GC28-6794-0

Systems

Guide to PL/S II

IBM

First Edition (May, 1974)

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.

Purpose of Guide

The PL/S II (Programming Language/Systems-Second Version) compiler is a proprietary program used by IBM to develop other programs that are made generally available. The PL/S II compiler is not available outside IBM. Programs written by IBM in PL/S II are documented by means of listings in microfiche form. This guide 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 II 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 II source programs. Furthermore, it does not list the assembler code generated for each PL/S II statement; the large number of possible combinations of source language elements makes such a list impractical.

Information on the first version of PL/S can be found in the publication, *Guide to PL/S-Generated Listings,* GC28-6786.

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/I (which PL/S II closely resembles).

The general knowledge of PL/S II obtained from this guide should assist these programmers in interpreting PL/S II program listings. They may find it easier to understand what a system module does by reading the PL/S II 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 II language, the compiler, and the output produced by a compilation.

Section 2, the PL/S II Language, describes the purpose and format of PL/S II 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 II.

Contents

Figures

Programming Language/Systems II (PL/S II) 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 II is designed to express operations used in systems programs. In assembler language many instructions are usually required to express these operations. With PL/S II, data can be defined and utilized with fewer statements. Because PL/S II is more compact and English-like than assembler code, PL/S II source programs can be understood by the reader faster than equivalent assembler programs.

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

The PL/S II Compiler

PL/S II 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 II compiler translates the PL/S II source language statements into assembler language instructions suitable for input to an assembler. Several assembler language instructions usually result from a single PL/S II statement.
2. An 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 II translation process.

