GC33-0001-2
File No. S360/S370-29

Program Product

OS
PL/I Optimizing Compiler: General Information

Program Number 5734-PL1
(This program product is available
as part of composite package 5734-PL3)

IBM

Revised June, 1973
By TNL: GN33-6095

Preface

This publication gives general information about the OS PL/I Optimizing Compiler, a processing program of the IBM Operating System.

The information provided is to be used as a planning aid only, and is intended to introduce installation managers, systems analysts, and programmers to the facilities available with this compiler. The reader is assumed to be conversant with the IBM Operating System and with the PL/I language as implemented by version 5 of the PL/I (F) Compiler.

Chapter 1 discusses how the performance of the OS PL/I Optimizing Compiler compares with that of version 5 of the PL/I (F) Compiler; the compiler options that are available; the debugging aids designed to increase programmer productivity; the degree to which the compiler is compatible with version 5 of the PL/I (F) Compiler; and the features designed to allow communication with assembler programs and with programs written in COBOL or FORTRAN.

Chapter 2 is a discussion of the methods used for the optimization process, such as expression simplification or the substitution of in-line code in place of calls to library subroutines.

Chapter 3 is about the PL/I language itself, including the features that can be used with this compiler, and a comparison of the language implemented by the optimizing compiler with that implemented by version 5 of the PL/I (F) Compiler.

Chapter 4 briefly describes both the machine and the operating system requirements of the optimizing compiler.

Appendix A is a list of the PL/I and the implementation-defined keywords used by this compiler.

Appendix B documents the most significant incompatibilities that might cause a program written for version 5 of the PL/I (F) Compiler to give errors when run on the optimizing compiler.

Appendix C is a bibliography of related IBM publications and their order numbers.

Appendix D provides planning information for the use of the optimizing compiler under CMS.

Third Edition (December 1972)

This is a major revision of, and obsoletes, GC33-0001-1 and Technical Newsletters GN33-6069, GN33-6057, GN33-6044, GN33-6034, and GN33-6024. Changes or additions to the text and figures are indicated by a vertical line to the left of the change.

This edition applies to Version 1, Release 2, Modification 0 of the OS PL/I Optimizing Compiler, Program Product 5734-PL1, and to all subsequent versions, releases, or modifications until otherwise indicated in new editions or technical newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822, and associated technical newsletters, for the editions that are applicable and current.

VSAM material in this publication is for planning purposes only. CMS material in this publication applies to Release 1, PLC5(Program Level Change 5) of the Virtual Machine Facility/370 (VM/370).

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM United Kingdom Laboratories Ltd., Publications Department, Hursley Park, Winchester, Hampshire, England. Comments become the property of IBM.

Contents

Figures

Introduction

The OS PL/I Optimizing Compiler is designed for the compilation of programs written in PL/I into efficient object programs. It requires a minimum of 50K bytes of main storage on an IBM System/360 or System/370 with the universal instruction set and is intended to meet the requirements of the PL/I user of medium-sized or larger installations. Good all-round performance is achieved by a compiler design that incorporates the best design features of a number of well-proven compilers, including the PL/I (D), PL/I (F), FORTRAN IV(G), FORTRAN IV(H), and COBOL (F) compilers. The main features include:

- Extensive Optimization - if optional optimization is specified, the machine instructions generated will be optimized using the features described in chapter 2, to produce a very efficient object program. Some optimization is always performed even if the option is not specified. A secondary effect of optimization could be a reduction in the amount of main storage required for the object program during execution. If optional optimization is not specified, compilation time will be reduced.

- Advanced Level of PL/I - an implementation of PL/I with language extensions beyond the language implemented by the PL/I (F) Compiler. New features for OS users include:

  DEFAULT statement

  Entry variables

  File variables

- Extensive Debugging Aids - time and effort required for program checkout are minimized by:

  Extensive implementation of on-units

  Support of the CHECK condition

  Data-directed input/output

  Comprehensive range of compiler options

  Clear and precise compile-time and execution-time diagnostic messages

  Optional statement number trace facilities

  Optional statement frequency counting facility.

