GC28 - 6786 - 0

Systems

Guide to PL/S-Generated Listings

IBM

First Edition (July, 1972)

Changes will appear in new editions or Technical Newsletters. The RETAIN/370 System will be used to notify the field of new editions or newsletters.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

Preface

Purpose of Guide

The PL/S (Programming Language/Systems) compiler is a proprietary program used by IBM to develop other programs that are made generally available. The PL/S compiler is not available outside IBM. Programs written by IBM in PL/S are documented by means of listings in microfiche form. The Guide to PL/S-Generated Listings provides general information on reading and interpreting these listings. The book also provides some guidelines on how to modify compiler-generated assembler code. However, the specifications of PL/S and the style of assembler code generated are subject to change in the interest of improving IBM programs.

This guide does not contain information on writing and compiling PL/S source programs. Furthermore, it does not list the assembler code generated for each PL/S statement; the large number of possible combinations of source language elements makes such a list impractical.

Users of Guide

Readers should be experienced systems programmers who have this background:

- They know the basic assembler language.
- They are familiar with a higher level language such as FORTRAN, COBOL, or preferably PL/1 (which PL/S closely resembles).

The general knowledge of PL/S obtained from this guide should assist these programmers in interpreting PL/S program listings. They may find it easier to understand what a system module does by reading the PL/S statements rather than by reading the more detailed assembler language instructions.

Format of Guide

The guide is organized into the following sections:

Section 1, the Introduction, provides an overview of the PL/S language, the compiler, and the output produced by a compilation.

Section 2, the PL/S Language, describes the purpose and format of PL/S source statements and built-in functions.

Section 3, Compiler Output, describes the compiler-generated code and information listings produced by the compiler.

Section 4, Guidelines for Code Modification, lists some guidelines for consideration when modifying the compiler-generated assembler code.

Section 5, the Glossary, defines terms associated with PL/S.

Contents

Figures

Section 1: Introduction

Programming Language/Systems (PL/S) is a language designed for IBM systems programmers. It is related to the higher level languages such as FORTRAN, COBOL, and particularly PL/I.

PL/S is designed to express operations used in systems programs. One such operation is the storing and retrieving of information in tables. In assembler language many instructions are usually required to express these table operations. With PL/S, a table can be defined and its elements utilized with fewer statements. Because PL/S is more compact and English-like than assembler code, PL/S source programs can be understood by the reader faster than equivalent assembler programs.

PL/S also allows assembler statements to be used in a PL/S program. The PL/S statement GENERATE (abbreviated GEN) marks such insertions. GEN DATA marks assembler data definitions; the compiler places this data in the data area it creates.

The PL/S Compiler

PL/S language statements are grouped into a source program called a procedure. A procedure is converted to object code through successive steps of compilation and assembly. These two translation steps can be summarized as follows:

1. The PL/S compiler translates the PL/S source language statements into assembler language instructions suitable for input to a System/360 assembler. Several assembler language instructions usually result from a single PL/S statement.

2. A System/360 assembler program accepts as its input the compiler-generated assembler instructions and translates them into an object module, which is link edited in the normal manner.

Figure 1 provides an overview of the PL/S translation process.

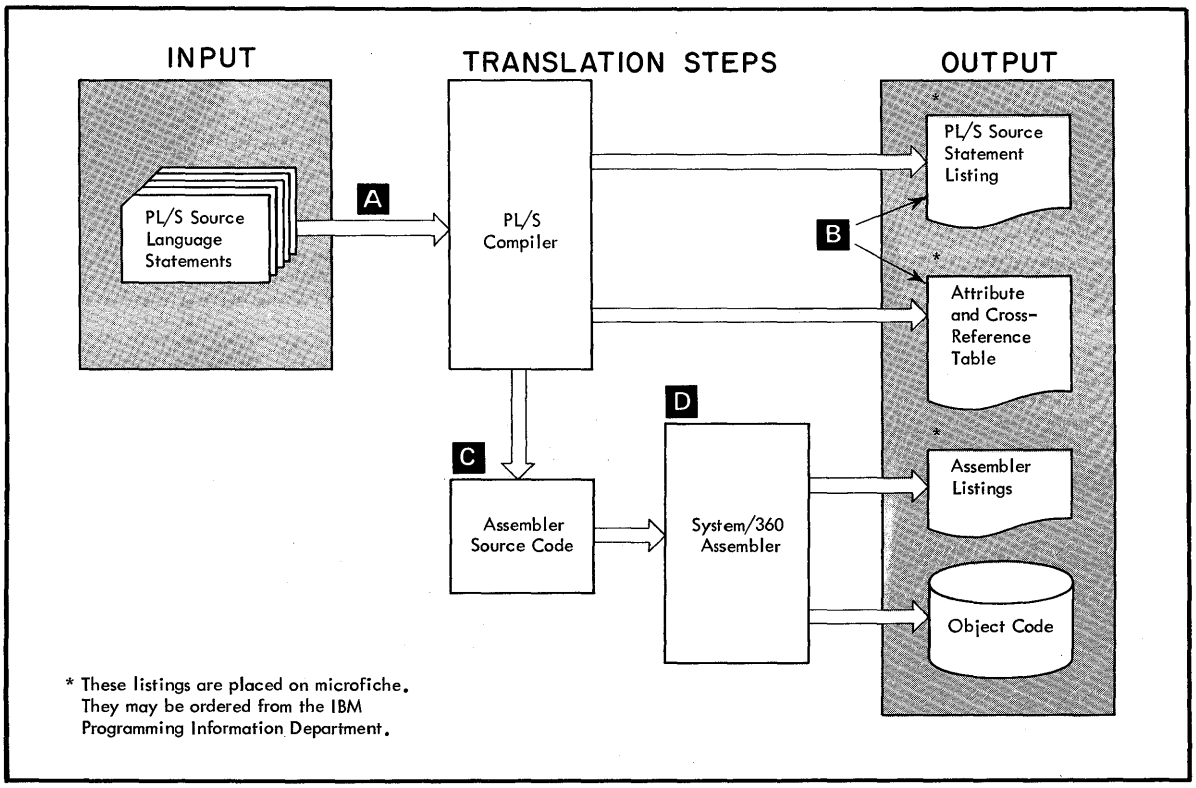


Figure 1. Overview of PL/S Translation Process

A The PL/S language statements are input to the PL/S compiler.

B The compiler produces the PL/S source statement listing and the PL/S attribute and cross-reference table.

C The generated assembler language instructions, containing PL/S statements interspersed as assembler comments, are input to the assembler.

D The System/360 assembler translates the compiler-generated assembler language instructions into object code. The assembler also produces the assembler listing.

Code Modifications

If you are considering modifications to your operating system, you can order the assembler source modules in machine readable form from the IBM Programming Information Department. After making changes to the assembler source code, you can assemble and link edit the modules into the system.

Index to PL/S Keywords

PL/S statements appear on one or more lines; they are terminated by a semicolon.

Executable statements may start with one or more labels, which are separated from the statement and from each other by colons. The following is an example of a labeled statement:

```
LABEL1:
      A = B+C;
```

PL/S comments are delimited by the symbols /* and */. For example:

```
/* THIS IS A COMMENT */
```

PL/S Procedures

PL/S programs are divided into external and internal procedures.

An external procedure, after compilation and assembly, is one assembler CSECT. Internal procedures are subdivisions of external procedures; they are wholly contained within external procedures.

All procedures begin with a PROCEDURE statement (abbreviated PROC) and end with an END statement. (You may find a GENERATE statement, but no others, before the PROC of an external procedure.) The label that precedes the PROC keyword is the name of the procedure, and is its primary entry point. For external procedures, this name is the object module name that you will find on microfiche cards and listings.

Parameters and options often follow the PROC keyword. Parameters are a means of communicating from one procedure to another. They are a list of variables - enclosed in parentheses and separated by commas - that immediately follow the PROC keyword. Options affect the way the compiler produces code for the procedure. They are a list of keywords, enclosed in parentheses, that follows the OPTIONS keyword.

The sample PROCEDURE statement below has two parameters and one option:

```
IKJEFF01:PROC   (A,B)   OPTIONS(REENTRANT) ;
```

SAVING REGISTERS ACROSS PROCEDURES

The assembler code produced for procedures follows standard linkage conventions. Figure 2 shows the registers used for the standard linkage functions.