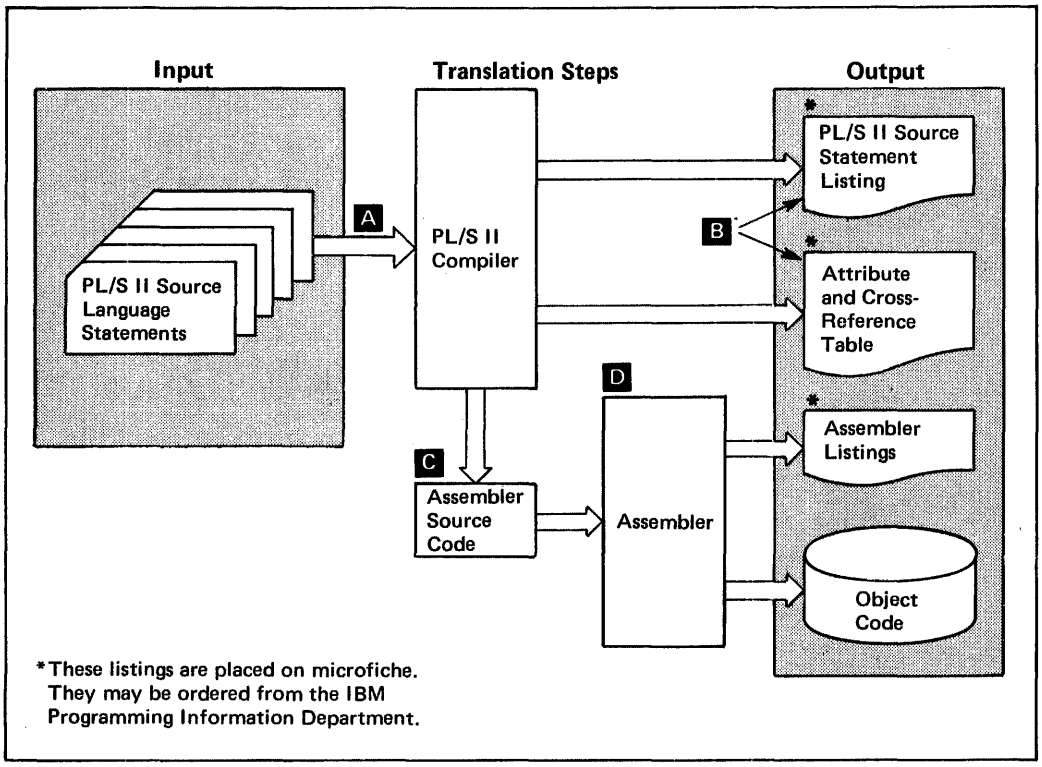


Figure 1. Overview of PL/S II Translation Process

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

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

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

D The 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 II Keywords

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

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

PL/S II Procedures

PL/S II 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. Its use is for necessarily first assembler statements, such as macro definitions.) 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)  ;
```

OPTIONS on the PROCEDURE Statement

The following table indicates the OPTIONS which are available to alter PL/S II compiler-generated prolog and epilog code. Note that some can be coded only on the PROCEDURE statement of an external procedure, while others can be coded on either external or internal procedures.

REENTRANT*	indicates that compiler-generated code should be reentrant code.
CODEREG*	specifies the register or registers to be used for addressing code.
NOCODEREG*	specifies that the compiler should not set up code addressing registers.
DATAREG*	specifies the register or registers to be used for addressing data.
NODATAREG*	specifies that the compiler should not generate a GETMAIN, FREEMAIN or data addressability.
SAVE	specifies which registers are to be saved on entry to and restored on exit from the procedure.
NOSAVE	specifies which registers should not be saved on entry to and restored on exit from the procedure.
SAVEAREA	specifies that a save area, chained via register 13, is to be generated.
NOSAVEAREA	specifies that no save area is to be generated for the procedure.
ID*	specifies that an identifying character string is to appear as part of the compiler-generated prologue.
NOID*	indicates that there is no identifying character string.
OS*	indicates that the operating environment of the procedure is OS.
DOS*	indicates that the operating environment of the procedure is DOS.
ENTREG	specifies that register 15 locates the entry point.
NOENTREG	indicates that no register can be relied on to locate the entry point.
RETREG	specifies that register 14 locates the return location.
NORETREG	indicates that no register can be relied on to locate the return location.
PARMREG	specifies that register 1 locates parameter lists.
NOPARMREG	indicates that no register can be relied on to locate a parameter list.
SAVEREG*	specifies that register 13 locates save areas on the save chain.
NOSAVEREG*	indicates that there is no save register and therefore, no save chain.
PROLOG*	indicates that the compiler generates prologue code for the PROCEDURE and ENTRY statements.
NOPROLOG*	specifies that no prologue code is generated by the compiler.
EPILOG*	specifies that no epilogue code is generated by the compiler for the procedure RETURN and END statements.
NOEPILOG*	suppresses generation of epilogue code by the compiler.
AUTODATA*	indicates the size limit of the dynamic DSECT created for a reentrant procedure.
NOAUTODATA*	indicates that there should be no dynamic data.
KEY*	indicates that the protection key is modified in a procedure.

*external procedure only

Figure 2. Procedure Statement Options

Saving Registers Across Procedures

The assembler code produced for procedures follows standard linkage conventions. Figure 3 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 3. Linkage Registers

A procedure normally has a save area to preserve its registers. The format of this area is shown in Figure 4.

Word	Contents
1	Not used.
2	Address of calling procedure's 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 4. Save Area Format

The size of the area can be governed by the SAVEAREA option. The area can be eliminated by the NOSAVEAREA option. If neither option is specified, the compiler may eliminate it if there are no CALL or GENERATE statements, or may (for internal procedures) alter its size and/or bypass chaining.

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 NOSAVE option appears on the PROC statement. SAVE is followed by an explicit list of registers to be saved and restored by the procedure; NOSAVE is followed by a list of those that are not to be saved and restored.

Compiler Register Assignments

As the PL/S II compiler produces assembler code, it assigns registers to the functions shown in Figure 5.

Register	Function
0	Used in REENTRANT prolog and epilog. Otherwise available.
1	Parameter list pointer across calls. Used in REENTRANT prolog and epilog. Otherwise available.
13	Save area pointer.
14	Used to contain a return address across calls. Otherwise available.
15	Used to contain an entry point address across calls. Otherwise available.
Some unused register	Base register for addressing code. For REENTRANT , another unused register is assigned to address dynamic data.
All other registers	Used for arithmetic, indexing, and addressing.

