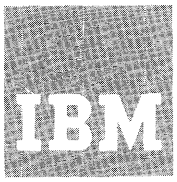
File No. S360-29
Order No. GC33-2002-2



Systems Reference Library

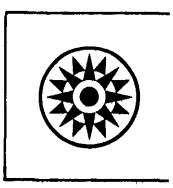
IBM System/360 Conversion Aids:
FORTRAN IV-to-PL/I Language Conversion Program for IBM System/360 Operating System

Program Number 360C-CV-710

The System/360 FORTRAN IV-to-PL/I Language Conversion Program (LCP) assists in the transition to PL/I by converting FORTRAN IV programs into PL/I programs. The LCP is distributed in object module form for inclusion in the user's system library.

The user should have an understanding of the System/360 Operating System and be familiar with the following publications:

IBM System/360 FORTRAN IV Language, Form C28-6515
IBM System/360 Basic FORTRAN IV Language, Form C28-6629
IBM System/360 Operating System, PL/I (F) Programmer's Guide, Form C28-6594
A Guide to PL/I for FORTRAN Users, Form C20-1637
IBM System/360, PL/I Reference Manual, Form C28-8201



Second Edition (January 1973)

This is a revision of GC33-2002-1. It incorporates changes issued in the following Technical Newsletters:

N33-7002 (dated October 4, 1968)
N33-7004 (dated January 15, 1969)
GN33-7007 (dated August 13, 1970).

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form for reader's comments appears at the back of this publication. Address any additional comments concerning the contents of this publication to: IBM France, Centre d'Etudes et Recherches, Programming Publications, 06610 - La Gaude, France.

CONTENTS

TABLES

The FORTRAN IV-to-PL/I Language Conversion Program, referred to in this publication as "the LCP," is a program provided by IBM to assist its customers in the transition from FORTRAN IV to PL/I. The LCP can be added to the user's System/360 Operating System program library to help him convert his FORTRAN IV source programs into PL/I programs. The resulting PL/I program is referred to as the PL/I "target" program.

The LCP does the following:

- It recognizes and converts FORTRAN IV statements into PL/I statements having the same meaning and effect.
- It detects and flags FORTRAN IV statements that have no PL/I equivalents or that cannot be meaningfully or unambiguously translated into PL/I statements.
- It produces an output listing of the PL/I program as well as messages providing information on the conversion actions. The user may specify that the listing also contain the original FORTRAN IV statements.
- It produces, when specified to do so by the user, the converted program on cards, or as card images on tape or on disk.

In certain programs, limited manual changes will be necessary to make the generated PL/I program compilable and faithful to the FORTRAN IV source program. In such cases, the user will be guided by output-listing messages.

SOURCE LANGUAGE

The LCP processes programs written in System/360 Operating System FORTRAN IV language. It will, therefore, process the output of the current FORTRAN II to System/360 Operating System FORTRAN IV Language Conversion Programs for the IBM 1401 (see IBM System/360 Conversion Aids: FORTRAN II Language Conversion Program for the IBM 1401, Form C28-6560).

Moreover, the LCP will convert FORTRAN IV source programs written for current IBM systems other than the System/360. When such source programs are converted, however, the user should note the two following points:

1. From the point of view of language, all FORTRAN IV source programs can be converted by the LCP, subject to the restrictions indicated in Appendix C.
2. There is no guarantee that the converted programs will be correctly executed. This is due to differences in implementation between System/360 and other current IBM systems (storage allocation, magnitude of data, etc.).

OUTPUT LANGUAGE

The LCP converts source programs into System/360 Operating System PL/I language for compilation by the PL/I (F) compiler version 4 and the following versions.

DISTRIBUTION OF THE LCP

The LCP will be distributed in object module form for inclusion in user's system library.

SAMPLE PROGRAM

The disk pack or the tape distributed by IBM will contain, in addition to the LCP, a sample program, written in FORTRAN. This program is described in detail in Appendix H.

SYSTEM REQUIREMENTS

The machine requirements depend on the type of run to be made: a conversion run or a link-editing run for generating the LCP load module.

The minimum System/360 configuration required for a conversion run by the LCP is:

- One System/360 Model 40 with 128K bytes of main storage. The LCP itself needs a minimum of 70K bytes to operate in a PCP or MFT environment. The 70K bytes include the Data Management Routines and buffers. To use the LCP with MVT, it is suggested that 6K be added to the SIZE chosen so as to obtain the REGION specification.
- Standard instruction set
- Decimal Arithmetic feature
- Floating-Point Arithmetic feature
- Minimum peripheral equipment required by the Operating System

The logical data sets used by the LCP are shown in Table 1. Note that when these data sets are on DASD, they may be placed on the same volume as the system residence.

Table 1. Logical Data Sets Required by the LCP for a Conversion Run

DATA SET	FUNCTION	DEVICE OPTIONS
SYSIN	Source Input	Magnetic Tape Unit, Card Reader, Direct Access Storage Device (DASD)
SYSIOERR	Message Output	Magnetic Tape Unit, Printer, DASD
SYSPRNT	Listing Output	Magnetic Tape Unit, Printer, DASD
SYSUT1 SYSUT2	Auxiliary Storage Auxiliary Storage	DASD DASD
SYSPCH[1]	Deck Output	Magnetic Tape Unit, Card Punch, DASD

[1]SYSPCH is required only for punched-card output (or card image on magnetic tape or DASD) of the converted program.

For generation of the LCP load module, an additional 2311 Disk Storage Drive or magnetic tape unit is required to run the distributed program.

In order to generate the LCP load module and/or execute a conversion run, the Operating System must include the modules for the PL/I (F) compiler and its library.

CONTROL INFORMATION

A conversion run requires control cards prepared by the user, specifying which of the options provided by the LCP he has chosen. The control cards required and the options available are described in Appendix B.

GENERAL DESCRIPTION OF THE LCP

The LCP is designed to do a maximum amount of conversion. It provides a number of options that permit the user to apply it effectively to a wide range of conversion needs.

The program is particularly versatile in the following respects:

- Messages in the output listing identify statements that cannot be converted and those that, once converted, may give incorrect results on execution.
- In addition to the printed listing, the converted output may also appear on cards, magnetic tape, or DASD.
- The PL/I program can be generated in either the 48- or the 60-character set, depending on the character set used in the source program.
- Optionally, the FORTRAN source program can be listed.
- The user can specify the size of main storage available in the machine used for conversion.

- A character code option (BCD or EBCDIC) is provided, which remains the same for input and for output.

CHARACTERISTICS OF FORTRAN PROGRAMS TO BE CONVERTED

A source program to be converted by the LCP must be error-free; that is, source program statements must conform to the specifications for System/360 Operating System FORTRAN IV.

The source program can be in the form of punched cards or of 80-character (blocked or unblocked) card images on tape or on DASD.

Source programs should be converted by the LCP before any hand changes are made. This makes the best use of the LCP and avoids incorrect conversion caused by coding errors.

All FORTRAN source programs and subprograms are converted independently, except BLOCK DATA subprograms, which must immediately precede the main program to which they belong. Within a single given batch of FORTRAN programs, the user need not insert new LCP control statements unless he wishes to change original LCP control information.

LCP ACTIONS

The LCP analyzes each statement of the FORTRAN source program and takes one of three types of action:

- Full Conversion: The LCP converts the statement completely into a form acceptable to the PL/I (F) compiler.
- Conversion, with Warning: The statement is converted into a form acceptable to the PL/I (F) compiler. However, the execution of the target program may give results that are not equivalent to those obtained by the execution of the FORTRAN-compiled source program. A warning message is issued.
- No Conversion, with Warning: When a source program statement cannot be converted, a message identifies the statement and gives the reasons for non-conversion.

A list of restrictions is given in Appendix C.

USE OF THE OUTPUT LISTING

The output listing from the LCP always contains the converted statements and all messages that have been generated. Optionally, it may also contain the source program statements.

Using the output listing, the programmer can analyze the conversion and determine whether any manual changes are required. The necessary modifications can be made in the output deck. The deck is then ready for compilation by the PL/I (F) compiler.

PERFORMANCE

On a System/360 Model 50 with 128K bytes of main storage, the average conversion time (T) for a FORTRAN program containing N cards is given in the following formula (in seconds):

T=55+37*S+0.75*N

where S is the number of subprograms

The time given is that which is applicable when the user has specified the options SOURCE and DECK in his EXEC card, thus requiring the source program to be listed on SYSPRNT and the target program to be punched on SYSPCH. The devices to be used are as follows:

2540 Card Read Punch for SYSIN

1403 Printer for SYSPRNT

2540 Card Read Punch for SYSPCH

When SYSIN, SYSPRNT and SYSPCH are 2401 Magnetic Tape Units (Model 3):

T=55+37*S+0.60*N

NOTATION USED IN THIS DOCUMENT

The object of this paragraph is to provide a simple way of describing the conversion process, and not a comprehensive theory of languages.

In order to present the general form of a source language statement, as well as that of its target language equivalent, the following syntactic notation is used.

A syntactic variable of a language is used to represent one element of a particular set of elements of the language that have the same syntactic function. The range of values of a syntactic variable is therefore the set of elements.

In FORTRAN, for example, the statement numbers constitute a set of elements that have the same syntactic function. The corresponding syntactic variable has as its range of values the set of all possible FORTRAN statement numbers (each a sequence of from one to five decimal digits).

For the purposes of this publication, a syntactic variable is given a name, made up of a finite sequence of characters, which is a mnemonic representation of the corresponding syntactic function, e.g.:

statement number

In order to differentiate between syntactic variables that are not part of the language and basic symbols of the language, syntactic variables are enclosed in corner brackets , e.g.:

<statement number>

In addition, some keywords of the language (GO TO, IF, etc.) are used in connection with syntactic variables. For example, the FORTRAN arithmetic IF statement is written as follows:

IF(<arithmetic expression>)<statement number>,<statement number>,
<statement number>

When it is necessary to specify different values of a given syntactic variable, numeral suffixes are used. In order to specify in the example shown above that the three statement numbers are different, the following notation is used:

IF(<arithmetic expression>)<statement number 1>,<statement number 2>,
<statement number 3>

The operator conv is used to denote the result of the conversion of a FORTRAN syntactic variable into its PL/I equivalent:

conv<FORTRAN syntactic variable>

For example, the format of the FORTRAN unconditional GO TO statement and that of its PL/I equivalent are represented as follows:

GO TO<statement number> and GO TO conv<statement number>

The operator "conv" is sometimes omitted when there is no risk of confusion.

Conversion from FORTRAN IV to PL/I involves several general problems due to differences between the two languages.

The sections that follow address these problems and note those which the LCP can solve.

FORTRAN MATHEMATICAL FUNCTION SUBPROGRAM

The table in Appendix B shows the correspondence between the PL/I built-in function names and the FORTRAN IV mathematical function subprogram names.

Normally, the name of each FORTRAN IV mathematical function subprogram in the source program is changed to the corresponding PL/I built-in function name, except when suppression of such conversion has been indicated in an LCP control card (see Appendix E). For example, the function IABS is normally converted to ABS wherever it appears.

PREVENTION OF NAME CONFLICTS

To avoid name conflicts, the LCP provides an LCP substitution name in the following cases:

1. COMMON block name
   The name is lengthened 7 characters by concatenating the first characters of the word COMMON with the original common name (see the COMMON statement).
2. Other symbolic name
   The LCP checks each name written by the user against
   - a list of PL/I built-in function names which are different in FORTRAN or which are used for conversion by the LCP, namely, LOG, LOG10, TRUNC, MAX, MIN, FIXED, IMAG, BINARY
   - the operators of the 48-character set, if this option is selected, namely, GT, GE, NE, LT, LT, OR, AND, NOT, NL, NG, CAT, and PT

   If they match, a suffix <name-tail> is added to the name specified by the user. The value given to <name-tail> is as many characters of the string VARFUN as are required to make the substitution name seven characters long. For example, LOG will become LOGVARF, and GE will become GEVARFU. Optionally, a list of such source names and their corresponding LCP substitution names can be provided in the output listing.

ARRANGEMENT OF ARRAYS IN STORAGE

The manner of storing arrays differs in FORTRAN and in PL/I.

The FORTRAN convention is that the elements of an array are stored in ascending locations, with the value of the first subscript varying most rapidly, and that of each succeeding subscript varying less rapidly than that of its predecessor. Thus, a two-dimensional array is stored column by column in ascending locations.

The PL/I convention is the inverse of that for FORTRAN: the elements of an array are stored in ascending locations, but the value of the first subscript varies least rapidly, and the value of each subscript varies more rapidly than that of its predecessor. Thus, a two-dimensional array is stored row by row in ascending locations.

To conform with the PL/I convention for array storage, the LCP makes the necessary conversion, as shown in the following example:

Original

```
DIMENSION A(3,4,5,6)
...
...  = A(1,2,3,4)
...
```

Converted

```
DECLARE A(6,5,4,3) FLOAT BINARY;
...
...  = A(4,3,2,1);
...
```

The components of the FORTRAN IV language are discussed in approximately the order in which they are described in the publication IBM System/360 Operating System FORTRAN IV, Form C28-6515, i.e., under the following headings:

- General considerations
- Elements of the Language (Constants, Variables, Arrays)
- Arithmetic Expressions
- Logical Expressions
- Assignment Statements
- Control Statements (DO, GO TO, IF, etc.)
- Specification Statements (COMMON, DIMENSION, EQUIVALENCE, etc.)
- Statement Functions
- Function and Subroutine Subprograms
- Other FORTRAN statements accepted by the System/360 Operating System FORTRAN IV compiler
- Service Subprograms
- Input/Output Statements (READ, WRITE, FORMAT, etc.)