Register	Function
15	Contains the address of the entry point in the called procedure.
14	Contains the address of the return point in the calling procedure.
13	Contains the address of the calling procedure's save area.
1	Contains the address of a parameter list, if arguments are passed to the called procedure.

Figure 2. Linkage Registers

Every procedure provides a save area to preserve its registers unless the NOSAVEAREA option appears on its PROC statement. The format of this area is shown in Figure 3.

Word	Contents
1	Not used.
2	Address of calling procedure's save area.
3	Address of called procedure's save area.
4	Register 14
5	Register 15
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

Figure 3. Save Area Format

All of the registers shown in Figure 3, and register 13, are saved on entry to a procedure and restored on exit from a procedure, unless the SAVE or DONTSAVE option appears on the PROC statement. SAVE is followed by an explicit list of registers to be saved and restored by the procedure; DONTSAVE is followed by a list of those that are not to be saved and restored.

COMPILER REGISTER ASSIGNMENTS

As the PL/S compliler produces assembler code, it assigns registers to the functions shown in Figure 4.

Register	Function
0	Used for some arithmetic calculations.
1	Parameter list pointer.
10	Used for data moves.
11	Base register for addressing code.
13	Save area pointer.
14	Used to contain a return address. Also used for arithmetic calculations and data moves.
15	Used to contain an entry point address. Also used for arithmetic calculations.
All other registers	Used for indexing, external data addressing, and pointer manipulation.

Figure 4. Compiler-Assigned Functions for Registers

These standard assignments apply unless modified by the PROC statement options REENTRANT, CODEREG, and DATAREG.

REENTRANT OPTION

This option, which you may find on the PROC statement for an external procedure, tells the compiler to provide for reentrant code for the external procedure and all procedures internal to it. The compiler produces code to obtain a storage area dynamically for the external procedure and its internal procedures on entry to the external procedure. This storage area contains:

- save areas
- data defined under a GENERATE DATA statement
- temporary storage used by the compiler
- data declared in the procedure, except data with the STATIC or INITIAL attributes

The compiler maps this area into a DSECT labeled @DATD.

Reentrant code requires separate base registers for addressing the data in the dynamic area and for addressing code. Register 11 is used for code addressing and register 12 is used for addressing the data in the dynamic area. These register assignments will be in effect unless the CODEREG or DATAREG options were used to change them.

CODEREG AND DATAREG OPTIONS

The CODEREG option is followed by one or more register numbers. These registers replace register 11 as the base register for code addressing. CODEREG(0) tells the compiler not to establish addressability - it is not needed or is provided by a GENERATE statement.

The DATAREG option is followed by one or more registers to be used as the base registers for addressing data. The combination DATAREG(0) with REENTRANT tells the compiler not to obtain a dynamic storage area, and not to establish addressability for data declared with the AUTOMATIC attribute.

TRANSFERRING CONTROL BETWEEN PROCEDURES

Control flow between procedures is accomplished by the CALL, RETURN, and END statements. The CALL keyword is followed by the label of the statement that receives control. This label is in an external procedure or in a procedure that is internal to the calling procedure. It is always the label of a PROCEDURE statement, which is the primary entry point of a procedure, or of an ENTRY statement, which defines a secondary entry point.

Control returns to the statement immediately following the CALL when execution reaches either a RETURN statement or an END statement that matches a PROCEDURE statement. A RETURN TO statement sends control to a return point specified on the statement; the return point will usually be in the calling procedure.

COMMUNICATION BETWEEN PROCEDURES

A calling procedure communicates with a called procedure by means of an argument list on the CALL statement. This list appears, in parentheses, following the entry point label. It may contain single variables, expressions, and constants.

The compiler creates an argument list that has one word for each argument; an address is inserted in each word. The address inserted depends on the type of argument, as shown in Figure 5.

If the argument is:	The argument list address is:
A variable, not in parentheses.	The address of a variable.
A constant, not in parentheses.	The address of the constant.
A variable or constant in parentheses.	The address of a temporary variable that contains a copy of the variable or constant.
An expression	The address of a temporary variable that contains the result of evaluating the expression.

Figure 5. Argument List Contents

The high-order bit in the last word of the argument list will be set on if the DECLARE statement for the entry point of the called procedure contains the keywords OPTIONS(VLIST). The bit indicates the end of a variable length argument list.

The called procedure will receive control at a PROCEDURE or ENTRY statement. These statements have parameter lists, and the parameters in them correspond positionally to the arguments on the CALL statement. Since the correspondence is positional, the names used for an argument and its associated parameter may not be identical.

When a called procedure returns control by means of a RETURN statement, it may pass back a value that is obtained from a variable, an expression, or a constant which follows the CODE keyword on the RETURN statement. The value is returned to the calling procedure in register 15.

Data Definitions

The attributes of data are described in DECLARE statements (abbreviated DCL). These statements start with the DCL keyword, followed by the data item's name, followed by the keywords that define the data item's attributes. Since many attributes are defined by default, check the data item's description in the Attribute and Cross-Reference listing for a complete list of explicit and default attributes of each data item.

A single DCL statement frequently defines multiple data items. Each declaration is separated from the next by a comma. For example:

```
DCL  A  POINTER(31),  AREA1 CHAR(12), AREA2 CHAR(12);
```

This example is a declaration of three data items - A, AREA1, and AREA2. Because AREA1 and AREA2 share a common attribute - CHAR(12) - the statement would normally appear in this form, which is equivalent to the preceding example:

```
DCL  A  POINTER(31),  (AREA1,AREA2)  CHAR(12);
```

When attributes follow data items that are in parentheses, the attributes apply to all of the data items that are in the parentheses. If a data item has unique attributes, the unique ones appear after the data item within the parentheses. For example:

```
DCL  A  POINTER(31), (AREA1 INIT('ABC'), AREA2)  CHAR(12);
```

The attribute INIT('ABC') applies to AREA1 only; CHAR(12) applies to both AREA1 and AREA2.

DATA TYPES

The DCL statement defines four types of data - arithmetic, string, pointer, and label.

Identifying Data Types

Arithmetic data is interpreted as a binary, fixed-point integer; it is identified by the keyword FIXED. This keyword may be followed by a number, in parentheses, which is the precision of the data, expressed in terms of bits. (Precision determines how many bytes will be assigned to contain the data).

String data is a sequence of bytes or a sequence of bits. Character strings are identified by the keyword CHARACTER (abbreviated CHAR) followed, in parentheses, by the number of bytes in the sequence. Bit strings are identified by the keyword BIT followed, in parentheses, by the number of bits in the sequence.

The keyword POINTER (abbreviated PTR) identifies data that is interpreted as the address of other data. This keyword may have a precision following it. This precision is expressed in terms of bits.

Labels are identified by either the ENTRY or LABEL keyword. A label declared with ENTRY is the address of a PROCEDURE or ENTRY statement; labels of other statements are declared with the LABEL keyword. (Often labels are not explicitly declared.)

INITIALIZATION

The INITIAL attribute (abbreviated INIT) is the means of initializing a data item at program load time. This attribute is followed by a constant or an expression involving the ADDR built-in function. PL/S has five types of constants - decimal, hexadecimal, character, bit, and binary. The general form of each is shown in Figure 6.

Constant Type	Format
Decimal	decimal digits
Hexadecimal	' any hex digits 'X
Binary	zeroes and ones
Bit	' zeroes and ones 'B
Character	' any EBCDIC characters '

Figure 6. Form of PL/S Constants

BOUNDARY ALIGNMENT

The purpose of the BOUNDARY attribute is to provide an explicit boundary alignment for a data item. This attribute (abbreviated BDY) is followed by the keyword BYTE, HWORD, WORD, or DWORD, corresponding to byte, halfword, fullword, and doubleword. These keywords, in turn, may be followed by a decimal number that indicates the starting byte position within the boundary. The digit 1 indicates the left-most byte.

WHERE DATA RESIDES

PL/S variables are either areas of main storage or registers. When they reside in main storage, they are assigned storage by one procedure but they may be used by others. The assigned storage may be in the CSECT of the assigning procedure or in a dynamic storage area.