Figure 5. Compiler-Assigned Functions for Registers

REENTRANT and AUTODATA Options

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
- dynamic data defined with 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.

The NOAUTODATA option prohibits the use of dynamic data. AUTODATA allows it, and may include a bound on its size.

CODEREG and DATAREG Options

The CODEREG option is followed by one or more register numbers. These registers are established as the base registers for code addressing. NOCODEREG 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 the dynamic data area. NODATAREG tells the compiler not to obtain the area, and not to establish addressability for it, although the compiler still sets up its description.

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 for an external procedure or for 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 6.

If the argument is:	The argument list address is:
A variable, not in parentheses.	The address of the 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 6. 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 attribute 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 the following 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.

Arithmetic data is interpreted as a binary, fixed-point integer; it is identified by the keywords BINARY and/or FIXED. Either 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). Data can also be specified as SIGNED (allowing negative values) or UNSIGNED (non-negative). Precisions of 15 and 31 can be SIGNED or UNSIGNED (defaulting to SIGNED); precisions of 8, 16, 24 and 32 can only be UNSIGNED.

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.)

The VALUERANGE attribute gives possible actual labels for BASED LABEL or ENTRY items. ENTRY data may also have an OPTIONS attribute to indicate special entry requirements and actions; possible OPTIONS values are:

EXIT	notes that the called procedure may exit unusually (e.g., ABEND).
FLOWS	notes possible continuation point labels.
NOSEQFLOW	notes that simple continuation flow cannot occur.
REFS	notes which data may be referenced.
SETS	notes which data may be assigned new values.
VLIST	notes the need for setting on the high-order bit in the last word of the argument list.

Figure 7. OPTIONS Attribute Keywords

Initialization and Constants

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 assembler-resolvable expression. PL/S II has five types of literal constants – decimal, hexadecimal, character, bit, and binary. The general form of each is shown in Figure 8.

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

Figure 8. Form of PL/S II Literal Constants

PL/S II also provides declaring names for constants. The DECLARE statement is used for giving the name, the data type attributes, and the attribute CONSTANT followed by the desired value.

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 II 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 reserves the register in the declaring procedure (either internal or external) and its contained procedures. The register may be released for compiler use at any point by a RESPECIFY statement. The RESPECIFY statement, when used to release a register, consists of the keyword RESPECIFY, one or more register names, and the keyword UNRESTRICTED.

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

Note on Register Restriction: Restriction applies to a particular name for a physical register; therefore, all symbolic names by which the physical register is known must be unrestricted to make the register available.

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 DECLARE statement with the NONLOCAL attribute means that the data is assigned storage in a procedure other than the declaring one.

Certain system data have absolute locations. Such data are declared with the LOCATION attribute, which includes the byte on which such data starts.

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, except 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 DEFINED or BASED attribute. A DECLARE with DEFINED or BASED simply defines a set of attributes. These attributes are applied to a storage area specified by the name of the overlayed item (DEFINED, abbreviated DEF) or by a locator address (BASED). The POSITION attribute (abbreviated POS) can be used with DEFINED or BASED to indicate a relative position with respect to the item or address.
Here are some examples:

```
DCL P POINTER;
DCL H BIT(8) DEFINED(P);
DCL B CHAR(4) BASED(P);
DCL C CHAR(1) DEF(B) POS(4);
```

H maps to the high-order byte of P. P locates bytes described by B as CHAR(4), the last of these described by C as CHAR(1).

The BASED keyword is not always followed by a locator, but before the variable is used, a locator will be supplied. PL/S II 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
```

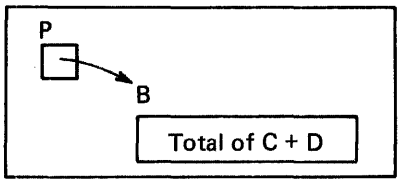
For example,

```
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 ;
```