The OS PL/I Optimizing Compiler translates PL/I source statements into machine instructions. However, it does not generate all the machine instructions required to represent the source program; in some cases it inserts references to subroutines that are stored in the resident library. During link-editing, subroutines from the resident library are included in the object module. These subroutines may contain references to other subroutines stored in the transient library; during execution, subroutines from the transient library are loaded, executed, and discarded as required.

The functions provided by the resident library are as follows:

- mathematical subroutines.
- data type conversions.
- edit-, list-, and data-directed I/O (stream-oriented transmission).
- program initialization.
- storage management.
- display.
- timer facilities.
- error handling.

The functions provided by the transient library are as follows:

- print and message subroutines of the error and interrupt handling subroutines.
- opening and closing files.
- record-oriented and stream-oriented transmission.

These libraries are not an integral part of the OS PL/I Optimizing Compiler; they consist of the following separate IBM program products:

- OS PL/I Resident Library, Program Number 5734-LM4
- OS PL/I Transient Library, Program Number 5734-LM5

Throughout this publication the terms "resident library" and "transient library" refer to these IBM program products.

Chapter 1: The OS PL/I Optimizing Compiler

The OS PL/I Optimizing Compiler allows the programmer the choice of fast compilation of his source program or fast execution of the resulting object program. It does so by means of a compiler option specifying the type of optimization required.

The compiler provides debugging aids to minimize the time and effort required for program checkout. Options are provided for use by the programmer to specify information or to request optional compiler facilities. The compiler is compatible with version 5 of the PL/I(F) Compiler in terms of source language, except for a few minor differences, and there are features that allow programs compiled by the optimizing compiler to communicate with programs written in other languages.

Compilation Speed

Compilation speed varies with the machine configuration, the version of the operating system, data set disposition, partition or region size, the complexity of the program, and the language features and compiler options employed. In general, faster compilation will be achieved with larger partitions or regions and minimum use of compiler options. Use of the IBM 3330 Disk Storage for compiler residence and the compiler spill file can decrease compilation time.

For programs that use the MACRO option only to include text from a source statement library, the INCLUDE option may be used instead, giving a decrease in compilation time.

When optimization is requested, compilation times will increase by an average of 25%; in some cases, the increase may be as much as 100%.

Execution Speed

During testing of the compiler with release 20.1 of the operating system, the total CPU time during the execution steps of a sample of scientifically-oriented programs that had been optimized was 50% less than that of the same sample compiled by version 5 of the PL/I (F) Compiler with a range of saving on individual jobs between 0% and 63%.

The total CPU time of a sample of commercially-oriented programs that had been optimized was 30% less than that of the same sample compiled by version 5 of the PL/I (F) Compiler with a range of saving on individual jobs between 0% and 59%.

Within the samples, the least improvement was typically shown by the shortest programs. The amount of improvement for any particular program depends on factors such as the overheads for initialization and input/output and the nature of the internal processing of the program. Programs most likely to show appreciable improvements in execution speed are those that contain features mentioned in chapter 2.

Space Requirements

The main storage requirements of the object programs produced by the optimizing compiler depend on the PL/I facilities used. Space is required not only for the object code generated by the compiler, but also for the resident library subroutines that will be link-edited with it to form a load module and for the transient library subroutines that are called when this load module is executed. If optimization is specified, the object code will usually require less space.

Debugging Aids

The optimizing compiler provides the following debugging aids to minimize the time and effort needed for program checkout:

Diagnostics: Comprehensive diagnostic messages are provided during both compilation and execution. In processing an erroneous statement, the compiler may attempt to correct the error by making an assumption about the intention of the statement. Messages produced during compilation will be for errors in the categories of unrecoverable, severe error, error, warning, and informatory. The messages will be listed in category groups in ascending statement number order. Each

message will indicate the number of the erroneous statement and, when applicable, the part of the statement involved, the exact nature of the error, and any assumptions made and action taken by the compiler.

Flow Trace: The flow trace facility can be used to produce a record of the changes in sequential flow prior to an interrupt. The trace is included as part of the diagnostic output produced by the SNAP option of the ON statement or by the PLIDUMP built-in subroutine. It contains a list of branch-out/branch-in pairs of statement numbers. The associated control block name (up to 8 characters) is also given. The number of statements to be included in the flow trace is given by the user in requesting the option. This feature is in addition to the optional inclusion of the statement number in diagnostic messages produced at execution time.

