT SEMICOLON.

Explanation: The compiler has found an illegal punch or a character not in the legal character set. The error could be a misplaced control card or the absence of a $MACRO control card.

E27 A PARAMETER IS IN THE FORM OF A DATA CONSTANT. PARAMETERS MUST BE SIMPLE DATA NAMES.

Explanation: A parameter is written as a data constant. Parameters must be simple data names. (Arguments may be data constants.) For example:

```
A:  PROC (X, Y, 3); /* 3 IS ILLEGAL */
```

E28 VARIABLE HAS AN INCORRECTLY WRITTEN SUBSCRIPT, OR SHOULD NOT BE SUBSCRIPTED.

Explanation: The form of the subscript is incorrect. For example:

```
A=B(I+J*K+1); /*MULT OPERATOR MUST BE
                 LAST IN SUBSCRIPT*/
```

E29 THE 'NOT' OPERATOR IS NOT USED WITH A RELATIONAL OPERATOR.

Explanation: The ¬ operator may only be used in combination with one of the relational operators (¬=,¬<, or ¬>). For example:

```
A = ¬B; /* ILLEGAL OPERATOR */
```

E30 VARIABLE IS NOT QUALIFIED BY A POINTER. QUALIFIER IS MISSING OR HAS NOT BEEN DECLARED A POINTER.

Explanation: Self-explanatory. For example:

```
DCL A BASED;
A = 3; /* A HAS NO QUALIFIER */
B ->A = 3;
    /*B WAS NOT DECLARED POINTER*/
```

E31 CONSECUTIVE VARIABLES OR CONSTANTS APPEAR.

Explanation: Consecutive data variables or constants were found. Operators or delimiters may be missing. For example:

```
A = B C + D; /* ERROR */
A = 1 B; /* ERROR */
```

E32 VARIABLE HAS NOT BEEN DECLARED BASED OR BASED ON A POINTER, BUT IS BEING LOCATED BY A POINTER.

Explanation: A pointer qualifier is used to locate an item that has not been declared BASED or is not BASED on a pointer. For example:

```
DCL B FIXED(31);
DCL A BASED(ADDR(B));
B = P -> A; /* ILLEGAL */
```

E33 THE RIGHT SIDE OF A RELATIONAL EXPRESSION IS SHORTER THAN THE LEFT SIDE. THIS IS NOT ALLOWED WITH STRING DATA.

Explanation: Self-explanatory. For example:

```
DCL A CHAR(4), B CHAR(2);
IF A = B THEN GOTO L1;
    /* B IS SHORTER THAN A */
```

E34 ILLEGAL BIT OPERATION. CONSULT 'BSL USER'S GUIDE'.

Explanation: A bit string constant is illegal (e.g., '1011A'B), or a bit operation violates one of the following restrictions:

- If a bit variable is assigned to a bit variable, each bit variable must be on a byte boundary and must be a multiple of eight bits.
- If a bit constant is assigned to a bit variable, and the bit variable is not on a byte boundary, then the assigned bit string must not cross two byte boundaries (it may cross one byte boundary). If the bit variable is aligned on a byte boundary, the length of the bit string is not restricted.
- A null string constant used in any context other than a simple assignment statement of the type A='';.
- If a comparison involves a bit variable that is not on a byte boundary or not a multiple of eight bits, then the bit variable must be:
  1. Less than eight bits.
  2. Entirely contained within the boundaries of one byte.
  3. On the left side of the comparison.
  4. Compared to a bit constant which is either all ones or all zeros.
  5. Compared by the = or ¬= operator.
- A bit string item that is not on a byte boundary cannot be used as a subscript or substring.
- A bit variable that is not on a byte boundary is used as an argument in a CALL statement.
- A bit string constant that is used as an argument in a CALL statement is not a multiple of eight bits.

E35 VARIABLE IS A FORMAL PARAMETER OF A CONTAINING PROCEDURE.

Explanation: The formal parameter referred to does not appear in the PROCEDURE statement of the current procedure. For example:

```
A:  PROC (X, Y, Z);
      .
      .
B:  PROC (L, M, N);
     N=X; /* ILLEGAL REFERENCE TO X */
```

E36 THE FORMAL PARAMETER IN THE ENTRY STATEMENT APPEARED IN A DIFFERENT POSITION IN THE PROCEDURE STATEMENT.