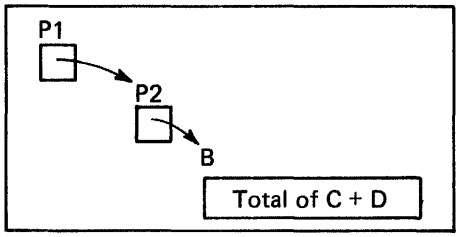
causes this:



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 expression);
```

The specified pointer expression 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 unless pointer notation or until another RESPECIFY statement changes it.

Arrays

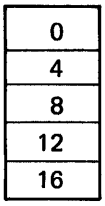
An array is a collection of variables (called elements) that have identical attributes and that occupy a contiguous storage area; the collection has a common name.

A DCL statement defines an array if the variable name is followed immediately by a decimal number or numbers in parentheses. Each number is a dimension of the array: i.e., the number of elements in it. An array can have up to 15 dimensions.

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);
```

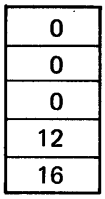
defines a five-element array of fullword arithmetic variables that is initialized like this:



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:



If the INITIAL attributes 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

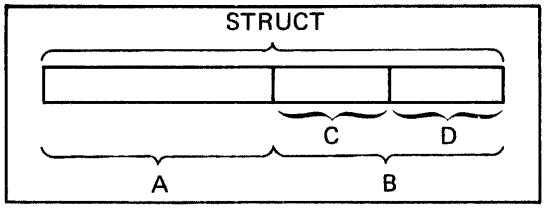
```
DCL  1  STRUCT FIXED(31),
        2 A FIXED(15),
        2 B FIXED(15);
```

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);
```

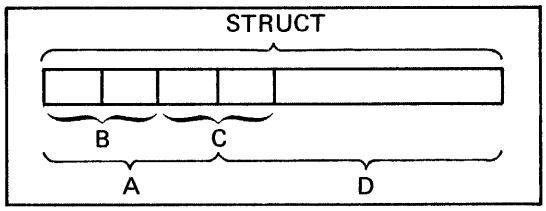
The mapping of this structure looks like this:



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 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.

The GENERATED Attribute

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

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

The GENERATE statement can indicate the items it defines by using the DEFS option.

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 9 shows the PL/S II 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 9. PL/S II Operators

References to Arrays and Strings

A reference to an array will contain a subscript (or subscripts if multiply dimensioned) 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 II facility for unconditional branches. This statement consists of the keywords GO TO (or the single keyword GOTO) followed by a transfer point, which is normally in the same procedure. The GOTO statement does not set up return linkage.

Conditional Flow

Conditional flow occurs 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, no false-path action is required.

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 10.

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

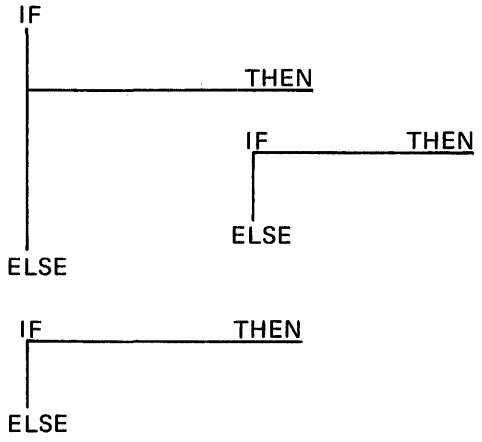
Figure 10. IF Statement Comparison Operators

Multiple comparison definitions are linked by the connectors & 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. Parentheses can be used to alter comparison ordering; the 'not' operation (¬) can be used preceding a parenthesized comparison (or linked comparisons) to invert the result of the comparison.

Note on the Symbols & and | : These symbols are used to "and" and "or" bit strings as well as to join comparison definitions. A comparison operand that contains & or | as bit string operators will be enclosed in parentheses.

IF Statement Format

The keyword ELSE is usually aligned in the same column as the IF keyword or the THEN 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 II facility for grouping statements in order to execute them as a group one or more times. Iteration of the group is controlled by:

- a control variable
- an initializing value for the control variable
- an increment or decrement value (BY, defaulting to 1)
- a limit value (TO)
- conditions for continuing, either as leading decision (WHILE) or trailing decision (UNTIL).

These are arranged in the statement as follows:

```
DO [control variable = initializing value
   [BY increment or decrement] [TO limit value]]
   [[WHILE(necessary condition)]
   [UNTIL (terminating condition)]]
```

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.

If the statement is just {DO;}, the group executes once. If just a WHILE or UNTIL appears, the group is repeatedly executed under control of the WHILE or UNTIL condition; the WHILE is tested before each execution, stopping when the condition fails; the UNTIL is tested after each execution, stopping when the condition becomes true.

Built-In Facilities

The PL/S II language has a number of built-in functions. They are not separate statements, but are embedded in PL/S II expressions when their functions are required. Figure 11 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 expression.
ADDR (variable)	Obtains the address of the specified data.
DIM (array name [,dimension])	Obtains the dimension extent (the number of elements) of the specified array. For multi-dimensioned arrays, the second argument selects the dimension.
LENGTH (variable name or string 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.
MAX (expression [,expression] . . .)	Obtains the maximum value among the set of arguments.
MIN (expression [,expression] . . .)	Obtains the minimum value among the set of arguments.

Figure 11. PL/S II Built-In Functions

PL/S II also has special statements corresponding to selected machine instructions. The general form is:

```
instruction-name [(operand[,operand] . . . )] ;
```

The instruction determines the operand requirements, with PL/S II assisting in such things as addressability and register use.

The GENERATE Statement

The GENERATE statement provides the ability to insert assembler instructions into the PL/S II procedure. The 'simple' GENERATE includes the assembler text in parentheses. The 'block' GENERATE (one without parenthesized text) inserts following lines until the ENDGEN control statement appears.

GENERATE may be used ahead of the PROCEDURE statement for the external procedure. In this situation, it is being used to insert necessarily-first assembler instructions, such as macro definitions or special prologues.

The DATA keyword indicates that the GENERATE is defining data. The data is declared with the GENERATED attribute, and may be STATIC LOCAL or AUTOMATIC. The DEFS keyword on the GENERATE statement says which items it defines and determines which data area to insert the definitions in.

A GENERATE statement inserting code can have keywords similar to the OPTIONS keywords:

EXIT	notes that the inserted instructions may exit the procedure.
FLOWS	notes possible continuation point labels.
NOSEQFLOW	notes that simple continuation flow will not occur.
REFS	notes which data may be referenced.
SETS	notes which data may be assigned new values.

Figure 12. GENERATE Statement Keywords

PL/S II programs are documented by means of the listings shown in Figure 13.

PL/S II Compilation Options
PL/S II Source Statements
PL/S II Attribute and Cross-Reference Table
PL/S II Messages
Assembler External Symbol Dictionary
Assembler Source Instructions
Assembler Relocation Dictionary
Assembler Cross-Reference

Figure 13. Sequence of Listings for a PL/S II Program

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

- A PL/S II Compilation Options Page.
- A PL/S II Source Statement Listing.
- A PL/S II Attribute and Cross-Reference Table.
- A PL/S II Listing with the Segmented Source Option.
- An Assembler Source Listing.

PL/S II Compilation Options Listing

```
PL/S II VERSION 6.1M                                    PAGE 1     TIME 13.34 DATE 74.049
 A                                                       F          G          H


          B
          C COMPILATION CONTROL CARDS
@PROCESS TITLE('SAMPLE PROGRAM') ;

             COMPLETE COMPILATION OPTIONS
               OPTIONS
               SIZE (MAX)   ACTUAL SIZE=149960 E
               BUFSIZE (010240)
               EXTEND (000000)
               MPERCENT(20)
               MARGINS (02,72)
               GENMARGINS (02,72)
               LINECOUNT (00060)
               FLAG (I)
               NOTERM
               NOMONITOR
               NOCOMPATIBLE
             D CONCHAR ('@')
               MACRO
               NOMSOURCE
               NOMDECK
               NOMXREF
               NODCLMAP
               NOFORMAT
               COMPILE
               NOANALYZE
               SOURCE (NOSEGMENT)
               XREF ( NUM)
               ASSEMBLE
               NODECK
               ANNOTATE
               MACHINE (S360)
               OPTIMIZE (3,SPACE)
               NOIDR
               ATITLE
               NOADEFS
```

Figure 14. PL/S II Options Used Listing

A The common heading information line in Figure 14 appears on each page of the information listing. The first heading item is the name and the version of the compiler used.

B The heading, 'INVOCATION PARM FIELD', would appear in this listing if options were passed in the PARM field of an EXEC statement invoking the compiler. The options appearing in the PARM field are printed following the heading.

C The heading 'COMPILATION CONTROL CARDS' appears when control cards have been used. The control cards are listed following the heading.

D Appearing at this point is a list of the PL/S II options in effect for the current compilation. Options are listed in phase related groups. When a phase is not executed, for example the format phase in the figure, other options concerning that phase would be meaningless and are not listed.

E When SIZE (MAX) is specified, the compiler supplies the ACTUAL SIZE=nnnnnn entry to indicate the number of bytes actually used.

F The page number is incremented by one for each new page.

G TIME indicates the time of day when the current compilation occurred. The time is based on the 24 hour clock.

H DATE indicates the year and day when the compilation occurred.

PL/S II Source Statement Listing

```
PL/S II VERSION 6.1M SAMPLE PROGRAM                                                                      PAGE 2    TIME 13.34 DATE 74.049
IF DO LINE STMT -L-------1---------2---------3-SOURCE--4---------5---------6---------7-R-------8  LINE CROSS-REFERENCE
A  B   C    D         E                              F                                         G                H
         1    1 MAIN:   PROCEDURE;                                                                    1
         2    2         DECLARE /*VARIABLE DATA ITEMS FOR THIS PROCEDURE*/
         3                  BUF CHAR(80),              /*INPUT CARD BUFFER          */                3
         4                  OUT CHAR(121),             /*OUTPUT LINE BUFFER         */                4
         5                  /* RETURN CODE VARIABLE */
         6                  CODE FIXED(31),            /* CODE SET BY READCARD      */                6
         7                  I FIXED(31) INIT(2);       /* INDEX TO OUTPUT LINE      */                7
         8    3         DECLARE /* ROUTINES CALLED */
         9                  READCARD ENTRY,            /* READS IN A CARD           */                9
        10                  PRINT ENTRY;               /* PRINTS A LINE             */                10
        11    4         /* OBTAIN AN INPUT CARD */
        12      OBTAIN: CALL READCARD(BUF,CODE);       /* GET A CARD, AND SET CODE:                   12,9,3,6
        13                                                =0, NORMAL READ
        14                                                =1, END-OF-FILE
        15                                                =2, ERROR                 */
        16    5         /* CHECK CODE FOR VALIDITY     */
        17              IF CODE=0 THEN                 /* VALID INPUT               */                6
1       18    6           /* PRINT OUT THIS CARD AND KEEP GOING                     */
1       19                DO;
1  1    20    7           OUT(1)=' ';                  /* SET FOR SINGLE SPACING    */                4
1  1    21    8           OUT(I:I+80)=BUF;             /* MOVE CARD TO OUTPUT LINE  */                4,7,7,3
1  1    22    9           CALL PRINT(OUT);             /* OUTPUT THE CARD           */                10,4
1  1    23   10           GO TO OBTAIN;                /* CONTINUE WITH NEXT CARD   */                12
1  1    24   11           END;
        25   12         ELSE                           /* NO MORE INPUT OR ERROR    */
1       26                RETURN;                      /* RETURN TO CALLING PROGRAM */
        27   13         END MAIN;                      /* END OF THE PROCEDURE      */                1
```

Figure 15. PL/S II Source Statement Listing

A The IF column count in Figure 15 increases by one each time a THEN path or ELSE path of an IF statement is encountered. The count decreases by one each time the related path is completed.

B The DO column count increases by one for each DO statement encountered. The count decreases by one for each END statement that closes a DO group.

C The LINE column contains consecutively numbered entries for each line in the source listing.

D The STMT column contains consecutively numbered entries for each statement in the source listing.

E L identifies the left margin of the input source statements.

F SOURCE identifies the field where the input source statements are listed. It is marked off in multiples of ten.

G R identifies the right margin of the input source statements.

H This field provides a cross-reference between the variables in the associated statements and the points in the source listing where they are first defined. If XREF(NUM) is specified, all references are in terms of line numbers; the heading is 'LINE CROSS REFERENCE.' If any form of XREF(STMT) is used, all references are in terms of statement numbers; the heading is 'STMT CROSS REFERENCE.'

PL/S II Attribute and Cross-Reference Table