The LCP does not convert certain FORTRAN statements that are incompatible with PL/I. These statements are identified in the discussion of the category to which they belong.

DATA SET TERMINOLOGY

The terms "data set" and "data set reference number" are used in the discussion of input/output statements. In System/360 Operating System programming, the term "data set" refers to a named collection of data. A given data set may reside on one or more input/output units. A "data set reference number" refers to the data set itself, without regard to the input/output unit (or units) on which it resides.

FORM OF CODING EXAMPLES

Coding examples illustrate how the LCP converts a statement. The format of these examples is:

Original
FORTRAN coding as it would appear in the source program

Converted
Coding as it would appear in PL/I output from the LCP

Note that the LCP output displayed or discussed in this manual is assumed to be in the 60-character set version of PL/I. Thus, a semicolon is represented as it is on a standard typewriter keyboard, the relationship "less than" by the character < , etc. Each example or discussion, however, remains valid with respect to the 48-character set version of PL/I when all necessary replacements are made. For example, the following statement in the 60-character set:

```
A:  IF X>Y THEN IF Z = W THEN IF W<P
THEN Y = 1; ELSE P = Q; ELSE;
ELSE X = 4; J :  Z = 5;
```

would read, in the 48-character code version:

```
A..  IF X GT Y THEN IF Z = W THEN IF W LT P
THEN Y = 1,.  ELSE P = Q,.  ELSE,.
ELSE X = 4,.  J..  Z = 5,.
```

GENERAL CONSIDERATIONS

BLANKS WITHIN WORDS

FORTRAN IV permits embedded blanks. The LCP removes such embedded blanks except when they occur within literal constants.

Original
```
T O TAL = A + B + 72 .01 92E-2
```

Converted
```
TOTAL=A+B+72.0192E-2;
```

COMMENTS

All comments appearing in the FORTRAN program will be converted. The */ character sequence, where it occurs in the source program, will be converted into the *- sequence. Note that card columns 73 through 80 are not significant to the FORTRAN compiler and may be used for various purposes. The LCP ignores the contents of these columns and inserts an identification number in the PL/I target program.

Original
```
C     THIS IS A /*COMMENT*/
```

Converted
```
/*    THIS IS A /*COMMENT*-*/
```

ELEMENTS OF THE LANGUAGE

PROGRAM UNIT

A FORTRAN main program is converted to a PL/I main procedure. The following PL/I statement is created first, even if BLOCK DATA subprograms are placed before the main program:

```
(NOZERODIVIDE):MAINPRO:PROCEDURE OPTIONS(MAIN);
```

A FORTRAN subprogram is converted to a PL/I external procedure (see FUNCTION and SUBROUTINE statements).

The condition NOZERODIVIDE is created to simulate the effect of FORTRAN division by zero.

STATEMENT NUMBERS

If <statement number> has as its value a source program statement number, the LCP converts it into the PL/I statement label:

conv<statement number>

where conv<statement number> is EXTLAB followed by <statement number> without leading zeros.

Original
```
02045 A=B**C
```

Converted
```
EXTLAB2045:A=B**C;
```

INTEGER CONSTANTS

FORTRAN integer constants appear in the PL/I conversion in the same form as in the source text, after elimination of embedded blanks.

Original
```
5 2 31
```

Converted
```
5231
```

If FORTRAN integer constants appear as arguments passed to a subroutine or as arguments of a function, binary conversion is forced by using the built-in function BINARY at the time the function or subprogram is called.

Original
```
CALL SUB(472)
```

Converted
```
CALL SUB(BINARY(472));
```

REAL CONSTANTS

The following rules apply in the conversion of real constants:

- Embedded blanks are suppressed.
- D is changed into E.
- Trailing zeros are added, where necessary, to double-precision constants to make up the required seven significant digits.
- The exponent E0 is added as a suffix to decimal real constants with no exponent part.
- The built-in function FLOAT is used to force the conversion of FORTRAN single-precision real constants containing seven significant digits into PL/I single-precision floating-point constants.

Original
```
3857.  1517 92
```

Converted
```
3857.151792E0
```

Original
```
.  7634 E- 7
```

Converted
```
.7634E-7
```

Original
```
33671312.507941D+63
```

Converted
```
33671312.507941E+63
```

Original
```
1.234567
```

Converted
```
FLOAT(1.234567E0,6)
```

COMPLEX CONSTANTS

The same rules apply as for the conversion of real constants, with the following additions:

- A comma followed by a sign is suppressed, otherwise it is replaced by a + sign.
- The letter I is added as a final suffix.

Original
```
(-4.  7D+2,1.9736148)
```

Converted
```
(-4.700000E+2+1.9736148E0I)
```

LOGICAL CONSTANTS

A FORTRAN logical constant has one of two forms:

.TRUE. or .FALSE.

.TRUE. is converted to '1'B, .FALSE. to '0'B.

Original
```
A.AND..TRUE.
```

Converted
```
A&'1'B
```

LITERAL CONSTANTS

FORTRAN literal constants are reproduced without change in the PL/I output.

Original
```
'DON''T PRINT /X-COORDINATE'
```

Converted
```
'DON''T PRINT /X-COORDINATE'
```

FORTRAN literal constants should not be passed as actual parameters to subprograms. Should this happen, however, a warning message will be issued to the user.

HEXADECIMAL CONSTANTS

Hexadecimal constants are not converted.

SUBSCRIPTED VARIABLES

If the operator / or ** or a left parenthesis appears in a subscript, a warning message will be issued. It is the user's responsibility to check whether the converted subscript is correct, e.g., insert TRUNC for integer division, invert the order of subscripts for a subscripted variable appearing in a subscript.

ARITHMETIC EXPRESSIONS

FORTRAN arithmetic expressions undergo the following modifications:

- Elimination of blanks embedded in constants and identifiers
- Generation of LCP substitution names, where necessary
- Conversion of the exponent D into the exponent E

- Addition of trailing zeros
- Addition of the exponent part E0, where necessary
- Use of built-in functions TRUNC, BINARY, FLOAT

Original
```
CEIL**(ABE+2)/7 .98D- 1
```

Converted
```
CEIL**(ABE+2)/7.980000E-1
```

The built-in PL/I function TRUNC is used to force the results of division of PL/I fixed-point expressions to be of the same precision as the results of division of FORTRAN IV integer expressions.

The results of arithmetic expressions may differ from the expected results, owing to differences in the implementation of (1) precisions and (2) conversion of mixed characteristics.

For exponentiation, FORTRAN IV produces type integer if base and exponent are integer items; this is not always the case with PL/I. Note that, in this case, the conversion of FORTRAN integer arithmetic expressions containing exponentiation is not always correct.

LOGICAL EXPRESSIONS

Owing to differences between FORTRAN and PL/I in the relative priorities of operators, the LCP always inserts an additional pair of parentheses around the converted form of the expression dependant on the FORTRAN .NOT. operator.

Original
```
(E+9.5D2.GE.2*E).OR.(L.NE.3.14E-1)
```

Converted
```
(E+9.500000E2>=2*E)|(L¬=3.14E-1)
```

Original
```
(A**F.GT.ROOT).AND..NOT.(I.EQ.E)
```

Converted
```
(A**F>ROOT)&¬((I=E))
```

Original
```
A.GT.D**B.AND..NOT.L.OR.N
```

Converted
```
A>D**B&¬(L)|N
```

Original
```
.NOT.A.GT.B
```

Converted
```
¬(A>B)
```

Original
```
A.AND..NOT.  X+Y*Z.LE.  SIN(Z).OR.P
```

Converted
```
A&¬(X+Y*Z<=SIN(Z))|P
```

Original
```
A.AND..NOT.(B.OR..NOT.C.EQ.E)
```

Converted
```
A&¬((B|¬(C=E)))
```

Original
```
.NOT.(A+B).GT.C
```

Converted
```
¬((A+B)>C)
```

LOGICAL_ASSIGNMENT_STATEMENTS

Original
```
G=.TRUE.
```

Converted
```
G='1'B;
```

Original
```
BOOL(I,J)=(A**F.GT.ROOT).AND..NOT.P
```

Converted
```
BOOL(J,I)=(A**F>ROOT)&¬(P);
```

ARITHMETIC_ASSIGNMENT_STATEMENTS

Original
```
Y=C**(-Y)/.3D-5
```

Converted
```
Y=C**(-Y)/.3000000E-5;
```

Arithmetic_Assignment_with_Truncation_from_REAL_to_INTEGER

The value of an expression of type REAL is obtained using implementation defined precision which gives an interval that includes the true value. If this interval contains an integer value, the result of truncation is undefined.

For example:

2.000001 will give 2

for a true value of 2

1.999998 will give 1

The above is valid when using a language for which there are several implementations. It is all the more applicable when going from one language to another with a different implementation defined precision, even if this difference is slight.

A warning message will be issued.

CONTROL STATEMENTS

UNCONDITIONAL GO TO STATEMENT

FORTRAN Syntax
GO TO <statement number>

PL/I Syntax
GO TO EXTLAB<statement number>;

COMPUTED GO TO STATEMENT

For each computed GO TO statement, the LCP creates a one-dimensional array of the same size as the argument vector of the GO TO, with elements that are, in order, the statement numbers themselves. Thus, for the i-th GO TO statement taking the form shown, an array BRANCHi is generated and a PL/I declaration is created giving the following information:

- The dimension (n) of BRANCHi
- The LABEL attribute for BRANCHi
- The values of the elements of BRANCHi; that is, the converted statement numbers

FORTRAN Syntax
GO TO (<statement number 1>,...,<statement number n>),<index>

where <index> has as its value an unsubscripted integer variable, with values ranging from 1 to n.

PL/I Syntax
<label part> IF(<index> LE n AND <index> GT 0) THEN GO TO
BRANCHi(<index>),.

where <label part> is empty or takes the form EXTLABm.. (To avoid confusion, the above example is shown using the 48-character set.)

Thus, should the value of <index> fall outside the dimension of the array BRANCHi, the GO TO statement is not executed, and control passes to the following statement.

Original
```
      2  GO TO (25,10,7),ITEM
```

Converted (48-character)

```
        DECLARE BRANCH(3)LABEL INITIAL(EXTLAB25,EXTLAB10,EXTLAB7),.
          •
          •
          •
EXTLAB..IF (ITEM LE 3 AND ITEM GE 0) THEN GOTO BRANCH01(ITEM),.
```

ASSIGN AND ASSIGNED GO TO STATEMENTS

FORTRAN Syntax

ASSIGN <statement number>TO<unsubscripted integer variable>
...
...
...
GO TO <unsubscripted integer variable>,(<statement number 1>,...,
<statement number n>)

PL/I Syntax

<unsubscripted integer variable>=EXTLAB<statement number>;
...
...
...
GO TO <unsubscripted integer variable>;

Note: <unsubscripted integer variable> is given the LABEL attribute in a generated declaration unless this item appears in a specification statement in the FORTRAN program. A conflict will result when this item is used as an integer variable elsewhere in the program. Warning messages to that effect are issued for the converted ASSIGN and assigned GOTO statements.

ARITHMETIC IF STATEMENT

FORTRAN Syntax

IF (<arithmetic expression>)<statement number 1>,<statement number 2>,
<statement number 3>

PL/I Syntax

IF (<arithmetic expression>)=0 THEN GO TO conv<statement number 2>;
ELSE IF (<arithmetic expression>)>0 THEN GO TO conv<statement number 3>;
ELSE GO TO conv<statement number 1>;

If the label of the statement following the arithmetic IF statement is one of the three transfers, or if two of these three labels are the same, the conversion is optimized.

Original

```
        IF (A(J,K)**3-B) 10,4,30
    4   D=B+C
          •
          •
          •
   30   C=D**2
          •
          •
          •
   10   E=(F*B)/D+1
```

Converted

```
        IF (A(K,J)**3-B)<0 THEN GOTO EXTLAB10;
        ELSE IF(A(K,J)**3-B)>0 THEN GOTO EXTLAB30;
EXTLAB4:D=B+C;
          •
          •
          •
```

LOGICAL IF STATEMENT

FORTRAN Syntax

IF (<logical expression>)<statement>

PL/I Syntax

Note: If <statement> is STOP<integer part> and <integer part> is an integer constant, the PL/I Syntax is:

IF(<logical expression>)THEN DO;<statement>;END

DO STATEMENT

FORTRAN Syntax

DO<statement number><DO var>=<initial>,<final><increment>
<statements within the range of the DO>

In either of the two cases illustrated below, <increment option> is empty if <increment> is empty, or is BY <step> if <increment> is ,<step>.

1. Both <initial> and <final> have unsigned integer constants as their values.

   a. <initial> does not exceed <final>.

      PL/I Syntax

      DO<DO var>=<initial>TO<final><increment option>;
      conv<statements within the range of the DO>:
      END;

   b. <initial> exceeds <final>.