Explanation: The position of a formal parameter in an ENTRY statement is different from its position in the PROCEDURE statement (or different from its position in another ENTRY statement). For example:

```
X:  PROC (A, B, C);
Y:  ENTRY (L, M, A); /* A IN WRONG
                        POSITION */
Z:  ENTRY (A, B, L); /* L IN WRONG
                        POSITION */
```

E37 INVALID CONSTANT.

Explanation: A constant is invalid for one of the following reasons:

- A hexadecimal string constant contains illegal digits (e.g., '01ABXY'X).
- A decimal number contains digits other than 0, 1, 2, 3, 4, 5, 6, 7, 8, 9; or is larger than 2^{31} -1.
- A binary number is longer than 31 digits.
- A string constant has more than 53 positions.

E38 xxxxxxxx IS NOT A POINTER, LABEL, OR ENTRY, BUT IS THE OPERAND OF A GOTO OR RETURN TO STATEMENT.

Explanation: The target in a GOTO or RETURN TO statement is not a POINTER, LABEL, or ENTRY item. For example:

GOTO V; is illegal if V is FIXED.

E39 RELATIONAL EXPRESSION HAS A STRING CONSTANT ON THE LEFT OR A VARIABLE LENGTH SUBSTRING ON THE RIGHT.

Explanation: A relational expression is incorrect for one of the following reasons:

- A string constant is the first item on the left side of a relational expression. For example:

```
IF ('FF'X & A) = 10 THEN GOTO L1;
  /* ILLEGAL USE OF STRING
  CONSTANT */
```

- A variable length substring appears on the right side of a relational expression and the left side is a constant length. For example:

```
IF A = B(I:J) THEN GOTO L1;
  /* ILLEGAL SUBSTRING */
```

E40 AN OFFSET OF MORE THAN 4096 IS REQUIRED IN CODE GENERATED FOR THIS STATEMENT. CONSULT 'BSL USER'S GUIDE'.

Explanation: An addressing error occurred because the constant displacement was greater than 4095 bytes. Constant displacement is explained under "Addresses" in Section V of this manual.

E41 THE END STATEMENT FOR A DO STATEMENT HAD NOT BEEN ENCOUNTERED WHEN ANOTHER PROCEDURE STATEMENT WAS ENCOUNTERED.

Explanation: A DO statement was not properly closed. For example:

```
A:  RPOC;
       .
       .
       .
   DO;
       .
       .
       .
   B:  PROC;  /* DO WAS NOT CLOSED
              BEFORE START OF INTERNAL
              PROCEDURE */
```

E42 xxxxxxxx IS A LABEL OR ENTRY ITEM, BUT IS NOT BEING USED AS SUCH.

Explanation: The named item is an LABEL or ENTRY item, but is being used for an arithmetic or logical operation. For example:

```
DCL L1 LABEL;
A=B+L1;  /* ILLEGAL USE OF LABEL */
```

E43 AN ELSE WAS ENCOUNTERED WHERE THERE WAS NO IF STATEMENT NEEDING AN ELSE.

Explanation: There is an unmatched ELSE clause. For example:

```
IF A = B THEN X = Y;
   A = B + 1;
   ELSE A=B-1; /*NO IF FOR THIS ELSE*/
```

E44 xxxx LABEL(S) HAVE BEEN REFERENCED BUT NOT DEFINED. UNDEFINED LABELS ARE MARKED WITH A 'U' IN THE XREF LIST.

Explanation: A number of labels have been referred to in the program, but were not defined. These undefined labels are marked with a 'U' in the attribute and cross-reference table.

E45 END OF FILE OR SEPARATOR EXPECTED HERE.

Explanation: BSL statements were found after the logical end of the BSL program. For example:

```
A:  PROC;
     .
     .
     .
  END A;
  X=Y+3; /*EOF OR $$$ SHOULD BE HERE*/
```

E46 AN OPERATION OF MORE THAN 256 BYTES IS INDICATED. THE SUBSTRING NOTATION SHOULD BE USED TO BREAK UP THIS TASK.

Explanation: The coding requires an operation of more than 256 bytes. For example:

```
DCL (B,C) CHAR(500);
B=C;  /* TOO MANY BYTES.  BREAK UP
      OPERATION BY USING SUBSTRING
      NOTATION */
```

E47 xxxxxxxx IS A STRING LONGER THAN 4 BYTES BEING USED IN AN ARITHMETIC EXPRESSION.