```
PL/S II VERSION 6.1M SAMPLE PROGRAM                                    PAGE 3    TIME 13.34 DATE 74.049
A DECLARED  B NAME      C ATTRIBUTES AND LINE REFERENCES

  3           BUF       D LOCAL CHARACTER(80) BOUNDARY(BYTE)
                        E  12 21
  6           CODE        LOCAL BINARY FIXED(31) BOUNDARY(WORD)
                          12 17
  7           I           LOCAL BINARY FIXED(31) BOUNDARY(WORD) INITIALIZED
                          21 21
  1           MAIN        LOCAL ENTRY EXTERNAL
                          27
  12          OBTAIN      LOCAL LABEL
                          23
  4           OUT         LOCAL CHARACTER(121) BOUNDARY(BYTE)
                          20 21 22
  10          PRINT       NONLOCAL ENTRY EXTERNAL
                          22
  9           READCARD    NONLOCAL ENTRY EXTERNAL
                          12

          F  029730 BYTES WERE ALLOCATED FOR MPERCENT
                 000556 BYTES WERE USED BY MPERCENT
             104048 BYTES WERE ALLOCATED FOR MACRO DEFINITIONS AND INVOCATIONS
                 000000 BYTES WERE USED BY MACRO DEFINITIONS AND INVOCATIONS
             00019968 BYTES OF THE SIZE OPTION WERE USED IN THE COMPILE PHASE
                 00019968 BYTES IS MINIMUM POSSIBLE COMPILER SIZE
             0008 VARIABLES WERE USED IN THIS COMPILATION
             0000 VARIABLES WERE IMPLICITLY DEFINED AS FIXED(31) OR PTR(31) IN THIS COMPILATION
             0000 VARIABLES WERE UNREFERENCED IN THIS COMPILATION
             000 WAS THE DEEPEST LEVEL OF MACRO NESTING
             000 WAS THE DEEPEST LEVEL OF INCLUDE NESTING
             001 WAS THE DEEPESE LEVEL OF PROC NESTING
             001 WAS THE DEEPEST LEVEL OF DO NESTING
             001 WAS THE DEEPEST LEVEL OF IF NESTING
```

Figure 16. PL/S II Attributes and Cross-Reference Table

A The DECLARED column identifies the statement or line numbers in which the named variables are declared. The term IMPLICIT will appear in this column for implicitly defined variables such as a CALL target variable which is not declared. The term BUILT-IN appears in this column when the name used is a built-in function.

B The NAME field contains a listing of variable names and built-in functions used in the source program.

C The term 'STATEMENT REFERENCES' appears in this heading when the XREF(STMT) option is used. If the XREF(NUM) option is used, the term 'LINE REFERENCES' appears instead.

D The first set of lines in this field, opposite its associated variable name, describes the attributes of that variable. For built-in functions the attribute is BUILT-IN.

E The second set of lines in this field lists all non-declaration references to the associated variable found in the source program. The references are line numbers if XREF(NUM) is in effect; for XREF(STMT) they are statement numbers.

F Notes pertaining to the compilation appear at the end of the listing.

The attribute and cross reference table is usually followed by a table of unreferenced variables. These variables are components of structures defined in the PL/S II 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:

LINE	NAME	LINE	NAME	...
37	CBFXYY	417	OCBF	...
38	CBFXYZ	423	OCBFTTY	...
	.	.		.
	.	.		.
	.	.		.

The LINE field references the PL/S II source line defining the items. If cross-reference were by statement, the heading would be STMT and it would reflect the defining statement.

PL/S II Segmented Source Listing

```
PL/S II VERSION 4.0   SAMPLE PROGRAM                                    PAGE 2   TIME 03.01 DATE 72.227
IF DO LINE STMT L--------1---------2---------3-SOURCE--4---------5---------6---------7-R-------8   LINE CROSS REFERENCE

        1,   1 P:PROC;                                                                    1.1
         2    2 DCL A FIXED(31);                                                          1.2
         3    3 A=1;                                                                      1.2
         4      @MACRO INCLUDE SYSLIB(M1); E      /* A COMPILER INSERT          */      *LISTING AT 2.5
        12    8 A=6;                                                                      1.2
        13    9 END P;                                                                    1.1
```

```
PL/S II VERSION 4.0   SAMPLE PROGRAM                                    PAGE 3   TIME 03.01 DATE 72.227
IF DO LINE STMT L--------1---------2---------3-SOURCE--4---------5---------6---------7-R-------8   LINE CROSS REFERENCE

         4      @MACRO INCLUDE SYSLIB(M1);        /* A COMPILER INSERT          */      START SEGMENT 2.5
         5    4 A=2;                                                                      1.2
         6      @INCLUDE SYSLIB(M2);                                                      START SEGMENT 3.7
        10    7 A=5;                                                                      1.2
        11      @ENDINCLUDE; F                    /* A COMPILER INSERT          */      END SEGMENT 2.5
```