Data in Registers

A register variable is identified by the keyword REGISTER (abbreviated REG) on its DCL statement. This attribute is followed by the number of the general purpose register used for the variable.

The attribute RESTRICTED is used with REGISTER to prevent the compiler from using the specified register in assembler instructions that it produces. If RESTRICTED does not appear on the DCL statement, or if the UNRESTRICTED attribute appears, then the compiler is free to use the register.

The RESTRICTED attribute does not reserve the register outside of the declaring procedure (either internal or external). And within the declaring procedure the register may be released for compiler use at any point by a RELEASE or RESPECIFY statement. The RELEASE statement consists of the keyword RELEASE followed by the register numbers or the variable names of the registers to be freed. The RESPECIFY statement, when used to release a register, consists of the keyword RESPECIFY, one or more register numbers or variable names, and the keyword UNRESTRICTED.

Similarly, an unrestricted register may be restricted at any point by a RESPECIFY statement that contains the keyword RESTRICTED, or by a RESTRICT statement.

Note on Register Restriction: Restriction applies to a physical register and, therefore, to all symbolic names by which the physical register is known.

Data in Main Storage

Data declared with the STATIC attribute is assigned storage in a fixed area. The AUTOMATIC attribute (abbreviated AUTO) causes the data to be assigned in a dynamically acquired area, which the compiler maps in a DSECT labeled @DATD.

Although the STATIC attribute causes data to be assigned in a fixed area, it does not specify which CSECT the fixed area is in. The LOCAL attribute does that. Data declared with the LOCAL attribute is assigned storage in the CSECT of the declaring procedure. A DCL statement with the NONLOCAL attribute means that the data is assigned storage in a procedure other than the declaring one.

DATA REFERENCES ACROSS PROCEDURES

A variable declared with the keyword INTERNAL (abbreviated INT) can be referenced in the declaring procedure and in any procedure internal to it. When a variable is referenced in two or more external procedures, it will be declared with the EXTERNAL attribute (abbreviated EXT) in each procedure.

The compiler produces an assembler language EXTRN instruction and an A-type address constant for data items declared NONLOCAL EXTERNAL. Branch points declared NONLOCAL EXTERNAL produce a V-type address constant. Items declared LOCAL EXTERNAL cause the compiler to produce an assembler ENTRY instruction.

INDIRECT ADDRESSING

Storage is assigned to variables declared STATIC or AUTOMATIC, but none is assigned when a variable is declared with the BASED attribute. A DCL with BASED simply defines a set of attributes. These attributes are applied to a storage area specified by a locator address. This locator may be specified following the BASED keyword. For example:

```
DCL  P  PTR;

DCL  B  CHAR(4) BASED(P);
```

When B is referenced, the four bytes starting at the address contained in P are used. These four bytes are considered to be a character string variable.

The BASED keyword is not always followed by a locator, but before the variable is used, a locator will be supplied. PL/S has two facilities for supplying the locator - pointer notation and the RESPECIFY statement. (If the BASED keyword does supply a locator, these same facilities can override it).

Pointer notation has the general form:

```
pointer variable -> BASED variable
```

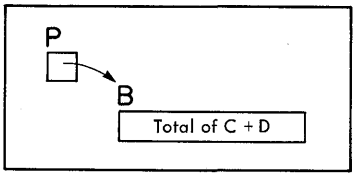
For example,

```
P -> B
```

means that a reference to BASED variable B is a reference to the storage area that starts at the address contained in pointer variable P. The statement

```
P -> B = C + D ;
```

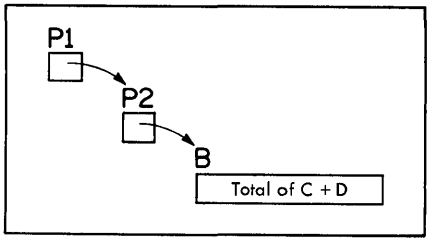
causes this:



Multiple levels of pointer notation are also possible. For example:

```
P1 -> P2 -> B = C + D ;
```

causes this:



Pointer notation supplies or changes a locator only temporarily; a locator will be supplied again before or upon subsequent references to the BASED variable. If pointer notation overrides a previous locator, then the previous locator will be used for subsequent references to the BASED variable.

The other facility for supplying or changing a locator, the RESPECIFY statement, has the form:

```
RESPECIFY (one or more BASED variable names) BASED(pointer variable);
```

The specified pointer variable will be used to locate the specified variable(s). Unlike pointer notation, the RESPECIFY statement has more than temporary effect. The new locator will be used until the end of the current procedure, unless pointer notation or another RESPECIFY statement changes it.

ARRAYS

An array is a collection of variables (called elements) that have identical attributes and that occupy a contiguous storage area; the collection has a common name.

A DCL statement defines an array if the variable name is followed immediately by a decimal number in parentheses. This number is the dimension of the array, i.e., the number of elements in it.

Attributes on a DCL statement for an array apply to all elements. However, the INITIAL attribute can initialize each element individually. For example, the statement

```
DCL   ARY(5)  FIXED(31)  INIT(0,4,8,12,16);
```

defines a five-element array of fullword arithmetic variables that is initialized like this:

0
4
8
12
16

When an asterisk appears in place of an initializing value, the corresponding element is not initialized. Multiple elements are initialized when a replication number appears, in parentheses, before an initial value. For example, this statement:

```
DCL   ARY(5)  FIXED(31)  INIT((3)0,12,16);
```

defines this:

0
0
0
12
16

If the INITIAL attribute does not specify enough values to initialize all array elements, the last elements are uninitialized. If INITIAL provides too many values, the last values are ignored.

STRUCTURES

A structure is a data collection that is divided into individually named components. The entire collection can be referenced by the structure name, or a component can be referenced individually by its name. Although all components can have the same attributes, they are usually assigned unlike attributes.

The DCL statement for a structure defines how components map into the structure, and defines the attributes that apply to the structure and its components. This DCL statement

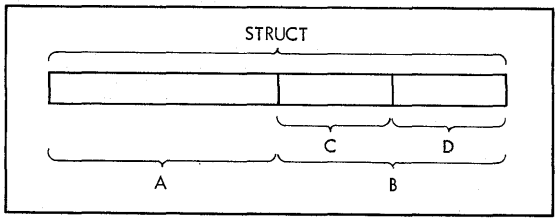
```
DCL  1  STRUCT FIXED(31),
        2 A FIXED(15),
        2 B FIXED(15);
```

defines a simple one-word structure that has two components, A and B. The numbers that precede structure and component names indicate the hierarchy of components within the structure. They are not used in references to the variables.

Components can themselves be structures and can have their own components. For example,

```
DCL  1  STRUCT FIXED(31),
        2 A FIXED(15),
        2 B FIXED(15),
            3 C BIT(8),
            3 D BIT(8);
```

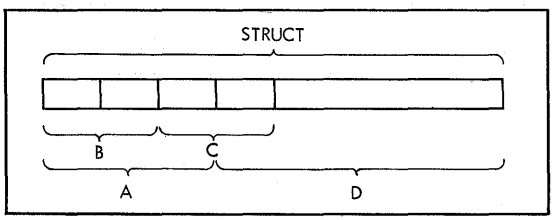
The mapping of this structure looks like this:



If the size of the structure is not sufficient to contain its components, then the components will overlap. For example:

```
DCL 1 STRUCT,
        2 A CHAR(3),
            3 B CHAR(2),
            3 C CHAR(2),
        2 D CHAR(4);
```

D will overlap both A and C in the resulting mapping:



If an asterisk appears instead of a structure or component name, then the structure or component will never be referenced explicitly by name. However, it may be referenced as part of another structure.

NOTE ON THE GENERATED ATTRIBUTE

A PL/S program can contain data defined by assembler instructions that follow a GENERATE DATA statement. Such data will not be referenced by PL/S statements unless the data is also defined by a DCL statement. This DCL statement will contain the attribute GENERATE (abbreviated GEND).

Similarly, labeled assembler instructions following a GENERATE statement will not be referenced by PL/S statements unless the labels are declared with the GENERATED keyword.

Data Manipulation

Data is copied from one location to another by a simple assignment of the form:

```
receiving variable = source;
```

When the source consists of operands connected by operators (an expression), the specified operations are performed and the result is placed in the receiving variable. Source variables are unchanged by the operations.