      PL/I Syntax

      DO<DO var>=<initial>;
      conv<statements within the range of the DO>:
      END;

2. Either <initial> or <final> has an integer variable as its value.

   PL/I Syntax

   DO<DO var>=<initial>TO MAX(<initial>,<final>)<increment option>;
   conv<statements within the range of the DO>;
   END;

Note: Whereas, in specific cases, FORTRAN allows a transfer out of the range of an innermost DO loop and a transfer back into the range of the loop, PL/I does not. (The PL/I (F) compiler will diagnose these transfers at compilation time.) The user should therefore ensure that his program contains no such transfer, and make the necessary hand changes.

CONTINUE STATEMENT

FORTRAN Syntax
<statement number> CONTINUE

PL/I Syntax
<label part><continue part>

where <label part> is empty, <continue part> is ; and if <statement number> is empty. Otherwise, <label part> is EXTLAB<statement number>: and <continue part> is ;

PAUSE STATEMENT

FORTRAN Syntax
PAUSE<message part>

where <message part> is either empty, or has as its value an unsigned integer constant or a literal constant. Thus, there are three cases to consider:

1. <message part> is empty.

   PL/I Syntax
   DISPLAY('PAUSE 00000') REPLY(NEXTSTA);

2. <message part> is an integer constant.

   PL/I Syntax
   DISPLAY('PAUSE <integer constant>') REPLY(NEXTSTA);

3. <message part> is '<character string>'.

   PL/I Syntax
   DISPLAY('PAUSE <character string>') REPLY(NEXTSTA);

Note: The character variable NEXTSTA is declared with the CHARACTER attribute and a length of (60) and receives a string that is a message to be supplied by the operator.

STOP STATEMENT

FORTRAN Syntax
STOP<integer part>

where <integer part> is either empty or has as its value an integer constant. Thus, there are two cases to consider:

1. <integer part> is empty.

   In a main program:

   PL/I Syntax
   RETURN;

   In a subroutine or function:

   PL/I Syntax
   STOP;

Table 2. Type and Length Specification Conversion

LENGTH	TYPE			
	LOGICAL	INTEGER	REAL	COMPLEX
1	BIT(1)[1]			
2		FIXED BINARY		
4	BIT(1)[1]	FIXED BINARY (31)	FLOAT BINARY	
8			FLOAT BINARY (53)	COMPLEX BINARY
16				COMPLEX BINARY (53)

[1]In PL/I, " the result of a comparison is a bit string of length one." (See the section "Comparison Operations" in the PL/I language specifications manual.)

IMPLICIT STATEMENT

Original
IMPLICIT INTEGER*2(A-H), REAL*8(I-K), LOGICAL (L, M, N)

This page intentionally left blank

2. <integer part> is an integer constant.

In a main program:

PL/I Syntax
```
DISPLAY('N');
RETURN;
```

In a subroutine or function:

PL/I Syntax
```
DISPLAY('N');
STOP;
```

END STATEMENT

FORTRAN Syntax
```
END
```

PL/I Syntax
```
END;
```

SPECIFICATION STATEMENTS

The LCP collects and saves the information contained in each specification statement. Following completion of the source program scan, DECLARE statements will be generated, listing the variables referred to in all statements and giving their types, precisions, dimensions, initial values, etc.

Note that FORTRAN integers represented in the target PL/I program with the precision attribute (15,0) will occupy four bytes in the PL/I version 4-produced object code and two bytes in the later versions.

PREDEFINED SPECIFICATION

All variables, including the predefined FORTRAN variables, are declared by the LCP, as shown in Table 2.

Assume that the only variables in the source program affected by this statement are A, DE, J, and M. In this case, the conversion is done as follows:

Converted
```
DECLARE A FIXED BINARY STATIC,DE FIXED BINARY STATIC,J FLOAT BINARY(53)
STATIC,M BIT(1)STATIC;
```

In the case of a function name, the type will be declared in the RETURNS attribute.

If FUNCT is a function name, the conversion is as follows:

Converted
```
DECLARE FUNCT ENTRY RETURNS(FIXED BINARY);
```

EXPLICIT SPECIFICATION STATEMENT

Original
```
INTEGER*2 ITEM/76/,A(2,2)/2*6,2*1/
```

Converted
```
DECLARE ITEM FIXED BINARY STATIC INITIAL(76),A(2,2)FIXED BINARY
STATIC INITIAL((2)6,(2)1);
```

If the specification statement concerns a function name, the type appears in its associated RETURNS attribute.

Original
```
REAL FUNCT*8
```

Converted
```
DECLARE FUNCT ENTRY RETURNS(FLOAT BINARY(53));
```

DIMENSION STATEMENT

Original
```
DIMENSION A(10), ARRAY(5,6,7)
```

Converted
```
DECLARE A(10)FLOAT BINARY STATIC,ARRAY(7,6,5)FLOAT BINARY STATIC;
```

COMMON STATEMENT

For each common block, the LCP creates a two-level structure of external scope, in which the first item of level 2 is a dummy element which serves to force alignment on a double word.

In the case of unlabeled common blocks, the name of the major structure created is UNLABCM, and the common blocks are processed backwards. Care should therefore be exercized when initializing common block variables using DO loops.

In the case of labeled common blocks, the name of the major structure created will be seven characters in length, and will be made up by concatenating the first n characters of the word COMMON to the original

common label, where n=7-(character length of the original label). The common label ST, for example, is converted into STCOMMO.

Original
```
COMMON A,B,C/R/D,E/ST/F(10)//G,H/SI/I,J/R/P//W
```

Converted
```
DECLARE 1 SICOMMO EXTERNAL,
          2 DUMITEM FLOAT BINARY(53),
          2 F(10) FLOAT BINARY,
          2 I FIXED BINARY(31),
          2 J FIXED BINARY(31);

DECLARE 1 RCOMMON EXTERNAL,
          2 DUMITEM FLOAT BINARY(53),
          2 D FLOAT BINARY,
          2 E FLOAT BINARY,
          2 P FLOAT BINARY;

DECLARE 1 UNLABCM EXTERNAL,
          2 DUMITEM FLOAT BINARY(53),
          2 A FLOAT BINARY,
          2 B FLOAT BINARY,
          2 C FLOAT BINARY,
          2 G FLOAT BINARY,
          2 H FLOAT BINARY,
          2 W FLOAT BINARY;
```

Restrictions

In version 4, because of the implementation of the PL/I (F) compiler, the conversion of FORTRAN integer constants two bytes in length will result in incorrect addressing. The user will therefore have to make the necessary corrections.

Similar inconveniences will appear in the conversion of logical data and may appear for elements with the CHARACTER attribute.

A warning message will be issued.

EQUIVALENCE STATEMENT

In the following example:

```
DIMENSION A(15),C(20)
EQUIVALENCE (A(2),B,C(12)),(D,E,F),(A(15),G)
```

each of (A(2),B,C(12)) and (D,E,F) and (A(15),G) form an equivalence group.

Furthermore, (A(2),B,C(12)) together with (A(15),G) form an "equivalence chain," since B, C(12), and G are made equivalent to an element of the array A. The group (D,E,F) on its own also forms an equivalence chain, since the elements D, E, and F are not made equivalent to any other element.

For each equivalence chain -- the j-th, say -- a one-dimensional array, EQUBLKj, is created. This array is placed within a structure to force alignment on a double word.

Each element of the equivalence chain is placed in a two-level structure defined using EQUBLKj, and made up of:

- A dummy element containing the position within the block
- The element itself, with its attributes

Note: In the interests of maximum efficiency, the method of conversion shown here takes advantage of the facility offered by the PL/I (F) compiler whereby, under certain circumstances, the attributes of the defined item may differ from those of the base identifier (see "Examples of Defining" in Section I of the PL/I reference manual).

Original
```
DIMENSION A(10),B(5,8),C(4,6,9)
EQUIVALENCE (A(3),B(2,3),C(3,4,5))
```

Converted:
```
DECLARE 1 EQUBLK01 STATIC,
          2 DUMITEM  FLOAT BINARY(53),
          2 DUMBASE(216)  FLOAT BINARY;
DECLARE 1 ITEM001 DEFINED EQUBLK01,
          2 DUMITEM(2)  FLOAT BINARY,
          2 C(9,6,4) FLOAT BINARY,
        1 ITEM002 DEFINED EQUBLK01,
          2 DUMITEM(101)  FLOAT BINARY,
          2 B(8,5)  FLOAT BINARY,
        1 ITEM003 DEFINED EQUBLK01,
          2 DUMITEM(110)  FLOAT BINARY,
          2 A(10)  FLOAT BINARY;
```

Restrictions:

- The PL/I (F) compiler does not accept initial values for DEFINED items. Initialization must, therefore, be done by the user.
- See "COMMON Statement."

COMMON VARIABLES APPEARING IN AN EQUIVALENCE STATEMENT

The one-dimensional array EQUBLKj is not created, and each element of the equivalence chain is placed in a two-level structure defined using the common name.

When the equivalence chain extends the size of the common block, a level 2 dummy one-dimensional array is added to the end of the common block.

Original
```
COMMON A,B,C
DIMENSION D(3)
EQUIVALENCE (B,D(1))
```

Converted
```
DECLARE 1 UNLABCM EXTERNAL,
          2 DUMITEM FLOAT BINARY(53),
          2 A FLOAT BINARY,
          2 B FLOAT BINARY,
          2 C FLOAT BINARY,
          2 DUMITEM2(1)  FLOAT BINARY;
DECLARE 1 ITEM001 DEFINED UNLABCM,
          2 DUMITEM(3)  FLOAT BINARY,
          2 D(3) FLOAT BINARY;
```

Restrictions: See "EQUIVALENCE statement."

STATEMENT FUNCTIONS

FORTRAN Syntax
<func>(<arg1>,...,<argn>)=<expression>

PL/I Syntax
```
<func>:PROCEDURE(<arg1>,...,<argn>)conv<type>conv<lengspec>;
       DECLARE<arg1>conv<type>conv<lengspec>,
       ...,<argn>conv<type>conv<lengspec>;
       RETURN(<expression>);
       END;
```

where conv<type> and conv<lengspec> are empty if <func> does not appear in a specification statement.

FUNCTION SUBPROGRAMS

FORTRAN Syntax
```
<type>FUNCTION<func><lengspec>(<arg1>,...,<argn>)
      ...
      <func>=<expression>
      ...
      RETURN
      ...
      END
```

where <type> and <lengspec> are optional.

PL/I Syntax
```
(NOZERODIVIDE):<func>:PROCEDURE(<arg1>,..,<argn>,<func><name tail>);
      ...
      <func><name tail>=<expression>;
      ...
      RETURN;
      ...
      END;
```

Note that no attempt is made by the LCP to simulate a FORTRAN IV call by value; that is, a call by value of a FUNCTION or SUBROUTINE subprogram formal parameter is treated as a call of that parameter by name.

Note 1: <func><name tail> is an additional parameter, created by the LCP, that simulates the effect of a FORTRAN function call, thus taking advantage of the facilities of the PL/I (F) compiler. The value given to <name tail> is as many characters of the string VARFUN as are required to make the parameter name seven characters long. This parameter is declared according to the predefined specification or according to the <type> or <lengspec> specified (see Table 3).

If <func> itself is replaced by a substitution name, the procedure and the additional parameter have the same name (see "Prevention of Name Conflicts").

Note 2: The user should keep in mind that FORTRAN IV and PL/I do not handle adjustable dimensions in the same manner. In PL/I, the dimensions of an array passed as argument are those of the calling program.

Note 3: The number of dimensions of an array used in the list of parameters must be equal to that of the corresponding argument.

SUBROUTINE SUBPROGRAMS

Original
```
SUBROUTINE SUB (X,*,/Y/,Z,*,R,/S/,*)
```

Converted
```
(NOZERODIVIDE):SUB:PROCEDURE(X,RETURN01,Y,Z,RETURN02,R,S,RETURN03);
```

Note: For conversion of the asterisks, see "RETURN Statement."

Original
```
SUBROUTINE SETUP
```

Converted
```
(NOZERODIVIDE):SETUP:PROCEDURE;
```

Adjustable Dimensions: See Note 2 under "Function Subprograms" above.

Array as Parameter: See Note 3 under "Function Subprograms"above.

ENTRY STATEMENT

Since initialization of parameters at primary and secondary entry points of a PL/I procedure is not generally performed in the same manner as in FORTRAN, conversion of each ENTRY statement will take place, but a warning message will be issued.

FORTRAN Syntax
```
ENTRY <entry name><arglist>
```

PL/I Syntax
```
<entry name>:ENTRY conv<arglist>;
```

where:

conv<arglist> takes exactly the same form as in the SUBROUTINE statement if the ENTRY statement appears in the body of a SUBROUTINE subprogram, or the same form as in the FUNCTION statement if the ENTRY statement appears in the body of a FUNCTION subprogram.

RETURN STATEMENT IN A SUBROUTINE SUBPROGRAM

FORTRAN Syntax
```
RETURN<index>
```

where <index> is empty or has as its value an integer constant or an integer variable. If <index> is empty, then the converted statement is RETURN;

If <index> has an integer value, or an integer variable value, then a DECLARE statement is created which takes the form:

```
DECLARE RETARAY(p)LABEL,RETURN01 LABEL,...,RETURNp LABEL;
```

where p is the maximum number of * characters appearing in the SUBROUTINE or ENTRY statement parameter lists.