Explanation: The named item is a string longer than four bytes; therefore, it cannot be used in an arithmetic expression. For example:

```
DCL A CHAR(6), (X, Y) FIXED;
X = A + Y; /* A IS TOO LONG */
```

E48 A COMPILER ERROR HAS OCCURRED. SUBMIT A TROUBLE REPORT TO DEPARTMENT D76, BUILDING 706, POUGHKEEPSIE, N.Y.

Explanation: Self-explanatory.

E49 AN ILLEGAL OR NONEXISTENT REGISTER IS BEING RESTRICTED OR RELEASED, OR THE STATEMENT IS INCORRECTLY WRITTEN.

Explanation: One of the following:

- The specified register is reserved for use by the compiler. For example:

```
RESTRICT(13); /*RESERVED
    REGISTER*/
```

- The specified register does not exist. For example:

```
RELEASE(17);
    /*NONEXISTENT REGISTER*/
```

- The statement has a syntax error. For example:

```
RESTRICT 8; /*SYNTAX ERROR*/
```

- The variable name specified has not been declared register. For example:

```
RESTRICT(X); /*X HAS NOT BEEN
    DECLARED REGISTER*/
```

E50 xxxxxxxx APPEARS BETWEEN TWO DELIMITERS THAT SHOULD BE ADJACENT. AN OPERATOR MAY BE MISSING.

Explanation: The named item appears between two delimiters that should be adjacent. For example:

```
LBL CALL X;  /*COLON MISSING*/
```

E51 xxxxxxxx HAS A BIT LENGTH WHICH IS NOT AN INTEGRAL NUMBER OF BYTES. LENGTH HAS BEEN TRUNCATED.

Explanation: The name that replaces xxxxxxxx is a bit variable substringed with variable bounds. Its upper bound is specified as the lower bound plus a constant. The length specified by the bounds is not a whole number of bytes. For example:

```
DCL B BIT(40);
 B(I:I+8)=X; /*ILLEGAL--SPECIFIES
                NINE BITS*/
 B(I:I+7)=X; /*LEGAL*/
```

Serious Error Messages

S01 xxxxxxxx IS A STRUCTURE WITH MORE THAN 255 ELEMENTS.

Explanation: The named item is a structure that contains more than 255 components. The structure named could be a minor structure or it could be the major structure.

S02 THE PROCEDURE OPTIONS ARE INCORRECTLY WRITTEN, OR ARE USED IMPROPERLY.

Explanation: One of the following errors was found in the procedure options:

- The syntax of the options is incorrect. For example:
  1. The options are in the wrong position -- they must follow any parameter list specified.
  2. The registers specified in the options CODEREG or DATAREG are not enclosed in parentheses.
  3. The options are not followed by a right parenthesis.
- An invalid register is specified for CODEREG or DATAREG.
- The same register is specified for CODEREG or DATAREG.
- CODEREG, DATAREG, or REENTRANT is specified on an internal procedure.
- SAVE and DONTSAVE are both specified for the same procedure.
- The register list for SAVE or DONTSAVE is incorrectly written or specifies a nonexistent register.

S03 ONE OF THE IF STATEMENTS IN THIS NEST CONTAINED AN ERROR CAUSING IMPROPER MATCHING OF IF'S AND ELSE'S.

Explanation: There are several IF statements which are nested, and at least one of them has an error. The ELSE statements which have been been processed up to this point will not correspond to the correct IF. (This message occurs only for a nest of IF statements. It corresponds to the E43 message which is given for single IF statements.) For example:

```
L1:  IF A=B THEN
L2:   IF C=D THEN
L3:     IF E=X|Y THEN  /* ERROR ON
                         THIS IF */
L4:       IF F=0 THEN FL=0;
          ELSE FL=1;    /* CLOSES L4 */
        ELSE FL=2;    /* CLOSES L2
                         BECAUSE L3
                         WAS FLUSHED */
      ELSE FL=3;    /* CLOSES L1 */
   ELSE FL=4;    /* S03 MESSAGE GIVEN
                    HERE BECAUSE THERE
                    ARE NO MORE IFS TO
                    CLOSE */
```

S04 xxxxxxxx IS A MINOR STRUCTURE WHICH HAS A TOTAL SIZE GREATER THAN 32,767 BYTES. THE ENTIRE STRUCTURE IS DELETED.