OPERATORS

Figure 7 shows the PL/S operators and their meanings.

Operator	Operation
+	Prefix plus
-	Prefix minus
*	Multiplication
/	Division for quotient
//	Division for remainder*
+	Addition
-	Subtraction
&	And
|	Or
&&	Exclusive or

*This operation yields the remainder contained in the even-numbered register.

Figure 7. PL/S Operators

REFERENCES TO ARRAYS AND STRINGS

A reference to an array will contain a subscript to specify which element is to be used. For example, this statement

```
A = B(4);
```

moves the fourth element of array B to A. The subscript in this example is the number 4, but subscripts may be variables or expressions.

Arrays are referenced one element at a time, but references to string variables can be to the entire string or to a portion of it. For example, if the character string A is declared like this,

```
DCL  A  CHAR(4);
```

then its first (left-most) byte might be referenced like this:

```
X = A(1);
```

Only the first byte of A is placed in X. The portion of a string to be referenced can be specified by a digit, as in the example above, by a variable, or by an expression. When more than one character or bit of a string is referenced, the starting and ending locations, separated by a colon, are specified. For example,

```
DCL  A  CHAR(80);
I = 10;
BUF = A(1:I);
```

moves the first to the tenth (inclusive) characters of A into BUF.

When a string is part of an array, then a reference to a portion of the string will specify both the array element and the string portion. In this example,

```
DCL  ARY(10)  CHAR(80);
X = ARY(4,80);
```

X receives the last byte of the fourth element of ARY. In this example,

```
I = 70;
X = ARY(4,I:80);
```

X receives the 70th through the 80th characters of the fourth element of ARY.

Control Flow Within a Procedure

UNCONDITIONAL BRANCHES

The GOTO statement is the PL/S facility for unconditional branches. This statement consists of the keywords GO TO followed by a transfer point, which is normally in the same procedure. The GOTO statement does not set up return linkage.

CONDITIONAL BRANCHES

Conditional branches occur at IF statements, which have this general form:

```
IF  definition of one or more comparisons
              THEN clause
ELSE clause
```

The THEN and ELSE clauses are the statements to be executed if the comparisons are true or false, respectively. When no ELSE clause appears, the statement following the THEN clause is executed if the comparison is false.

A single comparison definition is two operands joined by a comparison operator. The operands can be variables, constants, or expressions; the operators are shown in Figure 8.

Operator	Meaning
>	Greater than
<	Less than
¬>	Not greater than
¬<	Not less than
=	Equal to
¬=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

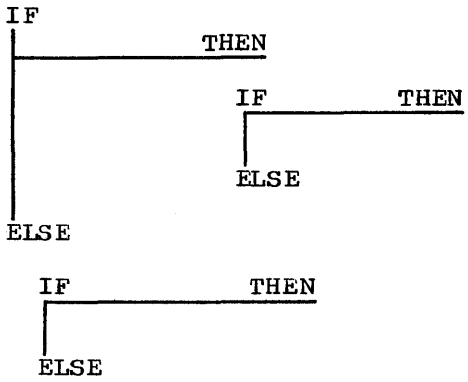
Figure 8. IF Statement Comparison Operators

Multiple comparison definitions are linked by the connector & or |. If two comparison definitions are linked by &, both must be true for the THEN clause to execute. If they are linked by |, either must be true.

Note on the Symbols & and |: These symbols are used to "and" and "or" bit strings as well as to join comparison definitions. To distinguish between the two uses, a comparison operand that contains & or | as bit string operators will be enclosed in parentheses in an IF statement.

IF Statement Format

The keyword ELSE is usually aligned in the same column as the IF keyword it is associated with. This alignment helps to identify paths, especially when IF statements are nested, i.e., when a THEN or ELSE clause contains another IF statement. For example,



The association of IFs with ELSEs is indicated by their alignment.

ITERATION

The DO statement is the PL/S facility for grouping statements in order to execute them as a group one or more times. Iteration of the group is controlled by four values:

- a control variable
- an initializing value for the control variable
- an increment or decrement value for the control variable
- a terminating value

These are arranged in the statement as follows:

```
DO  control variable = initializing value
    BY  increment or decrement  TO  terminating value
```

If no increment or decrement appears, it is assumed to be 1. If DO appears with no TO or BY values, i.e., DO; or DO I=10; then the group executes only once.

A single DO group extends from the DO statement to an END statement. When DO groups are nested, each will be closed by an END statement. An END statement is usually aligned vertically with the DO it closes.

Built-In Functions

The PL/S language has four built-in functions - ABS, ADDR, DIM, and LENGTH. They are not separate statements, but are embedded in PL/S statements when their functions are required. Figure 9 shows the general form and purpose of the built-in functions.

General Form	Purpose
ABS(variable or expression)	Obtains the absolute value of the variable or expression.
ADDR(variable or string constant)	Obtains the address of the specified data.
DIM(array name)	Obtains the dimension (the number of elements) of the specified array.
LENGTH(variable name or string constant or arithmetic constant)	Obtains the length of the specified data. The length will be in bytes, or in bits if the data was declared with the BIT attribute.

Figure 9. PL/S Built-In Functions

PL/S programs are documented by means of the listings shown in Figure 10.

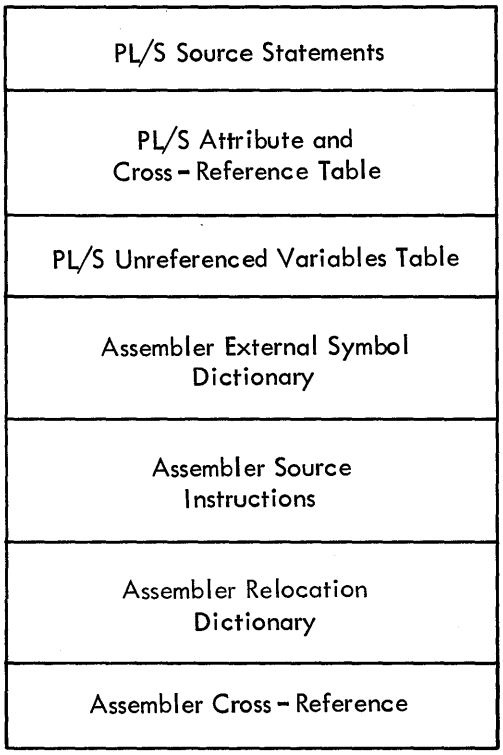


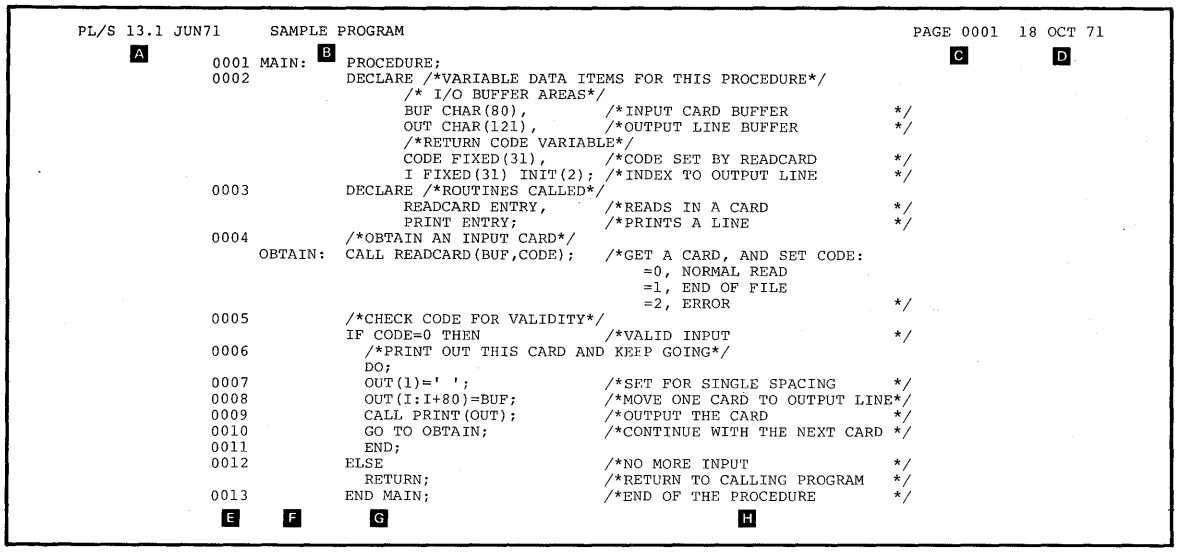
Figure 10. Sequence of Listings for a PL/S Program

This section describes the format and content of these major listings:

- A PL/S Source Statement Listing.
- A PL/S Attribute and Cross-Reference Table.
- A PL/S Unreferenced Variables Table.
- An Assembler Source Listing.
- An Assembler Cross Reference Table.