In addition, the RETURN <index> statement is converted to:

```
GC TO RETARAY(<index>);
```

Initialization of RETARAY is performed at the primary entry point as follows:

```
      GO TO EXTLABS;
      RETARAY(1):GO TO RETURN01;
      ...
      RETARAY(p):GO TO RETURNp;
EXTLABS:;
```

The conversion shown takes advantage of the alternative method available for the initialization of elements of non-static variable arrays (see the PL/I language specifications manual, chapter 4, section entitled "Initial Attributes").

CALL STATEMENT

FORTRAN Syntax
CALL<sbrtn><arglist>

PL/I Syntax
CALL<sbrtn> conv<arglist>;

where conv<arglist> is empty if <arglist> was empty, or takes the form <arglist> if no statement number appears in <arglist>.

Otherwise, if &<statement number> appears, it is converted to:

```
EXTLAB<statement number>
```

Note that FORTRAN literal constants should not be passed as actual parameters to subprograms. In the event of this happening, a warning message is issued.

The name of the PL/I equivalent of certain FORTRAN mathematical function subprograms cannot be passed as an argument. The PL/I built-in function names which cannot be used as arguments are: FIXED, ABS, MOD, REAL, MAX, MIN, FLOAT, IMAG, TRUNC, COMPLEX, and CONJG. For other PL/I built-in function names passed as arguments, the user must specify the ENTRY attribute in order to describe their entry points properly.

Array as Parameter: See Note 3 under "Function Subprograms"above.

EXTERNAL STATEMENT

FORTRAN Syntax
EXTERNAL<sbpgm1>,...,<sbpgmp>

PL/I Syntax
DECLARE<sbpgm1>ENTRY RETURNS(<type><length>),...,<sbpgmp>ENTRY RETURNS (<type><length>);

BLOCK DATA SUBPROGRAM

FORTRAN Syntax
```
BLOCK DATA
...
END
```

PL/I Syntax
```
BEGIN;
...
END;
```

All BLOCK DATA subprograms must immediately precede the main program to which they belong.

DATA INITIALIZATION STATEMENT

Initial values appearing in a DATA statement are placed in the INITIAL attribute of the corresponding variables.

When the initial values are literals, the corresponding variables are declared as CHARACTER and must remain so throughout the program.

Note: Since the PL/I (F) compiler does not accept initial values for DEFINED items, the initialization of variables in EQUIVALENCE statements must be done by the user.

DOUBLE PRECISION STATEMENT

FORTRAN Syntax
```
DOUBLE PRECISION <var1><dim1>,...,<varp><dimp>
```

where <vari> has as its value a variable, array, or function name, and where each <dimi> is either empty, or has as its value a subscript list in parentheses.

PL/I Syntax
```
DECLARE<var1><dim1>FLOAT BINARY(53)STATIC,...,<varp><dimp>FLOAT
BINARY(53)STATIC;
```

If <varj> is a function, the specification is given by RETURNS (<type>).

SERVICE SUBPROGRAMS

If a call to any of the following subprograms:

SLITE, SLITET, OVERFL, DVCHK, DUMP, PDUMP

occurs in the source program, it remains unchanged, but the LCP provides no corresponding external procedure

A call to the FORTRAN EXIT subprogram is converted directly to STOP, unless the subprogram name EXIT is followed by <arglist>. In this case, the converted statement takes the following form:

```
CALL EXIT(conv<arglist>);
```

A warning message is issued.

INPUT/OUTPUT STATEMENTS

Sequential input/output statements only are converted.

READ STATEMENTS

The LCP processes each of the three basic forms of the READ statement. The following syntactic variables are used in the discussion of READ statement conversion:

- <data set ref no>, which has as its value an unsigned integer constant or variable representing a data set reference number
- <formlist name>, which has as its value the statement number or array name of the FORMAT statement describing the data to be read, or a NAMELIST name
- <end err part>, which takes the form <end part><err part>, or <err part><end part>

  where:

  <end part> is empty or takes the form:

  , END = <end statement no>

  <err part> is empty or takes the form:

  , ERR = <err statement no>
- <list part>, which is empty, or is a list of variable or array names that may be indexed and incremented

Note: <data set ref no> is converted into FT<data set ref no>F01

For each of the three basic forms of the READ statement, the LCP processes the <end err part>, when present, as follows:

If END=<end statement no> is present, the LCP, just before converting the READ statement, generates the ON-condition statement:

```
ON ENDFILE(FT<data set ref no>F01)GO TO EXTLAB<end statement no>;
```

If ERR=<err statement no> is present, the LCP, just before converting the READ statement, generates the ON-condition statement:

```
ON TRANSMIT(FT<data set ref no>F01)GO TO EXTLAB<err statement no>;
```

In addition, if <end err part> is present, the LCP, immediately after the converted READ statement, generates the ON-condition statement:

```
ON<end err cond>(FT<data set ref no>F01)SYSTEM;
```

where <end err cond> is either ENDFILE or TRANSMIT.

If <formlist name> has the format <namelist name> associated with a NAMELIST statement, the LCP must process a statement of the form:

```
NAMELIST/<name 1>/<vararray list 1>/.../<name n>/<vararray list n>
```

where <namelist name> is a <name i>, and <vararray list i> is a list of variable or array names associated with <name i>.

For each <namelist name>, the LCP creates a table of the associated variable and array names to be referred to via the <namelist name> in a subsequently generated GET statement.

Note: If <formlist name> is an array name referring to a FORMAT statement, or if <data set ref no> is an integer variable, conversion does not take place and a warning message is issued.

Form READ (<data set ref no><namelist name><end err part>)

PL/I Syntax
GET FILE(FT<data set ref no>F01)DATA(<vararray list i>);

A warning message is issued after conversion.

Note: In PL/I, no search is made for a specific name list. It is therefore the user's responsibility to ensure that the data is arranged in the correct sequence.

Form READ (<data set ref no><format name><end err part>)<list part>

PL/I Syntax
GET FILE(FT<data set ref no>F01)EDIT(conv<list part>)(R(conv<format name>));

Source program variables within I/O lists may be indexed and incremented in the same manner as variables in a DO statement; the LCP treats them identically.

If the <list part> of the READ statement contains indexed I/O lists and/or arrays a warning message is issued. If the FORMAT statement referred to contains literal data or an A- or H-format code, conversion of the literals or of the A- or H-format codes affecting the elements of the list included in and following the first array or indexed I/O list is incorrect.

Original
```
    DIMENSION A(10),B(10)
    READ(5,100)(A(I),B(I),I=1,10)
100 FORMAT ('A=',E12.5,'B=',E12.5)
```

Converted (with warning)
```
   DECLARE A(10)FLOAT BINARY STATIC,B(10)FLOAT BINARY STATIC;
   GET FILE(FT05F01) EDIT((A(I),B(I) DO I=1 TO 10))
   (R(EXTLAB100));
   EXTLAB100:FORMAT(COLUMN(1),A(2),E(12,5),A(2),E(12,5));
```

Form READ (<data set ref no>)<list>

Since binary data cannot be directly transmitted in PL/I in the same manner as in FORTRAN, statements of this type are not converted; they are, however, identified by messages.

Form READ <format name>,<list part>

PL/I Syntax
GET FILE (SYSIN) EDIT (conv<list part>)(R(conv<format name>));

Original
```
READ 5, I
5 FORMAT (I5)
```

Converted
```
GET FILE (SYSIN) EDIT (I)(R(EXTLAB5));
EXTLAB5:FORMAT(COLUMN(1),F(5));
```

PRINT STATEMENT

Original
```
PRINT 5,I
5 FORMAT(I5)
```

Converted
```
PUT FILE(SYSPRINT)EDIT(I)(R(EXTLAB5));
EXTLAB5:FORMAT(COLUMN(1),F(5));
```

PUNCH STATEMENT

Original
```
PUNCH 5,I
5 FORMAT(I5)
```

Converted
```
PUT FILE(SYSPRINT)EDIT(I)(R(EXTLAB5));
EXTLAB5:FORMAT(COLUMN(1),F(5));
```

WRITE STATEMENT

Except for the absence of the parameters END and ERR, the LCP treats the WRITE statement in the same way as the READ statement, replacing GET by PUT.

Note: Differences in format will appear in the output from the converted program in the case of namelist transmission.

FORMAT STATEMENT

To force a new record each time that a FORMAT statement is used, the LCP inserts the control format item COLUMN(1) at the beginning of the converted format list.

Note that in PL/I, format items, even when they include control format items, are ignored if they appear after transmission of the last data list item. Differences in format may therefore appear in the output listing.

The following examples illustrate the conversion of the various forms of the FORTRAN FORMAT statement.

Numeric Format Items

I, F, E, D codes

Original
```
5 FORMAT (3I2, 4F11.4, 2E9.3, 3D20.  16)
```

Converted
```
EXTLAB5:FORMAT(COLUMN(1),3 F(2),4 F(11,4),2 E(9,3),3 E(20,16));
```

The BLKZR control card option enables the user to call the function LBLNK. LBLNK will then appear in the converted format code. On input, this function modifies numeric data as follows:

- replaces &(plus sign in BCDIC) by +
- replaces D (double-precision) by E
- inserts a zero if the external data field is blank
- inserts + after E, if the character following E is a blank, and replaces other embedded and trailing blanks by zeros.

If the FORMAT statement is only used for PRINT files, LBLNK is not generated.

Transmission of Complex Data: FORTRAN requires an E-format specification for each part of a complex number. This format can be used in the same program for both complex and real numbers. It is user's responsibility to replace E-format specifications by PL/I C-format specifications.

Original
```
5    FORMAT(F5.3,E10.3)
```

Converted (with option BLKZR)
```
EXTLAB5:FORMAT(COLUMN(1),F(LBLNK(5),3),E(LBLNK(10),3));
```

Scale Factor

<integer constant>P<real item>

where <integer constant> may be positive or negative, and <real item> takes the form Fw.d.

Original
```
8 FORMAT (-1PF11.4,F11.4)
```

Converted
```
EXTLAB8:FORMAT(COLUMN(1),F(11,4,-1),F(11,4,-1));
```

Since PL/I accepts the P-factor for F-format items only, the LCP does not convert this factor for E or D codes, and a warning message is issued.

Note: The effect of the scale factor is dynamic in FORTRAN IV. The results of the PL/I target program may therefore differ from the expected results. If the format refers to an input statement, it is the user's responsibility to invert the sign of the P-factor.

Logical Format Item

<format code> is L.

Original
```
7 FORMAT (2L10)
```

Converted
```
EXTLAB7:FORMAT(COLUMN(1),2 B(10));
```

The user should note that logical data in the external medium must be in a form accpetable to the PL/I compiler. Thus, the values .TRUE. and .FALSE. must be represented by 1 and 0 respectively, and may occur anywhere within the field of the size indicated in the FORMAT statement. On output, the truth values 1 and 0 will be left-adjusted within the indicated field. Thus, using the above example, external data for input might appear in the form:

<blank>(2)1<blank>(7)<blank>0<blank>(8)

where <blank>(p) is a string of p consecutive <blank>'s.

Similarly, output of logical data would take the form:

1<blank>(9)0<blank>(9)

A warning message is issued.

Character-String Format Item

<format code> is A.

Original
```
5 FORMAT (20A4)
```

Converted
```
EXTLAB5:FORMAT(COLUMN(1),20A(4));
```

Note 1: Using the A-format code, FORTRAN can read or write a character-string in a field having a variable name. This variable will be given the CHARACTER (4) attribute in the PL/I target program, except if it appears after an indexed list and/or an array. It, therefore, retains the CHARACTER (4) attribute throughout the program, i.e., it must not be used to contain numeric values. In particular, the user must check that the variable passed through the FORTRAN COMMON and CALL statements in both main program and subprograms has the same type of declaration in PL/I. Note also that, FORTRAN IV source programs written for current IBM systems other than System/360 may give incorrect results due to the number of characters that are transmitted.

Note 2: If the <list part> of the corresponding input/output statement contains indexed lists and/or arrays, the variables of the <list part> that are included in or follow the first indexed list or array, and that correspond to an A-format code, are not declared as CHARACTER.

Generalized Format Item

<format code> is G.

This format code is not converted; a message is issued.

Hexadecimal Format Item

<format code> is Z.

This format code is not converted; a message is issued.

Literal Data and H-Format Code

Original
```
98 FORMAT (' HEADING')
```

Converted
```
EXTLAB98:FORMAT(COLUMN(1),A(8));
```

Original
```
98 FORMAT (8H HEADING)
```

Converted
```
EXTLAB98:FORMAT(COLUMN(1),A(8));
```

In either case, a dummy variable containing the string ' HEADING' is created and transferred to the data list in the corresponding GET or PUT statement, and the remote format item R(EXTLAB98) is appended to the relevant statement.

For indexed I/O lists, the conversion of literals or of H-format code is incorrect (see READ statement).

Control Format Items

- Spacing Format Item:

  Original
  ```
  5 FORMAT (I10, 10X, 4I10)
  ```

  Converted
  ```
  EXTLAB5:FORMAT(COLUMN(1),F(10),X(10),4 F(10));
  ```

- Printing Format Item: <blank>,0,1,+,/.