Explanation: The named item is a component of a structure that contains more than 32,767 bytes.

Compiler Action: The entire major structure is deleted. For example:

```
DCL 1 ST,
      2 MINOR,
        3 MINA CHAR(30000),
        3 MINB CHAR(4000),
      2 MINOR2;
    /* MINOR HAS A SIZE GREATER THAN
    32,767. THE MAJOR STRUCTURE, ST,
    IS DELETED */
```

S05 VARIABLE HAS ILLEGAL REGISTER ATTRIBUTE. REGISTER STORAGE CLASS HAS NOT BEEN USED FOR THIS VARIABLE.

Explanation: The keyword REGISTER is not followed by a left parenthesis, or the number within the parentheses is not a legal register.

Compiler Action: The REGISTER storage class is not used for this variable.

S06 VARIABLE HAS ILLEGAL BASED ATTRIBUTE. BASED ATTRIBUTE HAS NOT BEEN APPLIED TO THIS VARIABLE.

Explanation: One of the following errors has been found in the BASED attribute:

- The syntax of the ADDR function is incorrect. For example:

```
DCL BAS BASED(ADDR XXX);
  /* NO PARENTHESIS AFTER ADDR */
DCL BAS BASED(ADDR(XYZ);
  /* NO CLOSING PARENTHESIS */
DCL BAS BASED(ADDR(XXX+4));
  /* OFFSET SHOULD BE OUTSIDE THE
  PARENTHESIS */
```

- The offset from the base is not a decimal number, or is greater than 32,767 bytes. For example:

```
DCL A BASED(P+40000);
  /* ILLEGAL */
DCL B BASED(ADDR(XX)+40000);
  /* ILLEGAL */
```

- The absolute value given as the base is greater than 32,767.

Compiler Action: In all cases the variable is treated as a STATIC item. (Only the BASED attribute is ignored.)

S07 VARIABLE HAS AN ILLEGAL NAME IN THE BASED ATTRIBUTE.

Explanation: The name given as the base is not a legal name. For example:

```
DCL A BASED(12B);  /* ILLEGAL NAME */
DCL C BASED(ADDR(1B));
    /* ILLEGAL NAME */
```

S08 xxxxxxxx IS BASED ON AN UNDECLARED OR IMPROPERLY DECLARED ITEM.

Explanation: One of the following errors was found in the BASED name:

- The named item is based on a variable that has not been declared or has been declared other than pointer: For example:

```
DCL FLG BASED(PFLG);
 /* WHERE PFLG IS NOT DCL'D OR NOT
 PTR */
```

- The variable in the ADDR function has not been declared. For example:

```
DCL CODE BASED(ADDR(FIELD));
  /* WHERE FIELD HAS NOT BEEN
  DECLARED */
```

- The named item is based on a dimensioned variable. For example:

```
DCL NMPTS(10) PTR;
DCL XPT BASED(NMPTS);
  /* ILLEGAL BECAUSE NMPTS IS A
  DIMENSIONED VARIABLE */
```

S09 MORE THAN 50 ITEMS HAVE BEEN FACTORED. A RIGHT PARENTHESIS WAS ASSUMED AFTER THE 50TH ITEM.

Explanation: More than 50 variables appear inside parentheses as factored items.

Compiler action: The statement is processed as if there were a right parenthesis after the 50th item.

S10 VARIABLE IS BASED ON THE ADDRESS OF A REGISTER.

Explanation: A variable is declared BASED, using the ADDR function, and the name in the ADDR function is the name of a register variable. For example:

```
DCL R1 REG(1);
DCL VBL BASED(ADDR(R1));
    /* ILLEGAL USE OF REGISTER VARI-
    ABLE */
```

S11 xxxxxxxx HAS MORE THAN ONE DIMENSION. ONLY THE FIRST DIMENSION HAS BEEN ACCEPTED.

Explanation: The named item is an element of a dimensioned structure, and the item itself is also dimensioned. For example:

```
DCL 1 CDIMG(10);
      2 NMS CHAR(20),
      2 CDS(5); /*ILLEGAL DIMENSION*/
```

Compiler Action: Only the dimension on the containing structure is accepted.