Statement Frequency Counting: This facility can be used to identify those parts of a program that are most heavily used and those statements that are never used. The statement count is given at program termination on the PLIDUMP file. If the PLIDUMP file does not exist, SYSPRINT is used.

On-Units: Program condition-handling is facilitated by the provision of on-units for all PL/I conditions. On-units permit either programmer-defined or system-defined action to take place when a particular interrupt occurs during execution. The range of conditions covers all arithmetic interrupts, input/output errors, and special program-checkout conditions such as CHECK, SUBSCRIPTRANGE, STRINGRANGE, SIZE, and STRINGSIZE.

Compiler Options

A number of compiler options are available for use by the programmer to specify information or to request optional compiler facilities.

The options available may be used to:

- Specify whether the source program is coded in the PL/I 48-character set or 60-character set, or punched in BCD or EBCDIC format.
- Specify the margins for source statements.
- Specify a margin indicator character for source statements.
- Control progress into syntax checking depending upon the severity of the diagnostics from previous processing.
- Control progress beyond syntax checking depending on the severity of the diagnostics from previous processing.
- Specify whether full or short diagnostic messages are to be printed.
- List statement numbers of declarations (if applicable) and all attributes assigned to all identifiers in the program.
- List the external symbol dictionary.
- List implementation dependent information (for example, the length of structures).
- Specify the number of lines to be printed on each page of output.
- Print block-levels and do-levels on the source program listing.
- List the options used by the compiler.
- List the source program.
- Specify the minimum severity level at which source program diagnostic messages will be printed.
- List statement numbers of declarations and of statements in which reference is made to all identifiers in the program.
- Specify that an object module is to be produced in a form suitable for input to the linkage editor.
- Specify that a NAME statement is to be produced for an object module.
- Indicate that the source program requires preprocessing.
- List source input to the preprocessor.
- Specify that the output from the preprocessor is to be generated in a card image format suitable for punching.
- Print the contents of registers and main storage in the event of compiler failure.
- Specify the amount of main storage available for compilation.
- Specify whether statement numbers are to be obtained by counting semicolons, or derived from line numbers.
- Specify that selected output from the

for a particular compilation.

- Specify that the output from the compiler is to be generated in a card image format suitable for punching.
- List the numbers of the last "n" discontinuities (for example CALL or GO TO) executed prior to an interrupt. These numbers are included in the output produced by the SNAP option.
- Specify that diagnostic messages produced during execution are to contain source program line numbers.
- Produce additional coding that will allow statement numbers from the source program to be included in messages produced during execution.
- Print tables showing the organization of static storage for the object module.
- Print statement numbers for statements internal to each procedure with their offsets relative to the primary entry point of the procedure.
- Specify optional optimization or not.
- Print a table giving the storage requirements for the object module.

- Count the number of times that statements are executed.
- Indicate that the %INCLUDE statement is the only PL/I preprocessor statement used, permitting faster compilation without the use of the preprocessor.
- List the generated code for the whole program or for a specified range of statements.

Compatibility with the (F) Compiler

Source programs written for version 5 of the PL/I (F) Compiler can be compiled by the Optimizing Compiler and will be executed correctly without modification, except for a few minor differences between the implementations of the PL/I language for the two compilers. These differences are described in appendix B.

Object modules produced by version 5 of the PL/I (F) Compiler, and modules from the PL/I (F) Compiler subroutine library, cannot be incorporated into programs compiled by the optimizing compiler. All PL/I source modules must be recompiled with the Optimizing Compiler, and all library subroutines used must be called from the resident or transient library as appropriate.

Compatibility between Releases

Different release levels of the Optimizing Compiler and its libraries will produce compatible execution, provided that:

1. The release level of the transient library is equal to or greater than that of the resident library.
2. The release level of the resident library is equal to or greater than that of the compiler.

Compatibility between OS and DOS

Because the OS Optimizing Compiler implements more of the PL/I language than the DOS Optimizing Compiler, all source programs compiled on the DOS Optimizing Compiler can be compiled on the OS Optimizing Compiler, thus aiding the conversion from DOS to OS.

Interlanguage Communication

Facilities are available which allow procedures compiled by the Optimizing Compiler to communicate at execution time with programs compiled by any of the following COBOL or FORTRAN compilers produced by IBM for the operating system.