  Printer control characters are printed as output data characters and are included in the format code.

  Original
  ```
  5 FORMAT ('1NEXT PAGE', T15,'DATA'/' NEXT LINE'///// ' SKIP',
  'FOUR LINES'/ '0DOUBLE SPACING')
  ```

  Converted
  ```
  EXTLAB5:FORMAT(PAGE,A(10),COLUMN(15),A(4),SKIP(1),COLUMN(1),A(10),
      SKIP(5),COLUMN(1),A(5),A(10),SKIP(3),A(15));
      A(10),SKIP(2),A(15));
  ```

In addition, the literals apppearing in the source FORMAT statement are placed in dummy variables which are transferred, in order, to the data list of the generated PUT (or GET) statement whose remote format reference is R(EXTLAB5).

The format code + is converted into the format item SKIP(0).

- Parentheses: When there are more data list items than format items, and in order to force the repetition of the format from the last-included left parenthesis, an additional pair of parentheses is required. Therefore, every time it encounters a pair of parentheses of level 2 in a FORMAT statement, the LCP automatically creates an additional pair of parentheses preceded by a repetition factor of 32767.

  Original
  ```
  5 FORMAT (I2,(F5.2,I4),I5)
  ```

  Converted
  ```
  EXTLAB5:FORMAT(COLUMN(1),F(2),32767(1(F(5,2),F(4)),F(5),SKIP(1)));
  ```

When the same FORMAT statement applies to PRINT and non-PRINT files, a warning message is issued.

Note: The first character of a string appearing at the beginning of a record is both converted as a control character and treated as data if one of the corresponding files is a PRINT file. When such a string contains only one character, it is converted as a control character if all the corresponding files are PRINT files.

END FILE STATEMENT

This statement is not converted; a warning message is issued in the output listing. However, if on the same data set an END FILE statement is dynamically followed by a REWIND statement, the effect may be the same as in FORTRAN.

REWIND STATEMENT

Original
```
REWIND 6
```

Converted
```
CLOSE FILE(FT06F01);
```

Note: The conversion of a REWIND statement by a CLOSE statement may give different effects, for the data set will be repositioned at the beginning of the tape after writing an end-of-file mark. A warning message is issued.

BACKSPACE STATEMENT

This statement is not converted; a message is issued.

The LCP can generate two forms of conversion output:

1. A listing of the converted program
2. The converted program on punched cards or in card-image form

The listing of the converted program is always provided, but the punched-card (or card image) output is optional. The listing contains the converted program together with messages generated during conversion. Additionally, the user can specify that a listing of the source program be included.

LISTING

The listing contains two major sections:

1. The optional source program listing
2. The converted program

The source program listing contains the original source statements exactly as they appeared in the input.

The listing of the converted program includes :

- The converted program itself. This again is divided into two parts. One part contains converted statements, statements generated by the LCP, messages replacing statements that have not been converted (i.e., FORTRAN statements for which conversion is not possible or not practical), and warning message flags. The other contains messages showing either that the FORTRAN source statement cannot be converted or that conversion has taken place, but that fidelity to the source statement cannot be guaranteed.
- A table of source program statement function or subprogram names, each of which has been replaced either by an equivalent built-in function or procedure (see Appendix B), or by a function name that avoids conflict with PL/I built-in function names not available in FORTRAN. The changed or substituted name appears next to the function name it has replaced.
- A table of source program variable or array names that have been replaced by an LCP substitution name. The replacement appears next to the corresponding variable or array name.

MESSAGES

Messages in the output listing indicate clearly the statements in the converted program to which they apply, thus enabling the user to scan the program for statements that require manual changes.

A message appears with each output statement that falls into one of the following categories:

- The FORTRAN statement is not convertible into PL/I.
- The PL/I statement may not have the same effect as the corresponding FORTRAN source statement.

Using the output listing, the present manual, and the PL/I language specifications manual , the user can determine the hand changes required to make the PL/I program suitable for compilation.

OUTPUT

The punched cards (or card images) produced by the LCP contain converted statements, the form of which matches that in the listing.

Statements flagged with a warning message should, where necessary, be corrected before the PL/I program is submitted for compilation.

APPENDIX A. CORRESPONDING FORTRAN AND PL/I BASIC SYMBOLS

FORTRAN IV Symbol	PL/I 60-Character Set Symbol	PL/I 48-Character Set Symbol
A-Z	A-Z	A-Z
$	$	$
0-9	0-9	0-9
blank	blank	blank
= or •EQ•	=	=
+	+	+
-	-	-
*	*	*
/	/	/
(	(	(
)	)	)
,	,	,
.	.	.
' (apostrophe)	' (apostrophe)	' (apostrophe)
•NOT•	¬	NOT
•AND•	&	AND
•OR•	|	OR
•GT•	>	GT
•LT•	<	LT
•GE•	>=	GE
•NE•	¬=	NE
•LE•	<=	LE

APPENDIX B. CONVERSION OF FORTRAN MATHEMATICAL FUNCTION SUBPROGRAMS

In the following table, unless otherwise specified, the number of arguments associated with each FORTRAN function is the same as that for the corresponding PL/I function.

FORTRAN Function	PL/I Function	
EXP	EXP	
DEXP	EXP	
CEXP	EXP	
CDEXP	EXP	
ALOG	LOG	
DLOG	LOG	
CLOG	LOG	
CDLOG	LOG	
ALOG10	LOG10	
DLOG10	LOG10	
ATAN	ATAN	
DATAN	ATAN	
ATAN2	ATAN	
DATAN2	ATAN	
SIN	SIN	
DSIN	SIN	
CSIN	SIN	
CDSIN	SIN	
COS	COS	
DCOS	COS	
CCOS	COS	
CDCOS	COS	
SQRT	SQRT	
DSQRT	SQRT	
CSQRT	SQRT	
CDSQRT	SQRT	
TANH	TANH	
DTANH	TANH	
MOD	MOD	see Note 2
AMOD	MOD	
DMOD	MOD	
IABS	ABS	
ABS	ABS	
DABS	ABS	
CABS	ABS	
CDABS	ABS	
INT	TRUNC	
AINT	TRUNC	
IDINT	TRUNC	

FORTRAN Function	PL/1 Function
AMAX0	MAX
AMAX1	MAX
MAX0	MAX
MAX1	MAX
DMAX1	MAX
AMIN0	MIN
AMIN1	MIN
MIN0	MIN
MIN1	MIN
DMIN1	MIN
FLOAT	FLOAT
DFLOAT	FLOAT
IFIX	FIXED
HFIX	FIXED
SIGN	See Note 3
ISIGN	"
DSIGN	"
DIM(<arg1>,<arg2>)	See Note 2
IDIM(<arg1>,<arg2>)	"
SNGL(<arg>)	"
REAL	REAL
AIMAG	IMAG
DBLE(<arg>)	See Note 2
CMPLX	COMPLEX
DCMPLX	COMPLEX
CONJG	CONJG
DCONJG	CONJG
TAN	TAN
DTAN	TAN
SINH	SINH
DSINH	SINH
COSH	COSH
DCOSH	COSH
ERF	ERF
DERF	ERF
ERFC	ERFC
DERFC	ERFC

If the source program contains subprogram names created by user that match FORTRAN IV mathematical function names, they must be specified in an LCP control card (see Appendix E) so as not to be converted to PL/I functions. Thus, if the assignment statement:

```
Y=EXP(X)+DEXP(Z)
```

appears in the source program, and DEXP is a user function name, this name must be listed in the LCP control card. The user can then provide his own DEXP FORTRAN function subprogram for conversion to PL/I.

Note 1

If a name created by the user (EXP, for example) coincides with one of the PL/I built-in function names listed above (except for COMPLEX and REAL), a conflict may arise if the name is also the PL/I equivalent of a FORTRAN function (DEXP, for example) used elsewhere in the program. In this case, the converted form of the assignment statement illustrated in the preceding paragraph would be:

```
Y = EXP(X) + EXP(Z)
```

Note 2

The following mathematical functions:

DIM, IDIM, SNGL, DBLE, ARSIN, DARSIN, ARCOS, DARCOS, COTAN, DCOTAN, GAMMA, DGAMMA, ALGAMMA, DLGAMMA

are not converted.

The conversion provided for the mathematical functions MOD, AMOD, and DMOD produces correct results only if the first argument is greater than zero.

Note 3

The LCP converts the functions SIGN, ISIGN, and DSIGN by providing an internal procedure for each.

For the SIGN function the procedure is:

```
SIGN:PROCEDURE(A1,A2) FLOAT BINARY;
     DECLARE(A1,A2) FLOAT BINARY;
     IF(A2<0) THEN RETURN(-ABS(A1));
     ELSE RETURN(ABS(A1));
     END;
```

In the ISIGN function, FLOAT BINARY is replaced wherever it appears by FIXED BINARY(31); in the DSIGN function, by FLOAT BINARY(53).

1. Hexadecimal and octal constants are not converted.

2. The conversion of subscripts containing the operators /or **, mixed mode expressions, function references, or subscripted names may give incorrect results.

3. The user must ensure that his program, in no event, uses an assigned variable for any purpose other than for the assigned GO TO statement.

4. The user must ensure that his program, in no event, contains a transfer back into a DO loop.

5. When converting a COMMON statement, the user should ensure that common blocks in the various subprograms are the same, that is, the COMMON statements in the various subprogram must be identical.

6. In version 4, the conversion of FORTRAN integer constants two bytes in length, and of logical data items may give incorrect results.

7. The number of dimensions of an array used in the list of parameters must be equal to that of the corresponding argument.

8. On entry to a function or to a subroutine, initialization of parameters made on a previous entry may be lost.

9. In a DATA statement, when the initial values are literals, the corresponding variables are declared as CHARACTER by the LCP, and must remain so throughout the program, i.e., they must not be used to contain numeric values. In particular, the user must check that variables passed through the FORTRAN COMMON and CALL statements in both main program and subprograms have the same type of declaration in PL/I. An implied DO in a DATA statement is not converted.

10. FORTRAN literal constants should not be passed as arguments to a subprogram. The user should note that FORTRAN IV and PL/I do not handle adjustable dimensions in the same manner, and that in PL/I the dimensions of an array passed as argument are those of the calling program.

11. Initial values assigned to variables in EQUIVALENCE statements should be adjusted by the user.

12. The LCP does not provide PL/I external procedures to simulate the effect of FORTRAN service subprograms OVERFL, DVCHK, SLITE, SLITET, DUMP, and PDUMP.

13. The LCP does not provide external procedures to simulate the effect of the following FORTRAN mathematical function subprograms:

    ARSIN, DARSIN, ARCOS, DARCOS, COTAN, DCOTAN, GAMMA, DGAMMA, ALGAMA, DLGAMA, DIM, IDIM, SNGL, DBLE.

14. A READ/WRITE statement is not converted if:

    a. The statement applies to direct-access mode.

    b. A FORMAT statement is referred to by an array name.

c. Binary data transmission is indicated (no format reference).

d. A data set reference number is an integer variable.

15. The following restrictions, due to differences in the implementation of PL/I, apply to the conversion of the FORMAT statement:

a. When an input/output statement contains arrays or indexed I/O lists, and the FORMAT statement referred to contains literal data, or an A- or H-format code, conversion of the literals or of the A- or H-format codes affecting the elements of the list included in and following the first array/indexed I/O list is incorrect.

b. In PL/I, format items, even when they include control format items, are ignored if they appear after transmission of the last data list item. Consequently, differences in format may appear in the output listing.

c. E-format codes associated with complex numbers must be changed into PL/I C-format codes.

d. P-scale factors associated with E- or D-format codes are not converted. Moreover, the effect of the scale factor associated with an F-format code may differ. If the format refers to an input statement, it is the user's responsibility to invert the sign of the P-scale factor.

e. G-, Z-, and O-format codes are not converted.

f. When a format refers to a PRINT file and/or a non-PRINT file, the first character of a string appearing at the beginning of a record is treated, in general, both as a control character and as data.

16. The BACKSPACE and END FILE statements are not converted.

17. The conversion of the REWIND statement by a CLOSE statement gives equivalent effects only if the REWIND statement applies to the first data set on the tape.

18. If a name created by the user coincides with a PL/I built-in function name, there may be conflict if the name is also to be the PL/I equivalent of a FORTRAN mathematical function used elsewhere in the program.

19. The name of the PL/I equivalent of certain FORTRAN mathematical function subprograms cannot be passed as an argument. These PL/I built-in function names which cannot be used as arguments are: FIXED, ABS, MOD, REAL, MAX, MIN, FLOAT, IMAG, TRUNC, COMPLEX, and CONJG. For other PL/I built-in function names passed as arguments, the user must specify the ENTRY attribute in order to describe their entry points properly.

The LCP is distributed by IBM in one of two forms:

- On a disk pack for users having no tape units
- On tape

PROGRAMS ON DISK PACK

CONTENTS OF THE DISK PACK

The following data sets are written on the disk pack with label FLCPRS:

1. A sequential data set (DSNAME=FOCCARD) containing the control cards required by the linkage editor to create a partitioned data set with one member: the LCP.

2. Eighteen data sets containing the 18 modules of the LCP. These modules are in object form, being the output from the PL/I (F) compiler version 5, or from the System/360 Operating System (F) assembler.