PL/S Source Statement Listing

Figure 11 shows a sample PL/S source statement listing.



```
PL/S 13.1 JUN71     SAMPLE PROGRAM                                          PAGE 0001  18 OCT 71
     A                B                                                        C          D
          0001 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 READCARD       */
                            I FIXED(31) INIT(2); /*INDEX TO OUTPUT LINE         */
          0003              DECLARE /*ROUTINES CALLED*/
                            READCARD ENTRY,        /*READS IN A CARD            */
                            PRINT ENTRY;           /*PRINTS A LINE              */
          0004              /*OBTAIN AN INPUT CARD*/
               OBTAIN:      CALL READCARD(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(I:I+80)=BUF;         /*MOVE ONE CARD TO OUTPUT LINE*/
          0009                CALL PRINT(OUT);         /*OUTPUT THE CARD         */
          0010                GO TO OBTAIN;            /*CONTINUE WITH THE NEXT CARD */
          0011                END;
          0012              ELSE                       /*NO MORE INPUT           */
                              RETURN;                  /*RETURN TO CALLING PROGRAM   */
          0013              END MAIN;                  /*END OF THE PROCEDURE    */
                     E    F       G                          H
```

Figure 11. PL/S Source Statement Listing

A The compiler, its level number, and its IBM internal release date.

B The name of the compiled PL/S program.

C The page number.

D The date of the compilation.

E The PL/S statements are consecutively numbered for reference purposes.

F Procedure names, entry names, and statement labels appear in this column.

H Remarks, which apply to particular statements, appear to the right.

PL/S Attribute and Cross-Reference Table

Figure 12 is a sample attribute and cross-reference table. This table lists each variable defined in the PL/S source program, and shows the declared and default attributes of each and the numbers of the statements that declared and referenced each.

```
PL/S 13.1 JUN71     SAMPLE PROGRAM                                    PAGE 0002  18 OCT 71

     DCL'D IN      NAME            ATTRIBUTE AND CROSS REFERENCE TABLE

 A     2   B     BUF          C  STATIC, LOCAL, CHARACTER(80), INTERNAL, BOUNDARY(BYTE,1)
                                 4, 8

       2         CODE            STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                 4, 5

       2         I               STATIC, LOCAL, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                                 8, 8

       1         MAIN            STATIC, LOCAL, ENTRY, EXTERNAL
                                 1, 13

       4         OBTAIN          STATIC, LOCAL, LABEL, INTERNAL
                                 4, 10

       2         OUT             STATIC, LOCAL, CHARACTER(121), INTERNAL, BOUNDARY(BYTE,1)
                                 7, 8, 9

       3         PRINT           STATIC, NONLOCAL, ENTRY, EXTERNAL
                                 9, 13

       3         READCARD        STATIC, NONLOCAL, ENTRY, EXTERNAL
                                 4, 13

                   *** NO VARIABLES WERE DEFAULTED TO FIXED(31)
                 D *** PROC. MAIN     HAD NO  ERRORS
                   000481 BYTES OF THE STORAGE ALLOCATED FOR THE SIZE OPTION WERE USED.
```

Figure 12. Attribute and Cross-Reference Table

A The DCL'D IN field identifies the PL/S statement that declared the variable. An asterisk following the statement number indicates only default attributes are given for the named variable. A variable so identified was not defined with attributes in a DECLARE statement.

B The NAME field lists the program variables in collating sequence.

C The ATTRIBUTE AND CROSS-REFERENCE TABLE field identifies the list of attributes that apply to the named variable and identifies, by statement number, those PL/S statements which reference the named variable.

D Notes pertaining to the compilation appear at the end of this table.

Unreferenced Variables Table

The attribute and cross reference table is often supplemented by a table of unreferenced variables. These variables are components of structures defined in the PL/S source program. Referenced components appear as normal entries in the attribute and cross reference table. Those not used appear in the unreferenced variables table. A sample entry appears like this:

```
DCL'D IN     NAME      IN STRUC

    26   VBLX          STRUCTA
    .      .             .
    .      .             .
    .      .             .
```

The DCL'D IN field references the PL/S source statement defining the entire structure in which the named variable resides.

Assembler Listing

This listing, shown in Figure 13, is a standard assembler listing produced by assembling the output of the PL/S compiler.

```
      SAMPLE PROGRAM                                                                    PAGE    1

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                          F15OCT70  10/18/71
                                                                                          C
                                      2          LCLA  &T,&SPN                          0001     00010
                                      3 .@001    ANOP                                   0001     00020
000000                                4 MAIN     CSECT ,                                0001     00030
000000 90EC D00C            0000C     5          STM   @E,@C,12(@D)                     0001     00040
000004 05B0                           6          BALR  @B,0                             0001     00050
000006                                7 @APSTART DS    OH                               0001     00060
000006                                8          USING @PSTART+00000,@B                 0001     00070
000006 50D0 B06E            00074     9          ST    @D,@SAV001+4                     0001     00080
00000A 41F0 B06A            00070    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
                                     13 *OBTAIN:  CALL READCARD(BUF,CODE);   /*GET A CARD, AND SET CODE:          00120
                                     14 *                                       =0, NORMAL READ                   00130
                                     15 *  A                                    =1, END OF FILE                   00140
                                     16 *                                       =2, ERROR                 */      00150
000014 41E0 B022            00028    17 OBTAIN  LA    @E,@CL9FF                        0004     00160
000018 0700                          18          CNOP  2,4                              0004     00170
00001A 58F0 B062            00068    19          L     @F,@V1            ADDRESS OF READCARD            0004     00180
00001E 051F                          20          BALR  @1,@F                            0004     00190
000020 000000B8                      21          DC    A(BUF)                           0004     00200
000024 00000184                      22          DC    A(CODE)                          0004     00210
                                     23 *  B      /*CHECK CODE FOR VALIDITY*/                                    00220
                                     24 *         IF CODE=0 THEN              /*VALID INPUT               */      00230
000028 1BFF                          25 @CL9FF  SR    @F,@F                            0005     00240
00002A 59F0 B17E            00184    26          C     @F,CODE                          0005     00250
00002E 4770 B056            0005C    27          BC    07,@9FD                          0005     00260
                                     28 *           /*PRINT OUT THIS CARD AND KEEP GOING*/                     00270
                                     29 *           DO;                                                      00280
                                     30 *           OUT(1)=' ';             /*SET FOR SINGLE SPACING      */   00290
000032 9240 B102       00108         31          MVI   OUT,C' '                         0007     00300
                                     32 *           OUT(I:I+80)=BUF;        /*MOVE ONE CARD TO OUTPUT LINE*/   00310
000036 5810 B182            00188    33          L     @1,I                             0008     00320
00003A 41A1 B101            00107    34          LA    @A,OUT-1(@1)                     0008     00330
00003E D24F A000 B0B2 00000 000B8    35          MVC   0(80,@A),BUF                     0008     00340
000044 9240 A050       00050         36          MVI   80(@A),C' '                      0008     00350
                                     37             CALL PRINT(OUT);         /*OUTPUT THE CARD           */   00360
000048 41E0 B00E            00014    38          LA    @E,@CL9FC                        0009     00370
00004C 0700                          39          CNOP  2,4                              0009     00380
00004E 58F0 B066            0006C    40          L     @F,@V2            ADDRESS OF PRINT               0009     00390
000052 051F                          41          BALR  @1,@F                            0009     00400
000054 00000108                      42          DC    A(OUT)                           0009     00410
                                     43 *           GO TO OBTAIN;           /*CONTINUE WITH THE NEXT CARD */   00420
000058 47F0 B00E            00014    44          BC    15,OBTAIN                        0010     00430
                                     45 *           END;                                                     00440
                                     46 *         ELSE                      /*NO MORE INPUT             */   00450
                                     47 *           RETURN;                 /*RETURN TO CALLING PROGRAM */   00460
                                     48 *         END MAIN;                 /*END OF THE PROCEDURE      */   00470
00005C                               49 @9FA    EQU   *                                0013     00480
00005C 58D0 D004            00004    50 @EL01   L     @D,4(0,@D)                       0013     00490
000060 98EC D00C            0000C    51          LM    @E,@C,12(@D)                     0013     00500
000064 07FE                          52          BCR   15,@E                            0013     00510
000066                               53 @DATA1  EQU   *                                       00520
000000                               54 @0      EQU   00                EQUATES FOR REGISTER 0-15        00530
000001                               55 @1      EQU   01                                         00540
000002                               56 @2      EQU   02                                         00550
```