COBOL Compilers:

COBOL (E)	Level E
COBOL (F)	Level F
COBOL ANS(FULL)	Level F

FORTRAN Compilers:

FORTRAN IV(Basic)	Level E
FORTRAN IV	Levels G and H

Existing COBOL and FORTRAN users can write new applications in PL/I while still using existing libraries of COBOL and FORTRAN programs; in addition, existing applications can be modified by the use of PL/I procedures.

Communication between PL/I programs and programs written in different languages is specified in the usual way, by a CALL statement. A FORTRAN program can be called

also by a function reference.

The interlanguage communication facilities are obtained by specifying ASSEMBLER, COBOL, or FORTRAN as an option in a PROCEDURE, ENTRY, or DECLARE statement. (See "Interlanguage Communication," in chapter 3.)

Chapter 2: Optimization

The main aim of the Optimizing Compiler is to generate object programs that can be executed as fast as possible and that occupy as little space as possible during execution. In many cases this will involve generating efficient code for statements in the sequence written by the programmer; in other cases, however, the compiler may alter the sequence of statements or operations to improve the performance while producing the same result.

The following types of optimization are carried out by the compiler:

- Common expression elimination
- Transfer of expressions from loops
- Redundant expression elimination
- Simplification of expressions
- Initialization of arrays and structures
- In-line code for conversions
- In-line code for RECORD I/O
- Key handling for REGIONAL data sets
- Matching format lists with data lists
- In-line code for string manipulation
- In-line code for built-in functions
- Special-case code for DO statements
- Structure and array assignments
- Library subroutines
- Elimination of common constants
- Elimination of common control data
- Use of registers
- Code for program branches

Common Expression Elimination

An expression can occur twice or more in such a way that the flow of control always passes through the first occurrence of the expression to reach a subsequent occurrence. A common expression is an expression that occurs more than once in a program but is obviously intended to result in the same value each time it is evaluated, that is, an expression that is identical to another, with no intervening modification to any operand. The compiler eliminates a common expression by saving the value of the first occurrence of the expression either in a temporary (compiler-generated) variable, or in the program variable to which the result of the expression is assigned. For example:

```
X1 = A1 * B1;
.
.
.
Y1 = A1 * B1;
```

Provided that the values of A1, B1, and X1 do not change between the execution of these statements, the statements can be optimized to the equivalent of the following PL/I statements:

```
X1 = A1 * B1;
.
.
.
Y1 = X1;
```

If the first occurrence of the common expression involves the assignment of the value to a variable that is modified prior to the occurrence of the later expression, the value is assigned to a temporary variable. The example given above would become:

```
TEMP = A1 * B1;
X1 = TEMP;
.
.
X1 = X1 + 2;
.
.
Y1 = TEMP;
```

Also, if the common expression occurs as a subexpression within a larger expression, a temporary variable is created to hold the value of the common subexpression. For example, in the expression C1 + A1 * B1 a temporary variable would be created to hold the value of A1 * B1 if this were a common subexpression.

An important application of this technique occurs in statements containing subscripted variables where the same subscript value is used for each variable. For example:

```
PAYTAX(MNO)=PAYCODE(MNO)*WKPMNT(MNO);
```

The value of the subscript MNO is computed once only when the statement is executed. (The computation would involve the conversion of a value from decimal to binary if MNO were declared a decimal variable.)

Transfer of Expressions from Loops

Where it is possible to produce error-free execution without affecting the results of a program, the optimization process moves invariant expressions or statements from inside a loop to a point which immediately precedes the loop. An expression or statement occurring within a loop is said to be invariant if the compiler can detect that the value of the expression or the action of the statement would be identical for each iteration of the loop. A loop can be either a do-loop or a loop in a program that can be detected by the analysis of the flow of control within the program. For example:

```
DO I =  1 TO N;
B(I) = C(I) * SQRT(N);
P = N * J;
END;
```

This loop can be optimized to produce object code corresponding to the following statements:

```
TEMP = SQRT(N);
P = N * J;
DO I = 1 TO N;
B(I) = C(I) * TEMP;
END;
```

If the programmer wishes this type of optimization to be carried out, he must specify the optimization option REORDER for a BEGIN or PROCEDURE block which contains the loop. If the option is not specified, the default option, ORDER, is assumed, and optimization is inhibited.