```
PL/S II VERSION 4.0  SAMPLE PROGRAM                                     PAGE 4   TIME 03.01 DATE 72.227
IF DO LINE STMT L--------1---------2---------3-SOURCE--4---------5---------6---------7-R-------8   LINE CROSS REFERENCE

         6      @INCLUDE SYSLIB(M2);                                                      START SEGMENT 3.7
         7    5 A=3;                                                                      1.2
         8    6 A=4;                                                                      1.2
         9      @ENDINCLUDE;                      /* A COMPILER INSERT          */      END SEGMENT 3.7
```

```
PL/S II VERSION 4.0     SAMPLE PROGRAM                                  PAGE 5     TIME 03.01 DATE 72.227
DECLARED     NAME         ATTRIBUTES AND LINE REFERENCES

1.2          A            LOCAL BINARY FIXED(31) BOUNDARY(WORD)
                       G  1.3  2.5  3.7  3.8  2.10  1.12
1.1          P            LOCAL ENTRY EXTERNAL
```

Figure 17. PL/S II Segmented Source Listing

Figure 17 shows four pages of a segmented source listing. Segmented listings aid structured programming by listing selected text segments on individual pages. They are requested by the SOURCE (SEGMENT) compiler option.

A If SOURCE (SEGMENT) is specified, all side cross references and references in the Attribute and Cross Reference listing are of the form n.n. The number before the point is a level number. The number after the point is a statement or line number.

B "*LISTING AT 2.5" points to the segment containing the text included by "@MACROINCLUDE SYSLIB(M1);".

C "START SEGMENT 2.5" appears as the first line in the side cross reference on the page containing the included text. The end of the included text is marked by "END SEGMENT 2.5".

D Nested INCLUDEs generate higher level segments. "@INCLUDE SYSLIB(M2);" found within segment 2.5 generates segment 3.7. Segments are listed in level number order, e.g. , segments at the 3 level are listed after all segments at the 2 level.

Note: Later compiler versions replace the point in a reference by an L if a line reference, or by an S if a statement reference.

E "@MACROINCLUDE SYSLIB(M1);" is a compiler generated statement. The macro INCLUDE that it replaces is processed by the macro facility before compile time and is not available when the source listing is produced. Note that the INCLUDE statement appears both at the point of the INCLUDE and as the first statement on the page containing the included text.

F "@ENDINCLUDE;" is a compiler generated statement. It appears as the last statement on the page for each included segment. The END SEGMENT n.n identifier appears on the same line.

G To locate a line or statement referenced in the Attribute and Cross Reference listing or as a side cross reference, use the segment identifiers (START SEGMENT n.n). All references to level 1 (1.n) are in the first segment and preceed the first segment identifier. For levels higher than 1 there may be many segments having the same level number. To find a referenced item, locate the first segment having the referenced level number, then scan the line or statement numbers.

Assembler Listing This listing, shown in Figure 18, is a listing produced by assembling the output of the PL/S II compiler.

```
        SAMPLE PROGRAM                                                                          PAGE    2

  LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                F15OCT70   2/18/74
                                                                                                  C
000000                                2 MAIN     CSECT ,                                        0001 00003000
000000 90EC D00C            0000C     3 @PROLOG  STM   @14,@12,12(@13)                          0001 00004000
000004 05C0                           4          BALR  @12,0                                    0001 00005000
000006                                5 @PSTART  DS    0H                                      0001 00006000
000006                                6          USING @PSTART,@12                              0001 00007000
000006 50D0 C062            00068     7          ST    @13,@SA00001+4                           0001 00008000
00000A 41E0 C05E            00064     8          LA    @14,@SA00001                             0001 00009000
00000E 50E0 D008            00008     9          ST    @14,8(,@13)                              0001 00010000
000012 18DE                          10          LR    @13,@14                                  0001 00011000
                                     11 *        /* OBTAIN AN INPUT CARD */                   0004 00012000
                                     12 *OBTAIN: CALL READCARD(BUF,CODE);       /* GET A CARD, AND SET CODE:    00013000
                                     13 *                                           =0, NORMAL READ          0004 00014000
                                     14 *  A                                        =1, END-OF-FILE          0004 00015000
                                     15 *                                           =2, ERROR                  */ 00016000
000014 58F0 C0A6            000AC    16 OBTAIN  L     @15,@CV00040                             0004 00017000
000018 4110 C052            00058    17          LA    @01,@AL00004                             0004 00018000
00001C 05EF                          18          BALR  @14,@15                                  0004 00019000
                                     19 *        /* CHECK CODE FOR VALIDITY     */             0005 00020000
                                     20 *        IF CODE=0 THEN                 /* VALID INPUT              */ 00021000
00001E 58F0 C0B2            000B8    21          L     @15,CODE                                 0005 00022000
000022 12FF                          22          LTR   @15,@15                                  0005 00023000
000024 4770 C046            0004C    23          BNZ   @RF00005                                 0005 00024000
                                     24 *          /* PRINT OUT THIS CARD AND KEEP GOING                 */ 00025000
                                     25 *          DO;                                          0006 00026000
                                     26 *          OUT(1)=' ';                  /* SET FOR SINGLE SPACING   */ 00027000
000028 9240 C10A      00110          27          MVI   OUT,C' '                                 0007 00028000
                                     28 *          OUT(I:I+80)=BUF;             /* MOVE CARD TO OUTPUT LINE */ 00029000
00002C 5810 C0B6            000BC    29          L     @01,I                                    0008 00030000
000030 41F1 C109            0010F    30          LA    @15,OUT-1(@01)                           0008 00031000
000034 9240 F050      00050          31          MVI   80(@15),C' '                             0008 00032000
000038 D24F F000 C0BA 00000 000C0    32          MVC   0(80,@15),BUF                            0008 00033000
                                     33 *          CALL PRINT(OUT);             /* OUTPUT THE CARD          */ 00034000
00003E 58F0 C0AA            000B0    34          L     @15,@CV00041                             0009 00035000
000042 4110 C05A            00060    35          LA    @01,@AL00009                             0009 00036000
000046 05EF                          36          BALR  @14,@15                                  0009 00037000
                                     37 *          GO TO OBTAIN;                /* CONTINUE WITH NEXT CARD  */ 00038000
000048 47F0 C00E            00014    38          B     OBTAIN                                   0010 00039000
                                     39 *          END;                                         0011 00040000
                                     40 *        ELSE                           /* NO MORE INPUT OR ERROR   */ 00041000
                                     41 *          RETURN;                      /* RETURN TO CALLING PROGRAM */ 00042000
                                     42 *  B     END MAIN;                      /* END OF THE PROCEDURE     */ 00043000
00004C 58D0 D004            00004    43 @EL00001 L     @13,4(,@13)                              0013 00044000
000050                               44 @EF00001 DS    0H                                       0013 00045000
000050 98EC D00C            0000C    45 @ER00001 LM    @14,@12,12(@13)                          0013 00046000
000054 07FE                          46          BR    @14                                      0013 00047000
000056                               47 @DATA    DS    0H                                             00048000
000058                            D  48          DS    0F                                             00049000
000058 000000C0                      49 @AL00004 DC    A(BUF)                 LIST WITH   2 ARGUMENT(S)  00050000
00005C 000000B8                      50          DC    A(CODE)                                         00051000
000060 00000110                      51 @AL00009 DC    A(OUT)                 LIST WITH   1 ARGUMENT(S)  00052000
000064                               52          DS    0F                                              00053000
000064                               53 @SA00001 DS    18F                                             00054000
0000AC                               54          DS    0F                                              00055000
0000AC 00000000                      55 @CV00040 DC    V(READCARD)                                     00056000
0000B0 00000000                      56 @CV00041 DC    V(PRINT)                                        00057000
```

Fig. 8. Assembler Listing (Part 1 of 2)