Figure 13. Assembler Listing (1 of 2)

```
                                                                                          PAGE    2
      SAMPLE PROGRAM
                                                                              F15OCT70  10/18/71
  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT
                                                                                          00560
000003                              57 @3         EQU    03                               00570
000004                              58 @4         EQU    04                               00580
000005                              59 @5         EQU    05                               00590
000006                              60 @6         EQU    06                               00600
000007                              61 @7         EQU    07                               00610
000008                              62 @8         EQU    08                               00620
000009                              63 @9         EQU    09                               00630
00000A                              64 @A         EQU    10                               00640
00000B                              65 @B         EQU    11                               00650
00000C                              66 @C         EQU    12                               00660
00000D                              67 @D         EQU    13                               00670
00000E                              68 @E         EQU    14                               00680
00000F                              69 @F         EQU    15
000066 0000                                                                               00690
000068 00000000                     70 @V1        DC     V(READCARD) [E]                  00700
00006C 00000000                     71 @V2        DC     V(PRINT)                         00710
000070                              72            DS     0F                               00720
000070                              73            DS     0D                               00730
000070                              74 @DATA      EQU    *                                00740
000070                              75 @SAV001    EQU    @DATA+00000000   72 BYTE(S) ON WORD   00750
0000B8                              76 BUF        EQU    @DATA+00000072   80 BYTE(S)      00760
000108                              77 OUT        EQU    @DATA+00000152   121 BYTE(S)     00770
000184                              78 CODE       EQU    @DATA+00000276   FULLWORD INTEGER     00780
000188                              79            ORG    @DATA+00000280               [F] 00790
000188                              80 I          EQU    *                FULLWORD INTEGER     00800
000188 00000002                     81            DC     FL4'2'                           00810
000070                              82            ORG    @DATA                            00820
000070                              83            DS     00000284C                        00830
00018C                              84 @TEMPS     DS     0F                               00840
00018C                          [D] 85 @DATEND    EQU    *                                00850
000014                              86 @CL9FC     EQU    OBTAIN                           00860
00005C                              87 @9FD       EQU    @EL01                            00870
000000                              88            END    MAIN
```

Figure 13. Assembler Listing (2 of 2)

A The PL/S source statements producing executable instructions appear as comments ahead of the generated code. A label on a PL/S statement becomes the label of the first generated instruction.

B The compiler-generated labels appear in the label field of the listing.

C The statement numbers for those PL/S statements producing executable assembler code appear in the remarks field of the generated instructions.

D The assembler data area. This area is laid out in the format shown in Figure 14.

E The PL/S source program variables are listed in the data area of the assembler listing. For entry points and labels external to the procedure, the compiler creates V-type address constants; for other external data, the compiler generates A-type address constants and EXTRNs.

F The compiler generates assembler remarks describing the type of data being defined. The compiler derives these remarks from information in the original PL/S declaration.

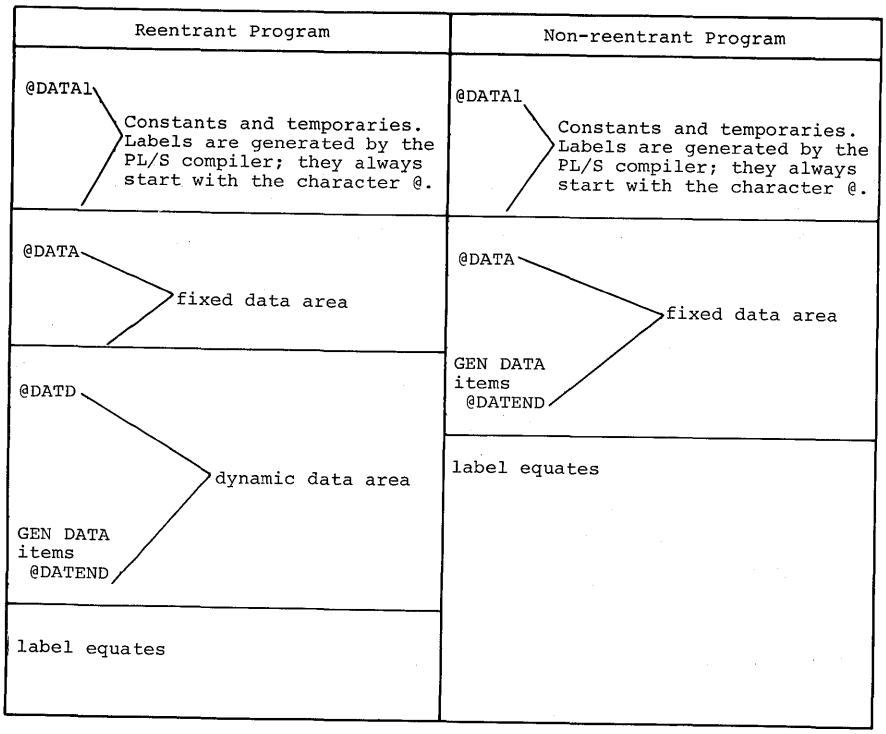


Figure 14. Data Area Layout

COMPILER-GENERATED LABELS

The labels (statement identifiers) that appear in the PL/S source program are reproduced in the compiler-generated assembler code. However, the compiler generates additional labels to identify areas, values, and statements created by expansion of the PL/S program into assembler code. To help you identify various items in the assembler code, the conventions for compiler-generated labels are listed in Figure 15.

As shown in Figure 15, compiler-generated labels begin with either @, &, or A. The label types that are used 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.

Label	What the Label Identifies
.@001	An ANOP following a LCLA assembler instruction.
&SPN	Subpool number of dynamic storage for reentrant procedures.
&T	Used to initialize some arrays.
@AD01	The address of @DATA1 in a non-reentrant program when the DATAREG option is specified.
@Ainteger	An A-type address constant.
@Binteger	A bit constant.
@Cinteger	A character constant.

Figure 15. Labels Generated by the Compiler (Part 1 of 2)

Label	What the Label Identifies
@CLC @MVC @NC @OC @XC	Instructions which are executed by means of an assembler EX instruction.
@CLinteger	Used in non-reentrant procedures to identify and branch around argument lists.
@CTEMPinteger	A string temporary.
@DATA	The start of the static data area, and the end of the executable generated code.
@DATA1	The start of the compiler constant and temporary area.
@DATD	The start of the DSECT that describes the dynamic storage area.
@DATEND	The end of the DSECT that describes the dynamic storage, or the end of the data area in a non-reentrant program.
@Dinteger	An arithmetic constant.
@DOinteger	Statements in the generation of a DO loop.
@ELinteger	The epilogue of a procedure.
@IFinteger	A value that represents the length of a temporary area to be cleared for string expressions (variable length substrings).
@integer	IF branches and branches around ELSE statements.
@L	A value that represents the length of a temporary area that must be set to zeros in reentrant procedures.
@PLinteger	An argument list for reentrant procedures.
@PSTART	First first instruction following the BALR that establishes the primary base register.
@SAVinteger	A procedure save area.
@SIZ001	A value that represents the size of the dynamic storage area.
@TEMPinteger	An arithmetic temporary which has high-order zeros.
@TEMPS	An area that contains space for temporaries.
@Tinteger	A temporary location used for evaluating an arithmetic expression.
@Vinteger	A V-type address constant.
@Xinteger	A hexadecimal constant.
@0,@1,...@F	Symbolic names for the general registers.
Ainteger	Name generated for items declared with an * instead of a name.