S12 xxxxxxxx IS MORE THAN 32,767 BYTES FROM THE BEGINNING OF A STRUCTURE. THE STRUCTURE IS DELETED.

Explanation: The named item is a component of a structure, and is offset more than 32,767 from the start of the structure. For example:

```
DCL 1 COM,
      2 AREA CHAR(32767),
      2 IND CHAR(1); /* OFFSET GREATER
                  THAN 32767 */
```

Compiler Action: The entire major structure is deleted.

S13 NO REGISTER IS AVAILABLE FOR ADDRESSING EXTERNAL DATA.

Explanation: A register is needed to obtain the address of an external item, but there are no registers available to the compiler.

S14 NO REGISTERS ARE AVAILABLE FOR CALCULATING SUBSCRIPTS OR SUBSTRINGS.

Explanation: A register is needed to calculate an index, but there are no registers available for the compiler's use.

S15 NOT ENOUGH REGISTERS ARE AVAILABLE FOR THIS COMPUTATION.

Explanation: Not enough registers are available to generate code for a statement that needs a number of registers. For example:

```
DCL CRSTR CHAR(10);
DCL XREC(5) EXTERNAL CHAR(50);
XREC(L,I:J) = CRSTR(K:10);
```

needs at least four registers to address XREC: one for the ADCON, one for the subscript, one for the substring, and one to do an EXECUTE of an MVC instruction.

S16 NO EVEN/ODD PAIR OF REGISTERS IS AVAILABLE TO DO A MULTIPLICATION THAT IS PART OF A SUBSCRIPT CALCULATION.

Explanation: The compiler needs an even/odd pair of registers to do a multiplication that is part of a subscript calculation. No such pair of registers is available.

S17 VARIABLE FOLLOWING "END" DOES NOT MATCH THE LABEL ON ANY OPEN DO STATEMENT.

Explanation: The name given on the END statement does not correspond to the label on any previous DO statements.

Compiler Action: All open DO statements will be closed.

S18 THE STATEMENT IS TOO LONG. A STRING CONSTANT MAY HAVE A MISSING QUOTE. THE STATEMENT HAS NOT BEEN COMPILED.

Explanation: The statement is too long, for one of the following reasons:

- A DECLARE statement is too long for the compiler to process. (The compiler can handle approximately 1000 characters, not counting blanks.)
- A string constant does not have a closing quote, in which case all statements following the string are taken as part of the string. (One indication of this is that no statement numbers will be printed for the statements that follow the string.)

S19 A PROCEDURE OR ENTRY STATEMENT IS INCORRECTLY WRITTEN.

Explanation: One of the following errors was found in a PROCEDURE or ENTRY statement:

- The syntax of the statement is incorrect. For example:
  1. The keyword is not preceded by a name.
  2. More than one name precedes the keyword.

- The procedure name is declared as other than ENTRY.
- An ENTRY statement has parameters, but the procedure it is contained in has no parameters.

S20 MORE THAN 75 UNIQUE DECIMAL CONSTANTS, OR MORE THAN 75 NONLOCAL EXTERNAL ITEMS HAVE BEEN USED.

<u>Explanation</u>: The compiler can handle a maximum of 75 unique decimal constants, or 75 NONLOCAL EXTERNAL items, in a single compilation.

S21 THERE ARE MORE THAN SEVEN IMPLICIT OR SEVEN EXPLICIT POINTERS IN A CHAIN.

<u>Explanation</u>: A statement has more than seven levels of one type of pointer. (With a combination of both types, a statement may have up to 14 pointers.) For example:

```
P1->P2->P3->P4->P5->P6->P7->P8->X = 0;
```

is a string of explicit pointers, and is illegal because there are more than seven. The next example:

```
DCL P1 PTR, P2 PTR BASED(P1),
    P3 PTR BASED(P2), P4 PTR
    BASED(P3),
    P5 PTR BASED(P4), P6 PTR
    BASED(P5),
    P7 PTR BASED(P6), P8 PTR
    BASED(P7),
    X BASED(P8);
```

contains implicit pointers. To locate X implicitly, more than seven pointers are required. Thus, the assignment X = 0; would be illegal. The next example:

```
P7->PA->PB->X = 0;
```

involves more than seven pointers, but is legal because there is a combination of implicit and explicit pointers.