3. A sequential data set (DSNAME=SAMPLF) containing the sample program written in FORTRAN. (For use of the sample program, refer to Appendix H.)

4. One data set (DSNAME=LBLNK) containing the object module of the function, provided to process blank as zero in numeric data under the control of the option BLKZR.

CREATING THE LOAD MODULE

The user must perform the following steps:

1. Transfer the data set mentioned in (1) above to cards, to obtain the control cards required by the linkage editor. The following is an example of the control cards needed to do this:

```
//AB         JOB     4727,SMITH,MSGLEVEL=1
//ABA        EXEC    PGM=IEBPTPCH
//SYSPRINT   DD      SYSOUT=A
//SYSUT2     DD      SYSOUT=B
//SYSUT1     DD      UNIT=2311,                          C
//                   DISP=OLD,                           C
//                   DSNAME=FOCCARD,                     C
//                   VOLUME=SER=FLCPRS
//SYSIN      DD      *
  PUNCH
/*
```

The control cards obtained contain all the information required by the linkage editor, i.e., names to be used for the load module to be created, overlay structure, etc.

2. Modify the JOB card and the volume serial number in the //SYSLMOD DD... card in order to use his own label. In addition, the user may have to modify other cards, depending on:

   - The level of his linkage editor
     (The editor used here is the 44K E-level linkage editor.)
   - Any change that may be required in the names of the load module

3. Add a /* card to the control cards obtained, and create a partitioned data set (DSNAME=FORLCP) containing the LCP (member name= LCPFORT), using as input to the linkage editor the 18 data sets described under "Contents of the Disk Pack."

Note: The same procedure will be used when maintaining the LCP; in this case, the input to the linkage editor will consist of the updated modules, also delivered in object form.

PROGRAMS ON TAPE

CONTENTS OF THE TAPE

Four files are written on the tape, which is blocked with a blocksize of 2400 bytes:

File 1, which contains the 18 modules of the LCP and the overlay structure

File 2, which contains the control cards required by the linkage editor

File 3, which contains the sample program written in FORTRAN

File 4, which contains the object module LBLNK

CREATING THE LOAD MODULE

The user must perform the following steps:

1. Transfer file 2 to cards to obtain the control cards required by the linkage editor. The following is an example of the cards needed to do this:

```
//ABC        JOB     4727,SMITH,MSGLEVEL=1
//ABA        EXEC    PGM=IEBPTPCH
//SYSPRINT   DD      SYSOUT=A
//SYSUT1     DD      UNIT=2400,LABEL=(2,NL),                          C
//                   VOLUME=SER=888888,DISP=OLD,                      C
//                   DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSUT2     DD      SYSOUT=B
//SYSIN      DD      *
  PUNCH
/*
```

   The control cards obtained contain all the information required by the linkage editor, i.e., names to be used for the load module to be created, etc.

2. Deblock file 1 using the IEBGENER utility program.

3. Modify the control cards, if required. (See item 2 of the section "Creating the Load Modules" under "Programs on Disk Pack.")

4. Add a /* card to the control cards obtained, and create a partitioned data set (DSNAME=FORLCP) containing the LCP (member name= LCPFORT), using the 18 modules in file 1 as input to the linkage editor.

Note: The same procedure will be used when maintaining the LCP; in this case, the input to the linkage editor will consist of the updated modules, also delivered in object form.

USING THE FUNCTION LBLNK

When executing the PL/I program and if the option BLKSZ has been used during the conversion, file 4 must be copied onto a disk pack in order to be used. The following is an example of the cards needed to do this.

```
//AB        JOB     4727,SMITH,MSGLEVEL=1
//A         EXEC    PGM=IEBGENER
//SYSIN     DD      DUMMY
//SYSPRINT  DD      SYSOUT=A
//SYSUT2    DD      DISP=(NEW,KEEP),UNIT=2311,                    C
//                  VOLUME=SER=XXXXXX,DSNAME=LBLNK,               C
//                  DCB=(RECFM=F,LRECL=80,BLKSIZE=80),            C
//                  SPACE=(TRK,(1,1))                             C
//SYSUT1    DD      UNIT=2400,LABEL=(4,NL),                       C
//                  VOLUME=SER=888888,DISP=OLD,                   C
//                  DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
/*
```

EXECUTING THE LCP

The following control cards must be supplied by the user to execute the LCP. The standard options are underlined.

```
| //AB          JOB  4727,SMITH,MSGLEVEL=1
  //JOBLIB      DD   DSNAME=FORLCP,DISP=OLD,UNIT=2311,VOLUME=SER=xxxxxx
  //            EXEC PGM=LCPROPT,PARM='SOURCE|NOSOURCE,DECK|NODECK,        C
  //                 BCD|EBCDIC,EXTREF|NOEXTREF,CHAR48|CHAR60,             C
  //                 BLKZR|NOBLKZR,SIZE=xxxxxx'
  //SYSPCH      DD   SYSOUT=B
| //SYSIOERR    DD   SYSOUT=A
  //SYSPRNT     DD   SYSOUT=A,DCB=(RECFM=VA,BLKSIZE=129,LRECL=125)
  //SYSUT1      DD   Parameters defining a work data set. See also
                   the note on the parameters SPACE and DCB.
  //SYSUT2      DD   As for SYSUT1
  //SYSIN       DD   Parameters defining source input data set
  /       (Starting in column 7) List of subprogram names created by the
          user that match FORTRAN IV mathematical function names
          and of data-set-numbers to be declared PRINT. Names
          and numbers may be mixed, but must be separated by commas.
          r
          |
          |          FORTRAN SOURCE PROGRAM
          |
          L
  /*
```

Note:

SYSUT1

SPACE	This parameter depends on the length of the FORTRAN source program to be converted One cylinder is required for 120 cards of source program or subprogram.
DCB	This parameter has the following form: DCB=(DSORG=DA,KEYLEN=9)

SYSUT2

SPACE	One cylinder is required for 500 cards of source program
DCB	This parameter has the following form: DCB=(DSORG=DA)

Batch Processing: In batch processing, the spaces used on the disk are the same. Therefore, to estimate the number of cylinders required, the total number of cards in the batch should not be taken into consideration, but only the number of cards in the largest program in the batch.

CONTROL CARD OPTIONS

EXEC CARD OPTIONS

The following options can be specified in the PARM field of the EXEC card. If no option is specified, the standard option (underlined) is assumed.

The total length of the options indicated between apostrophes in the PARM field must not exceed 40 characters, commas included. Because of this limitation, the LCP accepts abbreviated options (indicated between parentheses in the following text):

- NOSOURCE or SOURCE (NS or S)
  This option specifies whether the FORTRAN source program is to be listed on the device indicated by the SYSPRNT DD card.

- DECK or NODECK (D or ND)
  This option specifies whether the PL/I program is to be punched on the device indicated by the SYSPCH DD card.

- BCD or EBCDIC (B or EB)
  This option specifies the character code of the FORTRAN source program and, consequently, that of the LCP output.

- CHAR48 or CHAR60 (C48 or C60)
  This option specifies which character set is to be used to list the converted program.

- EXTREF or NOEXTREF (E or NE)
  This option specifies whether the name changes in the FORTRAN source program are to be listed on the device indicated by the SYSPRNT DD card.

- BLKZR or NOBLKZR (BZ or NBZ)
  This option specifies whether the external form of numeric input data must be processed during execution of the PL/I program; if this is the case, the function LBLNK is used for the conversion of E-, F- and I-format items.

- SIZE=xxxxx or SIZE=71680
  This option specifies the main storage size that is available to the LCP. The minimum size of main storage is 71680 bytes; this is the standard size for the purposes of this option. If the user:

  1. Specifies a smaller value, it is ignored and the standard size is assumed.

  2. Specifies a value greater than 71680. This will result in an improvement in performance.

LCP CONTROL CARDS

These cards, if necessary, are placed after a SYSIN DD statement and between programs in batch processing. These cards contain:

- / in column 1
- Starting in column 7:

  1. The subprogram names created by the user that match FORTRAN IV mathematical function names (from the list given in appendix B); these names are not changed by conversion.

  2. The data set numbers that he wishes declared with the PL/I attribute PRINT. Any numeric field of up to two digits is considered as a data set reference number. Note that the PL/I file FT06F01 (data set number=6) is automatically declared with the PRINT attribute. If the user wishes to override this declaration, he must specify 0 as the data set reference number.

EXECUTING THE PL/I TARGET PROGRAM

After any necessary hand changes are made, the converted PL/I target program may be used with a normal set of control cards for PL/I programs, for example, IBM supplied catalogged procedures such as PL1LFCLG.

If the user chooses the option BLKZR during the conversion run, he must link edit the LBLNK module when executing his target program. He will have to add the following control card for the link edit step:

```
//LKED.SYSIN DD DSNAME=LBLNK,DISP=OLD,UNIT=SYSDA,VOLUME=SER=xxxxxx
```

where xxxxxx is the volume serial number of the disk pack containing the LBLNK object module.

This appendix contains the list of messages that may be issued during the execution of the LCP. In each messages, xxxx represents an identification number of up to four digits which appears to the right of the converted line in the output listing and in the corresponding card image if the option DECK has been specified. The last digit of this number is that of the ten positions; the units position is not printed.

IPB001I READ OR WRITE STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: If the READ or WRITE statement contains indexed I/O lists and/or arrays, and the FORMAT statement referred to contains literal data or an A- or H-format code, conversion of these items is incorrect.

In the conversion of a READ or WRITE statement using NAMELIST, it should be noted that:

- A DEFINED item cannot appear in the data list in PL/I.
- On input, no search is made for a specific NAMELIST name

Required Action: For transmission of a character string appearing among the values of an indexed list or of an array, the user should:

- On input, create a dummy variable containing the character string and insert it in the list.
- On output, either proceed as above, or insert the character string itself in the list.

For a DEFINED item appearing in a data-list, the user should replace this item by a dummy variable both in the data list and in the data.

IPB002I ENTRY STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: On entry to a function or a subroutine, initialization of parameter made on a previous entry may be lost.

Required Action: If needed, the user should insert additionnal dummy formal parameters or use dummy static variables.

IPB003I ASSIGNED GOTO STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: An assigned variable cannot be used for any purpose other than for the assigned GOTO statement.

Required Action: If a label used in an ASSIGN and in an assigned GOTO statement is also used as a variable elsewhere in the program, the label should be changed.

IPB004I COMMON/EQUIVALENCE STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: In general, the conversion of integer variables two bytes in length and of logical variables will result in incorrect addressing. This may also apply to elements with the CHARACTER attribute.

Required Action: The user should insert additional dummy variables to provide correct alignment.

IPB005I ARITHMETIC ASSIGNMENT STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: The right-hand part of the assignment statement is an expression of REAL type and the left-hand part is of INTEGER type. Due to difference of implementation, the results of the truncation may differ.

Required Action: It is the user's responsibility to check whether truncation due to conversion gives the expected result. The built-in function CEIL may be used if the expected result is not produced.

IPB006I SUBSCRIPT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: A subscript expression contains the operators ** and/or /, or a left parenthesis.

Required Action: If the subscript contains integer division, the user should insert the built-in function TRUNC. If it contains a subscripted variable, the user must reverse the order of the subscripts. The operation ** may give a result of REAL type. The user must verify that the result of the truncation is correct.

IPB007I FORMAT STATEMENT
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: PRINT and non-PRINT files have the same format, or one (or more) of the following items is detected:

- G-, Z- and O- format codes (not converted)
- P-scale factors associated with E- or D-format codes (not converted)
- L-format code.

Required Action:

1. PRINT and non-PRINT files:

The user must specify two FORMAT statements, one for PRINT files and another for non-PRINT files.

- G-, Z- and O-format codes:

The user must replace these format codes by a type acceptable to PL/I.

- P-scale factors:

The user should modify the corresponding data

- L-format code:

Logical data in the external medium must be in a form acceptable to PL/I.

IPB008I SERVICE SUBPROGRAMS
PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE xxxx

Explanation: This message is issued when a call to subprograms EXIT, DUMP, PDUMP, OVERFL, DVCHK, SLITE and SLITET is detected.

Required Action:

1. EXIT:
The CALL statement has been converted to STOP if EXIT has no argument list. If the user provides his own EXIT procedure, he must modify the converted statement. Otherwise, no action is required.

2. Other service subprograms:
If the user provides his own procedures, no action is required. Otherwise, the user may
- For DUMP and PDUMP: use the procedure IHEDUMP (or a PUT statement).
- For OVERFL and DVCHK: use ON-condition OVERFLOW or ZERODIVIDE.

IPB009I LITERAL PASSED AS ARGUMENT IN LINE xxxx
Explanation: A literal constant may not be passed as an argument to a subprogram.

Required Action: The user should declare with the CHARACTER(*) attribute the corresponding parameter in the called subprogram.

IPB010I HEXADECIMAL/OCTAL CONSTANT NOT CONVERTED IN LINE xxxx

Required Action: The user should modify the type of the constant.

IPB011I MATHEMATICAL FUNCTION NOT CONVERTED IN LINE xxxx

Explanation: This message is issued when the mathematical FORTRAN functions DIM, IDIM, MOD, AMOD, DMOD, SNGLE, and DBLE are invoked.