Figure 15. Labels Generated by the Compiler (Part 2 of 2)

Assembler Cross-Reference

The assembler cross-reference for a PL/S program shows no references for many variables actually used in the program. This happens because the PL/S compiler does not always use symbolic variable names in the assembler source code it produces. Only STATIC LOCAL and AUTOMATIC variables that are not in structures or arrays are referenced symbolically.

To determine which assembler instructions use a given variable, look up the variable in the PL/S Attribute and Cross-Reference Table. This table shows the statement numbers of the PL/S statements that reference the variable. Then find these statement numbers in the assembler source listing. The variable will be referenced in the assembler code produced from these PL/S statements. Figure 16 illustrates this process for the variable VBLE.

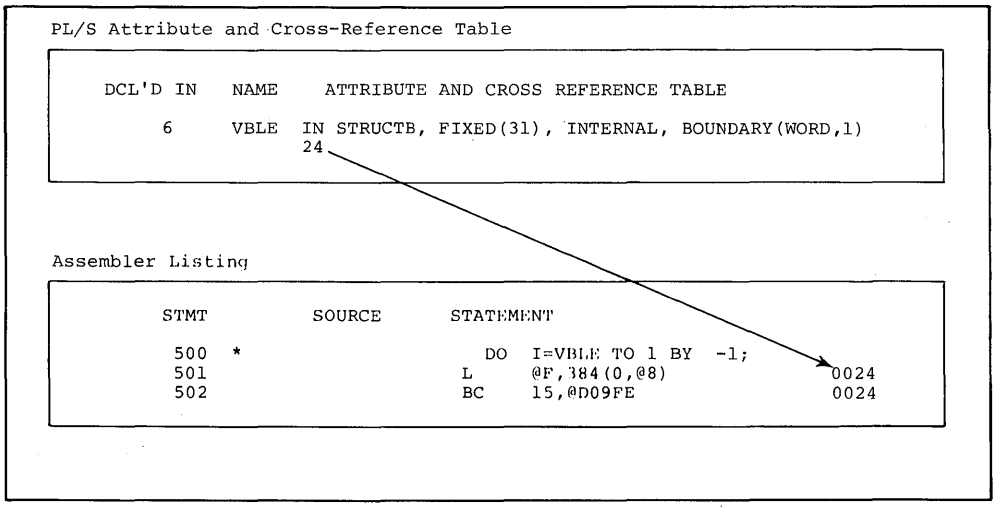


Figure 16. Locating a Variable Not Referenced Symbolically in Assembler Code

You can usually recognize the variable in the assembler instructions generated from the PL/S statement. In case of doubt, check the assembler cross-reference to find where the variable is defined, and look up the definition. The definition will give you a displacement by which you can recognize the reference to the variable. Figure 17 shows this process.

PL/S Attribute and Cross-Reference Table

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

    6        VBLE     IN STRUCTB, FIXED(31), INTERNAL, BOUNDARY(WORD,1)
                      24
```

Assembler Listing

```
     STMT        SOURCE          STATEMENT

      500  *                        DO    I=VBLE TO 1 BY -1;
      501                         L      @F,384(0,@8)              0024
      502                         BC     15,@D0@FE                 0024
       .                           .
       .                           .
       .                           .
     1200 VBLE                    EQU    STRUCTB+00000384  FULLWORD INTEGER
```

Assembler Cross-Reference

```
     SYMBOL         LEN       VALUE        DEFN       REFERENCES

     VBLE         00001     00000180       1200
```

Figure 17. Recognizing a Variable Not Referenced Symbolically in Assembler Code

Section 4: Guidelines for Modifying Assembler Code

You can order the assembler source code as machine readable material if you want to make modifications to the compiler-generated code. However, you should be aware that certain problems may arise from these modifications.

When modules coded in PL/S are recompiled by IBM for a new release, the assembler code for certain statements may be generated differently. Thus modifications may not work in a new release if they depend on PL/S statements always producing the same assembler source code. The following guidelines will simplify modifications and will help assure that the modifications you make are impacted as little as possible by subsequent PL/S compilations.

Modifying Instructions

1. Do not make references to compiler-generated labels (shown in Figure 15, Section 3). These labels may change when a PL/S module is re-compiled. Since the compiler does not generate labels that begin with a dollar sign character, a safe rule for making up your own assembler labels is to start them with the dollar sign character.

2. Give an explicit length when you add instructions that require a length. This is necessary because the PL/S compiler defines data to the assembler by means of simple equate statements.

3. Use explicit base and displacement values when making references to parameter lists, BASED variables, or NONLOCAL variables. This is necessary because the PL/S compiler defines data to the assembler by means of simple equate statements.

4. Do not insert any new assembler instructions into an instruction sequence generated for a single PL/S statement. Make insertions either before or after the generated instruction sequence.

5. When making modifications to code generated for a PL/S statement, replace the entire group of generated assembler instructions. This technique allows you to do a complete code replacement for a PL/S statement if a different sequence of instructions is generated for the same PL/S statement in a new release.

6. If you insert assembler instructions immediately after a CALL statement, you may also need to change the code generated for the CALL statement itself, or the code generated for the statement following the CALL statement. This is because the return point for the CALL is sometimes defined at the assembler level in terms of code generated for the statement following the CALL. This guideline will ensure that your inserted code is not bypassed.

In the following example, @CL9FF is the return point from the called procedure.

```
         CALL        READCARD(BUF,CODE);

OBTAIN       LA     @E,@CL9FF
             CNOP   2,4
             L      @F,@V1
             BALR   @1,@F
             DC     A(BUF)
             DC     A(CODE)
    /* CHECK CODE FOR VALIDITY */
 A
@CL9FF       SR   @F,@F
             .
             .
             .
```

Any code inserted before A will be bypassed. To avoid this problem, code inserted at A should have a unique label. Then you can modify the code generated for the CALL to make the new label the return point. For example:

```
OBTAIN       LA     @E,$NEWLABL
             CNOP   2,4
             L      @F,@V1
             BALR   @1,@F
             DC     A(BUF)
             DC     A(CODE)
    /* CHECK CODE FOR VALIDITY */

$NEWLABL         .
             .
             .
@CL9FF       SR   @F,@F
             .
             .
             .
```

Do not move the compiler generated return point label to your new code, because this label may change in a subsequent release.

Modifying Data

Add new data at the end of the generated assembler data area. The symbol @DATEND on the assembler listing identifies the end of this area (see Figure 14, Section 3 for a description of this area.) You should insert the additional data just ahead of this symbol.

If you insert new data elsewhere, all data following the inserted data will have new displacements. Then you must modify all instructions referring to the displaced data to use the new displacements. Similarly, if you increase the length of a data item, move it to the end of the generated assembler data area so that it does not displace all subsequent data.

When you increase or decrease the length of a data item, you must also modify the length in all instructions that refer to the item.

Refer to PL/S REGISTER variables by the PL/S name, not by the associated general register number. This will help keep such references valid if a different physical register is used in future compilations.

STRUCTURES

If you add new data to a structure that has the BASED attribute, add it at the end of the structure to avoid displacing data within the structure.

If you add new data to a structure that has the STATIC or AUTOMATIC attributes, you should also add it at the end of the structure to avoid displacing data within the structure. Then you should move the STATIC or AUTOMATIC structure to the end of the assembler data area to avoid displacing data that follows the structure.

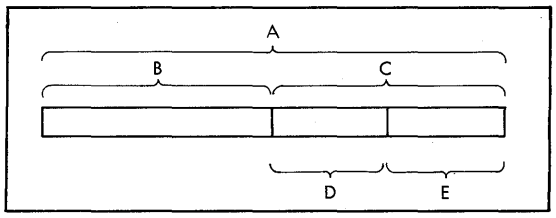
If you change the length of a component of a STATIC, AUTOMATIC, or BASED structure, you must change the length in all instructions that reference:

- the changed component, and
- any structure that contains the changed component

For example, this structure,