<u>Disastrous Error Messages</u>

D01 PROGRAM HAS TOO LARGE A DATA AREA. CONSULT 'BSL USER'S GUIDE'.

<u>Explanation</u>: A data item is more than 32,767 bytes from the start of the data area. This problem involves the offset from the beginning of the data area, not the total size. Once 32,767 bytes of data are declared, no additional data can be declared. For example:

```
DCL INTERFAC CHAR(32767);
DCL CODE CHAR(2); /* ILLEGAL */
```

If the above example is reversed, the declarations are legal. For example:

```
DCL CODE CHAR(2);
DCL INTERFAC CHAR(32767);
     /* LEGAL BUT NO DATA CAN BE
     DECLARED AFTER INTERFAC */
```

D02 THE PROGRAM IS TOO LARGE. THE DICTIONARY SPACE HAS BEEN FILLED. SEE SIZE OPTION IN 'BSL USER'S GUIDE'.

<u>Explanation</u>: The number of data items in the program is more than can be handled in the compiler's dictionary space.

<u>User Response</u>: Use the SIZE option to specify a larger dictionary (OS only), or remove some data items from the program.

D03 THE PROGRAM IS TOO LARGE. THE SPACE ASSIGNED FOR DICTIONARY, INITIAL VALUES, AND CROSS REFERENCE TABLES IS FILLED.

<u>Explanation</u>: The program is too large because of the number of data items, the number and size of initial values, and the number of references to the data items.

<u>User Response</u>: One or more of the following:

- Use the SIZE option to specify a larger dictionary (OS only).
- Use the NOXREF option to suppress the attribute and cross-reference table.
- Reduce the number or size of initial values by initializing the items dynamically.

D04 TOO MANY STRING CONSTANTS HAVE BEEN USED.

<u>Explanation</u>: The total number of characters in string constants (hexadecimal, bit, and character) is too large. There are about 1400 bytes available to hold these constants, and the compiler does eliminate duplicate strings.

<u>User Response</u>: Change the constant to a variable and initialize it.

D05 THE END OF AN INTERNAL PROCEDURE IS NOT FOLLOWED BY THE END FOR THE CONTAINING PROCEDURE OR A NEW INTERNAL PROCEDURE.

Explanation: The END statement of an internal procedure was not followed by another internal procedure or the END statement for the external procedure.

D06 THERE ARE MORE THAN 14 INTERNAL PROCEDURES.

Explanation: The total number of procedures that can be handled in one compilation is 15, including the external procedure. It does not matter if the internal procedures are nested or disjoint.

D07 THERE ARE MORE THAN 8 DO STATEMENTS IN A NEST.

Explanation: More than eight DO statements are open at the same time.

User Response: Break up the nest of DO statements.

D08 THERE ARE MORE THAN 14 IF STATEMENTS IN AN IF NEST.

Explanation: More than 14 IF statements are open at the same time.

User Response: Break up the nest of IF statements. Branch out on one of the IF statements, and continue with another IF nest at the branch point.

D09 THERE ARE MORE THAN 50 TRUE/FALSE BRANCHES IN A NEST OF IF STATEMENTS.

Explanation: The total number of true/false branches in a nest of IF statements is the sum of the number of IF statements and the number of logical connectives in the IF statements. This number cannot be greater than 50.

User Response: Simplify the nest of IF statements.

D10 THREE CONTROL CHARACTERS OR AN END OF FILE HAS BEEN ENCOUNTERED BEFORE ALL PROCEDURES HAVE BEEN CLOSED.

Explanation: At least one procedure is open, and three control characters or end of file has been read.

This error could be caused by a DO statement which does not have a corresponding END statement. The compiler would assume that the procedure's END statement is the close for the DO, and then look for another END statement.

D11 AN INCLUDE OPTION APPEARS WITHIN INCLUDED TEXT.

Explanation: Included text may not contain an INCLUDE option.

D12 THE INCLUDE OPTION IS INCORRECTLY WRITTEN.

Explanation: The ddname or member name is too long.

D13 THE INCLUDE OPTION ASKS FOR A LIBRARY MEMBER THAT CANNOT BE FOUND IN THE SPECIFIED LIBRARY.

Explanation: The member name specified in the INCLUDE option cannot be found in the specified partitioned data set.