Programming Considerations:

1. The transfer of expressions from inside a loop is performed on the assumption that every expression in the loop is executed more frequently than expressions immediately outside the loop. Occasionally this assumption fails, and expressions can be moved out of loops to positions where they are executed more frequently than they would have been if they had remained inside the loop. For example:

```
DO I = J TO K WHILE(X(I)=0);
X(I) = Y(I) * SQRT(N);
END;
```

   The expression SQRT(N) can be moved out of the loop to a position where it is possible for the expression to be executed more frequently than it would be in its original position inside the loop. This undesired effect of optimization can be prevented by the use of the ORDER option for the block in which the loop occurs.

2. Loops are detected by a flow-analysis process. This process can fail to recognize a loop, owing to the existence of flowpaths which the programmer knows will never be used. For example, the use of label variables can inadvertently cause optimization to be inhibited by making the recognition of a desired loop impossible.

Redundant Expression Elimination

A redundant expression is an expression that need not be evaluated for a program to be executed correctly. For example, the logical expression:

```
(A=D)|(C=D)
```

contains the subexpressions (A=D) and (C=D), the second of which need not be evaluated if the first is true. The effect of this optimization is to make the use of a compound logical expression in a single IF statement more efficient than an equivalent series of nested IF statements.

Simplification of Expressions

Simplification of an expression involves its conversion into a form that can be translated into more efficient object code. Where applicable, multiplication and division operations are converted into addition and subtraction operations, which can be performed by faster machine instructions. For example:

```
DO I = 1 TO N BY 2;
   .
   .
   .
A(I) = I * 4;
   .
   .
   .
END;
```

This loop can be optimized to produce

object code corresponding to the following statements:

```
    I = 1;          /* LOOP INITIALIZATION */
    IF I > N THEN GO TO F;
    TEMP = 4 * I;
G:                  /* LOOP ENTRY POINT */
      .
      .
      .
    A(I) = TEMP;    /* EXPRESSION AFTER */
                      /* SIMPLIFICATION */
      .
      .
      .
    I = I + 2;
    TEMP=TEMP + 8;
    IF I < N THEN GO TO G;
                    /* END-OF-LOOP TEST */
F:                   /* LOOP EXIT POINT */
      .
      .
      .
```

Modification of Loop Control Variables

Where possible, the expression-simplification process will modify both the control variable and the iteration specification of a do-loop to achieve more efficient processing when the control variable is used as a subscript. The calculation of addresses of array elements can be made faster by replacing multiplication operations by addition operations. For example, the loop:

```
DO I = 1 TO N BY 1;
A(I) = B(I);
END;
```

causes N element values from array B to be assigned to corresponding elements in array A. On the assumption that each element is four bytes in length, the address calculations used for each iteration of the loop are:

```
Base(A)+(4*I)    for array A, and

Base(B)+(4*I)    for array B,
```

where "Base" represents the base address of the array in storage. The repeated multiplication of the control variable by a constant representing the length of an element can be converted to faster addition operations. The optimized DO statement above is converted into object code equivalent to the following statement:

```
DO TEMP = 4 BY 4 TO 4*N;
```

The element address calculations are converted to the equivalent of:

```
Base(A) + TEMP    for array A, and

Base(B) + TEMP    for array B.
```

A loop control variable and its iteration specification can be optimized only when the control variable used as a subscript is incremented by a constant value. The value of the control variable must not be required outside the loop in which it is specified.

Defactorization

Where possible, a constant in an array subscript expression is used as an offset in the address calculation. For example, the address of a four-byte element:

```
A(I+10)
```

would be calculated as:

```
(Base(A)+4*10)+I*4.
```

Replacement of Constant Expressions

The expression-simplification process replaces constant expressions of the form A+B or A*B, where A and B are integer constants, with the equivalent constant. For example the expression 2+5 is replaced by 7.

Replacement of Constant Multipliers and Exponents

The expression-simplification process replaces certain constant multipliers and exponents. For example:

```
A*2 becomes A+A,
```

and

```
A**2 becomes A*A.
```

Initialization of Arrays and Structures

When arrays and some structures which have the BASED, AUTOMATIC, or CONTROLLED storage class are to be initialized by a constant specified in the INITIAL attribute, the first element of the variable is initialized by the constant, and the remainder of the initialization consists of

a single move which propagates the value through all the elements of the variable. For example:

```
DCL A(20,20) FIXED BINARY INIT((400)0);
```

The array A would be initialized in this way.