```
        SAMPLE PROGRAM                                                                                    PAGE    3

 LOC  OBJECT CODE    ADDR1 ADDR2  STMT   SOURCE STATEMENT                                    F15OCT70   2/18/74

0000B8                            57           DS    0D                                                00058000
0000B8                            58 CODE      DS    F                                                 00059000
0000BC 00000002                   59 I         DC    F'2'                                              00060000
0000C0                            60 BUF       DS    CL80                                              00061000
000110                            61 OUT       DS    CL121                                             00062000
000000                            62 @00       EQU   00          EQUATES FOR REGISTERS 0-15          00063000
000001                            63 @01       EQU   01                                              00064000
000002                            64 @02       EQU   02                                              00065000
000003                            65 @03       EQU   03                                              00066000
000004                            66 @04       EQU   04                                              00067000
000005                            67 @05       EQU   05                                              00068000
000006                            68 @06       EQU   06                                              00069000
000007                            69 @07       EQU   07                                              00070000
000008                            70 @08       EQU   08                                              00071000
000009                            71 @09       EQU   09                                              00072000
00000A                            72 @10       EQU   10                                              00073000
00000B                            73 @11       EQU   11                                              00074000
00000C                            74 @12       EQU   12                                              00075000
00000D                            75 @13       EQU   13                                              00076000
00000E                            76 @14       EQU   14                                              00077000
00000F                            77 @15       EQU   15                                              00078000
00004C                            78 @RF00005  EQU   @EL00001                                        00079000
000189                            79 @ENDDATA  EQU   *                                               00080000
000000                            80           END   MAIN                                            00081000
```

Figure 18. Assembler Listing (Part 2 of 2)

A The PL/S II source statements producing executable instructions appear as comments ahead of the generated code. A label on a PL/S II 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 II 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 basic format shown in Figure 19. (Note: for REENTRANT programs, some definitions for the fixed and dynamic areas may be interpersed.)

Reentrant Program	Non-reentrant Program
@DATA – constants, including address constants and static argument lists. – user normal STATIC LOCAL data. – user GENERATED STATIC LOCAL data. – BASED and DEFINED (on STATIC) name definitions. – register definitions. – label equates. @ENDDATA @DATD – save areas. – compiler temporary data. – user normal AUTOMATIC data. – user GENERATED AUTOMATIC data. – DEFINED (on AUTOMATIC) name definitions. @ENDDATD	@DATA – constants, including address constants and argument lists. – save areas. – compiler temporary data. – user normal data. – user GENERATED data. – BASED and DEFINED name definitions. – register definitions. – label equates. @ENDDATA

Figure 19. Data Area Layout

Compiler-Generated Labels

The labels (statement identifiers) that appear in the PL/S II 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 II program into assembler code. To help you identify various items in the assembler code, the conventions for compiler-generated labels are listed in Figure 20.

Label	Function
@00 to @15	General Registers 0-15
@AFTEMPS	Special Argument Temporaries
@ALn	Argument List (Start)
@CAdidx	Address Constant
@CBdidx	Bit Constant (including hex)
@CCdidx	Character Constant
@CFdidx	Fixed Constant (except halfword)
@CHdidx	Halfword Fixed Constant
@CVdidx	V-type Address
@DATA	Start of Static Data
@DATD	DSECT for Dynamic Data
@DBsn	Do loop Body (if loop also has WHILE)
@DCsn	Continuation Statement Past DO Loop
@DEsn	End-of-loop Text
@DLsn	Do loop Body (Statements in Loop or WHILE test)
@ECsn	Continuation Statement Past Entry
@EFpn	Label in Epilog
@ELpn	Epilog Corresponding to Simple RETURN
@ENDDATA	End of the Static Area
@ENDDATD	End of the Dynamic Area
@EPsn	Unique Portion of Entry Prologue
@ERpn	Label in Epilog
@GLn	Generated Label for a complex relational expression or a built-in function.
@MAINENT	Main Entry Point
@NMn	Substitute name supplied if no name or duplicate name is found
@PBpn	Continuation Statement past internal procedures
@PCpn	Copy of Parameter List Pointers
@PROLOG	Common Prolog
@PSTART	Start of Normal Addressability
@RCsn	Continuation Statement for relational expression when THEN and ELSE paths merge after an IF statement.
@RFsn	False or ELSE Path after an IF statement
@RTsn	True or THEN Path after an IF statement
@SApn	Save Area
@SCdidx	CLC Execute Target
@SIZDATD	Size of Dynamic Area
@SMdidx	MVC Execute Target
@SNdidx	NC Execute Target
@SOdidx	OC Execute Target
@SXdidx	XC Execute Target
@TFn	Fullword Temporary
@TSn	String Temporary
@ZLEN	Length of Zero Temporaries
@ZTEMPS	Start of Zero Temporaries
@ZT00001	Fullword Temporary with Hi-Order byte=0.
@ZT00002	Fullword Temporary with Two Hi-Order bytes=0.
@ZT00003	Fullword Temporary with Three Hi-Order bytes=0.

Notes:

'n' is a five-digit decimal number from 00001 to 99999.

'sn' is the five-digit statement number for D labels, E labels, and R labels, (of the DO or ENTRY or IF statements respectively).

'pn' is the five-digit procedure number (00001 to 00255) assigned by the compiler.

'didx' is the five-digit dictionary index of a constant or execute target.

Figure 20. Labels Generated by the Compiler

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 II 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 II 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 II compilations.

Modifying Instructions

1. Do not make references to compiler-generated labels (shown in Figure 20, Section 3). These labels may change when a PL/S II module is re-compiled. The compiler generates labels that begin with the @ character; your own assembler labels should have a non-conflicting rule such as 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 II compiler may define data to the assembler by means of equate statements or other statements not establishing the expected length.
3. Use explicit base values when making references to parameter lists, BASED variables, or NONLOCAL variables. This is necessary because the PL/S II compiler defines these data classes to the assembler by means of equate statements, equating the item to its displacement from a base point.
4. Do not insert any new assembler instructions into an instruction sequence generated for a single PL/S II statement. Make insertions either before or after the generated instruction sequence.
5. Protect your registers around any inserted code. This is desirable because of the compiler flexibility in changing the registers it uses.
6. Keep your modifications together, separate from the insertion points, by making the modification a macro and the actual insertion an invocation, or by surrounding each unit by an ANOP and AGO and making the actual insertion a corresponding AGO and ANOP. This will lessen the effect of sequence numbers changing in a subsequent release.
7. Develop modifications as separate modules.

Modifying Data

Add new data at the end of the generated assembler data area. The symbol @ENDDATA on the assembler listing identifies the end of the STATIC area, and @ENDDATD identifies the end of the AUTOMATIC area (see Figure 19, Section 3 for a description of these areas.) You should insert the additional data just ahead of these symbols.

If you insert new data elsewhere, make sure that you don't affect expected alignment of data that follows. Similarly, if you change the length of a data item, make sure that alignments are preserved. You should also check for displacements becoming greater than 4095, and modify references accordingly.

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 II REGISTER variables by the PL/S II 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 check alignment and displacement of following data.

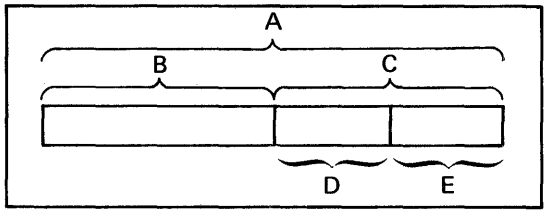
If you change the length of a component of a STATIC, AUTOMATIC, or BASED structure, you must change the length in all instructions that reference:

- 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 containing 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.

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.

assignment statement: A statement used to provide a new value for a data variable.

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

AUTODATA: A procedure option controlling the use of dynamic storage.

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: An expression facility of PL/S II that provides a desired value.

built-in instruction: One of the special statements which correspond directly to a selected machine instruction.

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. The compiler chooses a register 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.

DATA: A keyword on a GENERATE statement allowing data definitions .

DATAREG: See "data register".

data register: A register used in a reentrant environment to address data. The compiler chooses a register unless the DATAREG option on the PROCEDURE statement indicates otherwise.

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

DEFINED: A data attribute indicating an overlay for a data item.

DEFS: A keyword on a GENERATE statement indicating what items it defines.

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

dimension: The number of elements in a one-dimension array, or for multi-dimension arrays, a partition of the elements.

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 comparison 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.

EXIT: A keyword for the GENERATE statement and the OPTIONS attribute indicating unusual exit from the current environment.

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.

FLOWS: A keyword for the GENERATE statement and the OPTIONS attribute indicating possible continuation labels.

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

INCLUDE statement: A compiler statement to obtain text for compilation from a library.

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.

iteration: Repeated execution of a DO group.

keyword: A symbol that identifies a data attribute, a PL/S II 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.

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

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

LOCATION: A data attribute for an item at an absolute location.

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.

MAX: A built-in function to obtain the maximum of a set of values.

microfiche: Microfilm containing program listings.

MIN: A built-in function to obtain the minimum of a set of values.

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.

NOSEQFLOW: A keyword for the GENERATE statement and the OPTIONS attribute indicating that simple continuation flow does not occur.

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.

OPTIONS: A keyword for the PROCEDURE statement or an attribute for ENTRY data noting special requirements and actions.

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.

REFS: A keyword for the GENERATE statement and the OPTIONS attribute indicating what data may be referenced.

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.

RETURN: The PL/S II 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.

SETS: A keyword for the GENERATE statement and the OPTIONS attribute indicating what data may be assigned new values.

SIGNED: An attribute of arithmetic data which can be negative.

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.

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 control variable's limit value.

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

UNSIGNED: An attribute of arithmetic data which cannot be negative.

UNTIL: A DO statement keyword that specifies a terminating condition for iteration.

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

VLIST: A keyword used in the OPTIONS attribute for an entry 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.

WHILE: A DO statement keyword that specifies a necessary condition for iteration.

Guide to PL/S II

GC28-6794-0

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 your address 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)

Guide to PL/S II Printed in U.S.A. GC28-6794-0

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)