Required Action:

1. SNGLE, DBLE:
These functions should be replaced where they appear by the PL/I built-in function PRECISION.

2. DIM, IDIM:
These functions should be replaced by the expression

conv<argument1>-MIN(conv<argument1>,conv<argument2>)

3. MOD, AMOD, DMOD:
These functions are converted using the PL/I built-in function MOD. If the first argument could be lower than zero, the user must replace the conversion by the expression:

conv<argument1>-TRUNC(conv<argument1>/conv<argument2>)
*conv<argument2>

IPB012I DATA SET REFERENCE NUMBER IS VARIABLE IN LINE xxxx

Explanation: A data set number in the FORTRAN statement is an integer variable. The statement is not converted.

Required Action: The user may simulate the effect of FORTRAN by using a series of IF statements to test the values of the data set reference number. For example:

FORTRAN
WRITE(N,10)<data list>

PL/I
```
IF N=1 THEN PUT(FT01F01)EDIT(conv<data list>)(R(EXTLAB10));
IF N=2 THEN PUT(FT02F01)EDIT(conv<data list>)(R(EXTLAB10));
...
IF N=p THEN PUT(FT0pF01)EDIT(conv<data list>)(R(EXTLAB10));
```

where p is the maximum number of data sets

IPB013I FORMAT REFERENCE IS ARRAY NAME IN LINE xxxx

Explanation: A FORMAT statement is referred to by an array name. The statement is not converted.

Required Action: Attach the FORMAT statement to the input/output statement.

IPB014I BINARY DATA TRANSMISSION IN LINE xxxx

Explanation: Binary data transmission is indicated in the FORTRAN statement. The statement is not converted.

Required Action: Use the PL/I RECORD I/O facility.

IPB015I BACKSPACE, REWIND, OR END FILE AT LINE xxxx

Explanation: The FORTRAN statement is a BACKSPACE, REWIND, or END FILE statement. The statements BACKSPACE and END FILE are not converted; the statement REWIND is converted into the PL/I statement CLOSE, but the tape may be incorrectly positioned as a result.

Required Action: None.

IPB016I SYNTACTICAL ERROR IN LINE xxxx

Explanation: A FORTRAN statement is syntactically incorrect. It is not converted.

Required Action: Suppress error.

IPB017I EQUIVALENCE STATEMENT. CONVERSION WITH DEFINED ITEM MAY PRODUCE MESSAGE AT COMPILATION.

Explanation: The conversion of the EQUIVALENCE statement uses a PL/I (F) compiler facility; at compilation time an error message may be issued if the attributes of the DEFINED item differ from those of the base item, but execution is not prevented.

Required Action: None.

IPB018I THE FOLLOWING STRING NOT IDENTIFIED AS AN OPTION - YYYYYYYY

Explanation: The LCP is processing the option list passed to it as a parameter, when it finds a character string that it cannot identify as an option. The unidentifiable character string is ignored.

Required Action: Correct the erroneous parameter.

IPB019I FILE SYSUT2-INEXPLICABLE I/O ERROR

Explanation: One of the following I/O errors has occurred:

- Space allocation for SYSUT2 insufficient
- Permanent I/O error on disk. (Hardware fault)

Required Action: In first case, increase SYSUT2 space allocation. In second case, change disk or disk-drive.

Note: This message is always followed by the completion code ABEND 400.

APPENDIX G: PREPARATION OF DATA

The methods of entering data differ, in certain respects, in FORTRAN and in PL/I.

Where applicable, the user should therefore modify the data as follows:

1. For numeric data, the option BLKZR automatically makes the following alterations:

   - Plus sign in BCDIC (&) is replaced by +
   - D is changed to E
   - An all blank field is replaced by 0
   - If a blank appears after E, a + replaces it
   - Other embedded and trailing blanks are replaced by zeros.

   For other types of data, the user must do his own modification. In particular, it must be noted that the PL/I (F) compiler does not accept an exponent of more than 2 digits.

2. The items of FORTRAN logical data TRUE (or T) and FALSE (or F) should be changed into 1 and 0 respectively.

3. Data pertaining to a FORTRAN NAMELIST statement should be modified to ensure that:

   - The NAMELIST name is suppressed and the end-of-data group (&END) is replaced by a semicolon.
   - The repetition factors, if any, are expanded, and each value in a data list assigned to the corresponding element of the array.
   - The order of subscripts attached to variables is reversed.
   - The data must be in EBCDIC.

The disk pack or the tape distributed by IBM contains, in addition to the LCP, a sample program written in FORTRAN IV. The purpose of the sample program is to demonstrate the working of the LCP and to illustrate the explanations given in the various sections of this manual.

Once the load module for the LCP has been created, there are three steps to be performed:

1. Extraction of the sample program with its associated data from the disk pack or tape, and transfer onto punched cards

2. Execution of a conversion run for the program

3. Execution of a compile, link, and go run with the PL/I compiler, using the output from step 2 and the data from step 1

These steps are described in detail in the paragraphs that follow.

Step 1. Extraction of Sample Program

a. For users receiving their program on disk pack: The FORTRAN IV program and its data are written on the disk pack as a single data set (DSNAME=SAMPLE). The following is an example of the control cards required to obtain the punched cards:

```
//AB        JOB   4727,SMITH,MSGLEVEL=1

//ABA       EXEC  PGM=IEBPTPCH

//SYSPRINT  DD    SYSOUT=A

//SYSUT2    DD    SYSOUT=B

//SYSUT1    DD    UNIT=2311,                    C
//              DISP=OLD,                     C
//              DSNAME=SAMPLE,                C
//              VOLUME=SER=FLCPRS

//SYSIN     DD    *

  PUNCH
```

```
/*
```

b. For users receiving their program on tape: The FORTRAN IV program and its data are written on tape as a single file. The following is an example of the control cards required to obtain the punched cards:

```
//AB         JOB    4727,SMITH,MSGLEVEL=1
//ABA        EXEC   PGM=IEBPTPCH
//SYSPRINT   DD     SYSOUT=A
//SYSUT1     DD     UNIT=2400,LABEL=(3,NL),                    C
//                  VOLUME=SER=888888,                         C
//                  DCB=(RECFM=FB,LRECL=80,BLKSIZE=2400)
//SYSUT2     DD     SYSOUT=B
//SYSIN      DD     *
  PUNCH
/*
```

Step 2. Program Conversion Run

The following is an example of the control cards required for a program conversion run:

```
| //AB        JOB     4727,SMITH,MSGLEVEL=1
  //JOBLIB    DD      DSNAME=FORLCP,DISP=OLD,UNIT=2311,                C
  //                  VOLUME=SER=xxxxxx
  //E         EXEC    PGM=LCPFORT,PARM='SOURCE,DECK,BLKCR'
  //SYSPRNT   DD      SYSOUT=A,DCB=(RECFM=VA,BLKSIZE=129,LRECL=125)
| //SYSIOERR    DD    SYSOUT=A
  //SYSUT1    DD      DSNAME=UT2,SPACE=(CYL,(5,)),                     C
  //                  DCB=(DSORG=DA,KEYLEN=9),UNIT=2311
  //SYSUT2    DD      DSNAME=UT3,SPACE=(CYL,(2,)),DCB=(DSORG=DA),      C
  //                  UNIT=2311
| //SYSPCH      DD    SYSOUT=B
  //SYSIN     DD      *
  ...
  ...     The FORTRAN program cards:  SAMPR010 through SAMPR960
  ...
  /*
```