```
DCL  1  A  CHAR(80),
        2  B  CHAR(40).
        2  C  CHAR(40),
           3  D  CHAR(20),
           3  E  CHAR(20);
```

appears as:



E is referenced when C, its contianing structure, is referenced, and when A, the containing structure for C, is referenced. So if you change the length of E, you must change the length in all instructions that reference E, C, or A.

If you simply move a data item within a structure, you should modify all instructions using this data to use the new location.

Section 5: Glossary

ABS function: One of the built-in functions; it is used to obtain the absolute value of a variable or expression.

ADDR function: One of the built-in functions; it is used to obtain the address of some data.

argument: A constant, variable, or expression passed to a called procedure. Arguments appear on a CALL statement.

argument list: An area of storage used to contain the address of each argument that appears on a CALL statement when the CALL statement invokes another procedure.

array: A collection of data that has identical attributes. The data occupies a contiguous area of storage and is referenced by a common name.

attribute: A characteristic of data. Most attributes have an associated keyword in PL/S.

AUTOMATIC: A data attribute that causes a variable to be assigned space in a dynamic storage area.

BASED: A data attribute that causes no storage to be assigned to a variable. The attributes of a BASED variable are applied to a storage area indicated by a locator.

BIT: A data attribute used to define a bit string.

BOUNDARY: A data attribute used to align a variable on a specified boundary.

built-in function: A facility of PL/S that causes the compiler to perform a programming task.

BY: A DO statement keyword that specifies a value to be added to or subtracted from a control variable in order to control iteration of a DO group.

CALL statement: A statement used to invoke an external or internal procedure.

called procedure: A procedure invoked by another procedure. The CALL statement invokes another procedure.

calling procedure: A procedure that invokes another procedure. The CALL statement invokes another procedure.

CODE: A RETURN statement keyword that precedes a return value (a constant,variable,or expression.)

CODEREG: See "code register".

code register: A register used to address compiler-generated code. Register 11 is used unless the CODEREG option on the PROCEDURE statement indicates otherwise.

comparison definition: Two operands separated by a comparison operator that appears on an IF statement.

components: The parts of a structure. A component can itself be a structure.

connector: Either the & or | operator used to connect comparison definitions on an IF statement.

constant: A fixed or invariable value or data item.

control variable: A variable used (in conjunction with BY and TO values) to control iteration of a DO group.

DATAREG: See "data register".

data register: A register used in a reentrant environment to address data. Register 12 is used unless the DATAREG option on the PROCEDURE statement indicates otherwise.

DECLARE statement: A statement used to describe the attributes of data.

DIM function: One of the built-in functions; it is used to obtain the dimension of a previously declared array.

dimension: The number of elements in an array.

DO group: A set of statements that begin with a DO statement and end with an END statement. The group may execute once or repeatedly.

DO statement: A statement used to group a number of statements in a procedure.

dynamic storage area: The storage for data that is allocated automatically upon entry into a procedure.

element: One of a collection of data in an array.

ELSE clause: The part of an IF statement used to specify the action to be performed if the comparision of operands on the IF statement is false.

END statement: A statement used to indicate the end of a procedure or the end of one or more DO groups.

ENTRY: A data attribute applied to the label of a PROCEDURE or ENTRY statement. These labels are entry points.

entry point: Any place within a procedure to which control can be passed by another procedure.

ENTRY statement: A statement used to designate a secondary entry point for a procedure.

expression: Constants and variables used in combination with operators to represent an operation to be performed.

EXTERNAL: A data attribute. When two or more external procedures must reference a variable, the EXTERNAL attribute appears on the DECLARE statement for the variable in each procedure.

external procedure: A procedure that is not internal to another procedure.

GENERATE : The statement that allows one or more assembler instructions to be placed in PL/S compiler-generated code. The GENERATE DATA statement allows the assembler instructions to define data.

GOTO statement: A statement used to transfer control to a point preceding or following this statement.

IF statement: A statement used for conditional statement execution. This statement is always followed by a THEN clause and, optionally, an ELSE clause.

indirect addressing: A technique used to obtain data by referencing a variable that contains the address of the desired data.

INTERNAL: A data attribute which specifies that the associated variable is not referenced outside the declaring procedure and any procedures nested within the declaring procedure.

internal procedure: A procedure that is contained within another procedure.

keyword: A symbol that identifies a data attribute, a PL/S statement, or some qualifying information for a statement.

LABEL: A data attribute applied to the label of any statement other than PROCEDURE or ENTRY. These labels are not entry points.

LENGTH function: One of the built-in functions; it is used to obtain the length of some data.

level number: A number assigned to a structure or a component to indicate its position within the hierarchy of a structure.

LOCAL: A data attribute that causes storage for a variable to be assigned in the CSECT of the declaring procedure.

locator: A variable or expression that follows the BASED attribute and is used to locate data, or a pointer supplied by pointer notation when the data is referenced.

microfiche: Microfilm containing program listings.

nesting: Inclusion of one or more procedures, IF statements, or DO statements within another procedure, IF statement, or DO statement, respectively.

NONLOCAL: A data attribute that causes no storage for a variable to be assigned in the CSECT of the declaring procedure. Storage is assigned elsewhere.

operand: One or more constants and variables that are operated upon.

operator: One or more symbols used in combination to indicate the action to be performed on operands.

parameter: A variable name that appears on a PROCEDURE or ENTRY statement. This name is used in a called procedure to reference information passed to it from the calling procedure.

pointer: Data that is taken to be the address of some other data.

pointer notation: The notation used when data is to be located indirectly by an address contained at the location of some POINTER variable. The composite symbol -> appears between the POINTER variable and the name of the data.

precision: The number of bits assigned for the maximum positive value of either FIXED or POINTER data.

primary entry point: The major entry point of a procedure. It is signified by the appearance of a PROCEDURE statement.

procedure: An independent, named block of statements that defines a specific portion of a program.

PROCEDURE statement: A statement used to indicate the primary entry point for a procedure.

reentrant: A characteristic of a procedure that causes dynamic allocation of space for data, save areas, and compiler work areas. This characteristic is applied to a procedure when the REENTRANT option appears on the PROCEDURE statement of the external procedure.

RELEASE statement: A PL/S statement used to release a register that was restricted earlier in the procedure.

RESPECIFY statement: A statement used to provide or change a locator, or to alter register availability.

RESTRICTED: A data attribute which indicates that a specified register is not available for the compiler to use in the code it produces.

RESTRICT statement: A statement used to restrict the compiler's use of a certain register.

RETURN: The PL/S statement that sends control to the statement following the CALL statement in the calling procedure. The RETURN TO statement sends control to a specified labeled statement.

secondary entry point: An entry point in a procedure other than the primary entry point. It is signified by the appearance of an ENTRY statement.

source expression: That part of an assignment statement that appears to the right of the equal sign. Its value is assigned to the receiver.

static storage area: The fixed storage for data that once assigned is never reassigned.

string: A sequence of 8-bit EBCDIC characters or else a sequence of bits that are unrelated to each other.

structure: A collection of data that usually has unlike attributes (the data can have identical attributes). The data occupies a contiguous area of storage and names are assigned to parts of the data so that the entire area or portions of it can be referenced.

subscript expression: An expression that appears in parentheses following an array name. It is used to reference an element of an array.

substring expression: An expression that appears in parentheses following the variable name assigned to string data. It is used to reference a portion of string data.

terminating value: A constant, variable, or expression used to stop iteration of a DO group. Iteration stops when the control value exceeds the terminating value.

THEN clause: The part of an IF statement used to specify the action to be performed if the comparison of operands on the IF statement is true.

TO: A DO statement keyword that specifies a terminating value.

UNRESTRICTED: A data attribute which indicates that a specified register is available for the compiler to use in the code it produces.

variable: Symbolic representation of a quantity or data string that occupies a storage area.

VLIST: A keyword used in the DECLARE statement for a procedure name to indicate that the number of arguments passed by the procedure may vary. VLIST causes the parameter list to have its high-order bit in the last word set on.

Index

Guide to PL/S-Generated Listings

GC28-6786-0

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? _____
Number of latest Technical Newsletter (if any) concerning this publication: _____
Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.

Cut or Fold Along Line

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

GC28-6786-0

Guide to PL/S-Generated Listings Printed in U.S.A. GC28-6786-0

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)