User Response: Check the spelling of the member name in the INCLUDE statement, and check the data set name on the DD statement indicated in the INCLUDE option.

D14 AN I/O ERROR HAS OCCURRED WHILE SEARCHING FOR THE REQUESTED MEMBER IN THE USER'S LIBRARY.

Explanation: I/O error.

D15 THE PARTITIONED DATA SET SPECIFIED IN THE INCLUDE OPTION HAS A BLKSIZE GREATER THAN 3520 OR A RECFM OTHER THAN F.

Explanation: The partitioned data set may contain blocked records up to a block size of 3520 bytes, and must contain fixed format records.

User Response: Respecify the block size or record format of the data set and run the job again. Make sure that the INCLUDE option specifies the correct data set.

D16 MORE THAN 2559 COMPILER GENERATED LABELS HAVE BEEN USED.

Explanation: This is the maximum count of labels generated for DO statements, IF statements, and CALL statements. The count does not include any labels on data.

User Response: Reduce the number of IF, DO, and CALL statements.

D17 COMPILATION HAS BEEN TERMINATED BECAUSE OF SOURCE ERRORS THAT HAVE OCCURRED PREVIOUSLY. FIX ERRORS AND RERUN.

Explanation: The compilation cannot continue because of the errors that occurred previously.

User Response: Correct the errors and recompile.

D18 A COMPILER ERROR HAS OCCURRED. SUBMIT A TROUBLE REPORT TO DEPARTMENT D76, BUILDING 706, POUGHKEEPSIE, N.Y.

Explanation: Self-explanatory.

D19 INCORRECT INVOCATION OF BSL COMPILER

Explanation: A compiler error has occurred while trying to process the parameters passed to the compiler when it is dynamically invoked. The following kinds of errors could cause a compiler error:

- Register 1 contains an invalid address.
- Register 1 contains an address that is not on a fullword boundary.
- The address parameters to which register 1 points are not valid addresses.
- The second address parameter (list of alternate ddnames) has been omitted and the high order bit of the first address has not been set to one.
- No compiler options or alternate ddnames have been specified, and the high order bit of the first address has not been set to one, or the first address parameter has been omitted, or the first address parameter does not point to a halfword of zeros on a halfword boundary.
- One or both address parameters contain an address that is not on a halfword boundary.

User Response: Correct the calling sequence for the dynamic invocation of the BSL compiler. If the problem persists, submit a trouble report to Department D76, Building 706, Poughkeepsie, New York.

D20 THREE CONTROL CHARACTERS OR AN END OF FILE APPEARS WITHIN A BLOCK GENERATE.

Explanation: Three control characters or an end of file has been read while BSL source input is still being processed as the text statements of a block GENERATE. The corresponding $ENDGEN control statement for the block GENERATE is missing.

D21 THREE CONTROL CHARACTERS OR AN END OF FILE APPEARS WITHIN A COMMENT.

Explanation: Three control characters or an end of file has been read before the comment terminator (*/) of a comment was found. An indication of this error is that the statements following the comment will not have statement numbers.

D22 THREE CONTROL CHARACTERS OR AN END OF FILE APPEARS WITHIN A STRING CONSTANT.

Explanation: Three control characters or an end of file has been read before the closing quote of a string constant. An indication of this error is that the statements following the string constant will not have statement numbers.

Index

READER'S COMMENT FORM

BSL User's Guide

Form Z28-6682-2

- Is the material:

	Yes	No
Easy to read?	□	□
Well organized?	□	□
Complete?	□	□
Well illustrated?	□	□
Accurate?	□	□
Suitable for its intended audience?	□	□

- How did you use this publication?
  - □ As an introduction to the subject
  - □ For additional knowledge

  Other ....

- Please check the items that describe your position:
  - □ Customer personnel
  - □ IBM personnel
  - □ Manager
  - □ Systems Analyst
  - □ Operator
  - □ Programmer
  - □ Customer Engineer
  - □ Instructor
  - □ Sales Representative
  - □ Systems Engineer
  - □ Trainee

  Other ....

- Please check specific criticism(s), give page number(s), and explain below:
  - □ Clarification on page(s) ....
  - □ Addition on page(s) ....
  - □ Deletion on page(s) ....
  - □ Error on page(s) ....

Explanation:

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold

Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold

Fold