Since the option SOURCE appears in the EXEC card, a listing of the FORTRAN source program and of its translation into PL/I should appear on the printer as follows:

```
      FORTRAN IV TO PL/I LCP - V1L0

C     SIMULTANEOUS EQUATION ROUTINE                                          SAMPR010
C              THE FOLLOWING DATA SHOULD BE OUTPUT BY THE PROGRAM.           SAMPR020
C                     MATRIX A                                               SAMPR030
C   4.2150   -1.2120     1.1050                                              SAMPR040
C  -2.1200    3.5050    -1.6320                                              SAMPR050
C   1.1220   -1.3130     3.9860                                              SAMPR060
C                     MATRIX B                                               SAMPR070
C   3.2160                                                                   SAMPR080
C   1.2470                                                                   SAMPR090
C   2.3456                                                                   SAMPR100
C                      A-INVERSE                                             SAMPR110
C   0.2916    0.0833    -0.0467                                              SAMPR120
C   0.1632    0.3836     0.1118                                              SAMPR130
C  -0.0283    0.1029     0.3009                                              SAMPR140
C                     SOLUTION MATRIX                                        SAMPR150
C   0.9321                                                                   SAMPR160
C   1.2655                                                                   SAMPR170
C   0.7429                                                                   SAMPR180
      DIMENSION A(10,10),X(10),B(10)                                         SAMPR190
  301 FORMAT(1H1,10X,15HINCOMPATIBILITY)                                     SAMPR200
  302 FORMAT(1H ,10X,41HMORE EQUATIONS THAN UNKNOWNS-NO SOLUTIONS)           SAMPR210
  303 FORMAT(1H ,10X,46HMORE UNKNOWNS THAN EQUATIONS-SEVERAL SOLUTIONS)      SAMPR220
  304 FORMAT(1H0,10X,15HSOLUTION MATRIX)                                     SAMPR230
  305 FORMAT(1H1,10X,8HMATRIX A)                                             SAMPR240
  306 FORMAT(1H0,10X,8HMATRIX B)                                             SAMPR250
  307 FORMAT(1H0,08X,10H A-INVERSE)                                          SAMPR260
  308 FORMAT(1H ,10X,24HDIAGONAL ELEMENT IS ZERO)                            SAMPR270
   12 FORMAT (6I10)                                                          SAMPR280
      READ  (5,12) M1,M2,L1,L2,N1,N2                                         SAMPR290
C     M1 = NO. OF ROWS OF A                                                  SAMPR300
C     M2 = NO. OF COLS OF A                                                  SAMPR310
C     L1 = NO. OF ROWS OF X                                                  SAMPR320
C     L2 = NO. OF COLS OF X                                                  SAMPR330
C     N1 = NO. OF ROWS OF B                                                  SAMPR340
C     N2 = NO. OF COLS OF B                                                  SAMPR350
   13 FORMAT (7F10.4)                                                        SAMPR360
   17 FORMAT (10F10.4)                                                       SAMPR370
      IF (N2-1) 63,64,63                                                     SAMPR380
   64 IF (L2-1) 63,65,63                                                     SAMPR390
   65 IF (L1-M2) 63,66,63                                                    SAMPR400
   66 IF (M1-N1) 63,11,63                                                    SAMPR410
   63  WRITE (6,301)                                                        SAMPR420
      GO TO 2                                                                SAMPR430
   11 N=M1                                                                   SAMPR440
      N=M2                                                                   SAMPR450
      IF (M1-M2) 91,14,93                                                    SAMPR460
```

```
 91  WRITE (6,302)                                                  SAMPR470
     GO TO 2                                                        SAMPR480
 93  WRITE (6,303)                                                  SAMPR490
     GO TO 2                                                        SAMPR500
 14  WRITE (6,305)                                                  SAMPR510
     DO 70 I=1,N                                                    SAMPR520
      READ  (5,13)(A(I,J),J=1,N)                                    SAMPR530
      WRITE (6,17)(A(I,J),J=1,N)                                    SAMPR540
 70 CONTINUE                                                        SAMPR550
 89 FORMAT (F10.4)                                                  SAMPR560
      WRITE (6,306)                                                 SAMPR570
      READ  (5,89)(B(I),I=1,N)                                      SAMPR580
      WRITE (6,89)(B(I),I=1,N)                                      SAMPR590
 20 DO 120 K=1,N                                                    SAMPR600
     D=A(K,K)                                                       SAMPR610
     IF(D)40,200,40                                                 SAMPR620
 40 A(K,K)=1.0                                                      SAMPR630
 50 DO 60 J=1,N                                                     SAMPR640
 60 A(K,J)=A(K,J)/D                                                 SAMPR650
     IF(K-N)80,130,130                                              SAMPR660
 80  IK=K+1                                                         SAMPR670
     DO 120 I=IK,N                                                  SAMPR680
     E=A(I,K)                                                       SAMPR690
     A(I,K)=0.0                                                     SAMPR700
     DO 120 J=1,N                                                   SAMPR710
120 A(I,J)=A(I,J)-(D*A(K,J))                                        SAMPR720
130 IK=N-1                                                          SAMPR730
     DO 180 K=1,IK                                                  SAMPR740
140 I1=K+1                                                          SAMPR750
     DO 180 I=I1,N                                                  SAMPR760
     E=A(K,I)                                                       SAMPR770
     A(K,I)=0.0                                                     SAMPR780
170 DO 180 J=1,N                                                    SAMPR790
180 A(K,J)=A(K,J)-(D*A(I,J))                                        SAMPR800
     GO TO 202                                                      SAMPR810
200  WRITE (6,308)                                                  SAMPR820
     GO TO 2                                                        SAMPR830
202  WRITE (6,307)                                                  SAMPR840
     DO 201 I=1,N                                                   SAMPR850
      WRITE (6,17)(A(I,J),J=1,N)                                    SAMPR860
201 CONTINUE                                                        SAMPR870
400 DO 21 I=1,N                                                     SAMPR880
     X(I)=0.0                                                       SAMPR890
     DO 21 K=1,N                                                    SAMPR900
 21 X(I)=X(I)+A(I,K)*B(K)                                           SAMPR910
401 WRITE(6,304)                                                    SAMPR920
      WRITE (6,89)(X(I),I=1,N)                                      SAMPR930
  2  CALL EXIT                                                      SAMPR940
     STOP                                                           SAMPR950
     END                                                            SAMPR960
```

```
      (NCZERCCIVICE):  MAINPRC: PROCEDURE OPTIONS(MAIN);                MAIC0010
      DECLARE IK FIXED BINARY(31) STATIC,I1 FIXED BINARY(31) STATIC,N2 MAIC0020
      FIXED BINARY(31) STATIC,N1 FIXED BINARY(31) STATIC,L2             MAIC0030
      FIXED BINARY(31) STATIC,L1 FIXED BINARY(31) STATIC,M2             MAIC0040
      FIXED BINARY(31) STATIC,M1 FIXED BINARY(31) STATIC,D              MAIC0050
      FLOAT BINARY STATIC,K FIXED BINARY(31) STATIC,J FIXED BINARY(31) MAIC0060
      STATIC,N FIXED BINARY(31) STATIC,I FIXED BINARY(31) STATIC,B(     MAIC0070
          10) FLOAT BINARY STATIC,X(    10) FLOAT BINARY STATIC,A(      MAIC0080
          1C,    10) FLOAT BINARY STATIC;                               MAIC0090
      DECLARE DUMM008 CHARACTER( 15) INITIAL('INCOMPATIBILITY');        MAIC0100
      DECLARE DUMM007 CHARACTER( 41) INITIAL('MCRE EQUATIONS THAN UNKNOMAIC0110
WNS-NO SCLUTIONS');                                                     MAIC0120
      DECLARE DUMM006 CHARACTER( 46) INITIAL('MCRE UNKNOWNS THAN EQUATIMAIC0130
CNS-SEVERAL SOLUTIONS');                                                MAIC0140
      DECLARE DUMM005 CHARACTER( 15) INITIAL('SOLUTION MATRIX');        MAIC0150
      DECLARE DUMMC04 CHARACTER(  8) INITIAL('MATRIX A');               MAIC0160
      DECLARE DUMM003 CHARACTER(  8) INITIAL('MATRIX B');               MAIC0170
      DECLARE DUMM002 CHARACTER( 10) INITIAL(' A-INVERSE');             MAIC0180
      DECLARE DUMMC01 CHARACTER( 24) INITIAL('DIAGONAL ELEMENT IS ZERO'MAIC0190
);                                                                      MAIC0200
      DECLARE LBLNK ENTRY(FIXED BINARY)RETURNS(FIXED BINARY);           MAIC0210
      DECLARE(FT06F01)PRINT FILE;                                       MAIC0220
/*     SIMULTANEOUS EQUATION ROUTINE*/                                  MAIC0230
/*             THE FCLLCWING DATA SHOULD BE OUTPUT BY THE PROGRAM.*/    MAIC0240
/*                    MATRIX A*/                                        MAIC0250
/*   4.215C   -1.2120    1.1050*/                                       MAIC0260
/*  -2.12CC    3.5C5C   -1.6320*/                                       MAIC0270
/*   1.1220   -1.313C    3.9860*/                                       MAIC0280
/*                    MATRIX B*/                                       MAIC0290
/*   3.2160*/                                                         MAIC0300
/*   1.247C*/                                                         MAIC0310
/*   2.3456*/                                                         MAIC0320
/*                     A-INVERSE*/                                     MAIC0330
/*   C.2916    C.0833   -C.C467*/                                       MAIC0340
/*   C.1632    C.3836    C.1118*/                                       MAIC0350
/*  -C.C283    0.1029    C.3C09*/                                       MAIC0360
/*                    SCLUTION MATRIX*/                                MAIC0370
/*   0.9321*/                                                         MAIC0380
/*   1.2655*/                                                         MAIC0390
/*   C.7429*/                                                         MAIC0400
EXTLAB3C1:FCRMAT(PAGE,X(1C),A(15));                                     MAIC0410
EXTLAB302:FORMAT(COLUMN(1),X(10),A(41));                                MAIC0420
EXTLAB303:FORMAT(COLUMN(1),X(10),A(46));                                MAIC0430
EXTLAB304:FORMAT(SKIP(2),X(10),A(15));                                  MAIC0440
EXTLAB305:FCRMAT(PAGE,X(10),A(8));                                      MAIC0450
EXTLAB306:FCRMAT(SKIP(2),X(10),A(8));                                   MAIC0460
EXTLAB3C7:FCRMAT(SKIP(2),X(08),A(10));                                  MAIC0470
EXTLAB3C8:FORMAT(COLUMN(1),X(10),A(24));                                MAIC0480
EXTLAB12:FORMAT(COLUMN(1),6 F(LBLNK(10)));                              MAIC0490
```

```
            GET FILE(FT05F01)EDIT(M1,M2,L1,L2,N1,N2)(R(EXTLAB12));                MAI00500
          /*      M1 = NO. OF ROWS OF A*/                                         MAI00510
          /*      M2 = NO. OF COLS OF A*/                                         MAI00520
          /*      L1 = NO. OF ROWS OF X*/                                         MAI00530
          /*      L2 = NO. OF COLS OF X*/                                         MAI00540
          /*      N1 = NO. OF ROWS OF B*/                                         MAI00550
          /*      N2 = NO. OF COLS OF B*/                                         MAI00560
          EXTLAB13:FORMAT(COLUMN(1),7 F(LBLNK(10),4));                            MAI00570
          EXTLAB17:FORMAT(COLUMN(1),10 F(10,4));                                  MAI00580
                IF(N2-1)¬= 0 THEN GO TO EXTLAB63;                                 MAI00590
          EXTLAB64:IF(L2-1)¬= 0 THEN GO TO EXTLAB63;                              MAI00600
          EXTLAB65:IF(L1-M2)¬= 0 THEN GO TO EXTLAB63;                             MAI00610
          EXTLAB66:IF(M1-N1) = 0 THEN GO TO EXTLAB11;                             MAI00620
          EXTLAB63:PUT FILE(FT06F01)EDIT(DUMM008)(R(EXTLAB301));                  MAI00630
                GOTO EXTLAB2;                                                     MAI00640
          EXTLAB11:N=M1 ;                                                         MAI00650
                N=M2 ;                                                            MAI00660
                IF(M1-M2) = 0 THEN GO TO EXTLAB14;ELSE IF(M1-M2) > 0 THEN GO TO   MAI00670
                EXTLAB93;                                                         MAI00680
          EXTLAB91:PUT FILE(FT06F01)EDIT(DUMM007)(R(EXTLAB302));                  MAI00690
                GOTO EXTLAB2;                                                     MAI00700
          EXTLAB93:PUT FILE(FT06F01)EDIT(DUMM006)(R(EXTLAB303));                  MAI00710
                GOTO EXTLAB2;                                                     MAI00720
          EXTLAB14:PUT FILE(FT06F01)EDIT(DUMM004)(R(EXTLAB305));                  MAI00730
                DO I=1 TO MAX(1,N);                                               MAI00740
IPB001I         GET FILE(FT05F01)EDIT((A(J,I) DO J=1 TO MAX(1,N)))(R(EXTLAB13));  MAI00750
IPB001I         PUT FILE(FT06F01)EDIT((A(J,I) DO J=1 TO MAX(1,N)))(R(EXTLAB17));  MAI00760
          EXTLAB70:;                                                              MAI00770
                END;                                                              MAI00780
          EXTLAB89:FORMAT(COLUMN(1),F(LBLNK(10),4));                              MAI00790
                PUT FILE(FT06F01)EDIT(DUMM003)(R(EXTLAB306));                     MAI00800
IPB001I         GET FILE(FT05F01)EDIT((B(I) DO I=1 TO MAX(1,N)))(R(EXTLAB89));    MAI00810
IPB001I         PUT FILE(FT06F01)EDIT((B(I) DO I=1 TO MAX(1,N)))(R(EXTLAB89));    MAI00820
          EXTLAB20:DO K=1 TO MAX(1,N);                                            MAI00830
                D=A(K,K) ;                                                        MAI00840
                IF(D) = 0 THEN GO TO EXTLAB200;                                   MAI00850
          EXTLAB40:A(K,K)=1.0E0 ;                                                 MAI00860
          EXTLAB50:DO J=1 TO MAX(1,N);                                            MAI00870
          EXTLAB60:A(J,K)=A(J,K)/D ;                                              MAI00880
                END;                                                              MAI00890
                IF(K-N)>= 0 THEN GO TO EXTLAB130;                                 MAI00900
          EXTLAB80:IK=K+1 ;                                                       MAI00910
                DO I=IK TO MAX(IK,N);                                             MAI00920
                D=A(K,I) ;                                                        MAI00930
                A(K,I)=0.0E0 ;                                                    MAI00940
                DO J=1 TO MAX(1,N);                                               MAI00950
          EXTLAB120:A(J,I)=A(J,I)-(D*A(J,K)) ;                                    MAI00960
                END;                                                              MAI00970
                END;                                                              MAI00980
                END;                                                              MAI00990
```

```
          EXTLAB13C:IK=N-1 ;                                                                    MAIC1000
                DO K=1 TO MAX(1,IK);                                                            MAI01010
          EXTLAB14C:I1=K+1 ;                                                                    MAI01020
                DO I=I1 TO MAX(I1,N);                                                           MAI01030
                C=A(I,K) ;                                                                      MAI01040
                A(I,K)=0.0E0 ;                                                                  MAIC1050
          EXTLAB17C:DO J=1 TO MAX(1,N);                                                         MAI01060
          EXTLAB18C:A(J,K)=A(J,K)-(C*A(J,I)) ;                                                  MAI01070
                END;                                                                            MAI01080
                END;                                                                            MAI01090
                END;                                                                            MAI01100
                GOTO EXTLAB202;                                                                 MAI01110
          EXTLAB200:PUT FILE(FTC6F01)EDIT(DUMM001)(R(EXTLAB308));                               MAI01120
                GOTO EXTLAB2;                                                                   MAIC1130
          EXTLAB202:PUT FILE(FT06F01)EDIT(DUMM002)(R(EXTLAB307));                               MAI01140
                DO I=1 TO MAX(1,N);                                                             MAI01150
| IPB001I       PUT FILE(FT06F01)EDIT((A(J,I) DO J=1 TO MAX(1,N)))(R(EXTLAB17));               MAI01160
          EXTLAB201:;                                                                           MAI01170
                END;                                                                            MAI01180
          EXTLAB400:DO I=1 TO MAX(1,N);                                                         MAIC1190
                X(I)=0.0E0 ;                                                                    MAI01200
                DO K=1 TO MAX(1,N);                                                             MAIC1210
          EXTLAB21:X(I)=X(I)+A(K,I)*B(K) ;                                                      MAI01220
                END;                                                                            MAI01230
                END;                                                                            MAI01240
          EXTLAB401:PUT FILE(FTC6F01)EDIT(DUMMC05)(R(EXTLAB3C4));                               MAI01250
| IPB001I       PUT FILE(FT06F01)EDIT((X(I) DO I=1 TO MAX(1,N)))(R(EXTLAB89));                 MAI01260
| IPB008I EXTLAB2:STOP;                                                                        MAI01270
                DISPLAY('STOP ');STOP;                                                          MAI01280
                END;                                                                            MAIC1290
```

WARNING MESSAGES

```
| IPB001I READ OR WRITE STATEMENT
          PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE
            0075,0076,0081,0082,0116,0126

| IPB008I SERVICE SUBPROGRAMS
          PL/I AND FORTRAN RESULTS MAY DIFFER IN LINE
            0127
```

Step 3. Execution of PL/I Program

A normal compile, link, and go run can be performed, with or without the user's cataloged procedure. The following is a sample of the control cards when using a cataloged procedure:

```
//AB        JOB    2622,SMITH,MSGLEVEL=1
//          EXEC   PL1LFCLG
//SYSIN     DD     *
...
...     Card output from step 2:  cards MAI00010 through MAI01290
...
/*
//LKED.SYSIN DD      DSNAME=LBLNK,DISP=OLD,UNIT=SYSDA                C
//                   VOLUME=SER=XXXXXX
//GO.FT06F01 DD      SYSOUT=A,DCB=(BLKSIZE=129,LRECL=125,RECFM=VA)
//GO.FT05F01 DD      *
...
...     The seven FORTRAN data cards:  DATA0001 through DATA0007
...
/*
```

The output listed on FT06F01 (the printer specified in the above control cards) should read as follows:

```
        MATRIX A
 4.2150   -1.2120    1.1050
-2.1200    3.5050   -1.6320
 1.1220   -1.3130    3.9860

        MATRIX B
 3.2160
 1.2470
 2.3456

      A-INVERSE
 0.2916    0.0833   -0.0467
 0.1632    0.3836    0.1118
-0.0283    0.1029    0.3009

        SOLUTION MATRIX
 0.9321
 1.2655
 0.7429
```

INDEX

Where more than one reference is given, the first page number indicates the major reference.

FORTRAN IV-to-PL/I LCP Printed in U.S.A. GC33-2002-2

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]