In-Line Code for Conversions

Most conversions are performed by in-line code, rather than by calls to the resident or transient library. The exceptions are:

- Conversions between character and arithmetic data.
- Conversions from numeric character (PICTURE) data where the picture includes characters other than 9, V, Z, or a single sign or currency character.
- Conversions to numeric character (PICTURE) data where the picture includes scale factors or floating point picture characters.

For example, conversions to 'ZZ9V99' will be done in-line.

In-Line Code for Record I/O

For consecutive buffered files under certain conditions, the input/output transmission statements READ, WRITE, and LOCATE are implemented by in-line code rather than by calls to the resident or transient libraries.

Key Handling for REGIONAL Data Sets

In certain circumstances, key handling for regional data sets is simplified by avoiding unnecessary conversions between fixed binary and character-string data types, as follows:

REGIONAL(1): If the key is supplied as a fixed binary integer with precision (12,0) through (23,0), there is no conversion from fixed binary to character-string and back again.

REGIONAL(2) and REGIONAL(3): If the key is supplied in the form K||I, where K is a character string and I is fixed binary with precision (12,0) through (23,0), the rightmost eight characters of the resultant string are not reconverted to fixed binary. (This conversion would otherwise be necessary to obtain the region number.)

Matching Format Lists with Data Lists

Where possible, format and data lists of edit-directed input/output statements are matched at compile-time. This is possible only when neither list contains repetition factors that are expressions whose values are unknown at compile-time. This permits conversion to or from the data list item to be performed by in-line code. Also, on input the item can be taken directly from the buffer or on output placed directly in the buffer. Library calls are eliminated except when it is necessary to transmit a block of data between the input/output device and the buffer, for example:

```
DCL (A,B,X,Y,Z) CHAR(25);
GET FILE(SYSIN) EDIT(X,Y,Z) (A(25));
PUT FILE(SYSPRINT) EDIT(A,B) (A(25));
```

In this example, format list matching is performed at compile time; at execution time, library calls will be required only when the buffer contents are to be transmitted to or from the input/outout device.

In-Line Code for String Manipulation

Operations on many character strings (such as concatenation and assignment of adjustable, varying length, and fixed-length strings) are performed in-line. Similar operations on many aligned bit strings are performed by in-line code.

In-Line Code for Built-In Functions

Many built-in functions are executed by in-line code. INDEX and SUBSTR are examples of functions for which in-line code is usually generated. TRANSLATE, VERIFY, and REPEAT are examples where in-line code is generated for simple cases.

Special-Case Code for DO Statements

Where possible, for a do-loop, the Optimizing Compiler will generate code in which the value of the control variable and the values used in the iteration

specification are held in registers throughout execution of the loop. For example, the compiler will attempt to maintain in registers the values of the variables I, K, and L in the following statement:

```
DO I = A TO K BY L;
```

This form of optimization permits the most efficient loop control instructions to be used.

Structure and Array Assignments

Structure and array assignment statements are implemented by single move instructions whenever possible. Otherwise the assignment is performed by the simplest loop possible for the operands specified in the assignment. For example:

```
DCL A(10),B(10), 1 S(10), 2 T, 2 U;

    A=B;

    A=T;
```

The first assignment statement will be implemented by a single move instruction, while the second will be implemented by a loop, since array T is interleaved with array U, thereby making a single move impossible.

Library Subroutines

The resident and transient libraries have been designed as sets of subroutines containing logically-related functions. These functions are distributed among the library subroutines in such a manner that a link-edited object program will contain only code likely to be necessary for the functions used in that program.

The groups of functions particularly concerned with this efficient structuring include record-oriented input/output, stream-oriented input/output, conversions, and error handling.

Elimination of Common Constants

If a constant is used more than once in a program, a single copy of that constant is kept. For example, in the following statements:

```
WEEK_NO = WEEK_NO + 1;
RECORD_COUNT = RECORD_COUNT + 1;
```

the 1 is stored once only provided that WEEK_NO and RECORD_COUNT have the same attributes.

Elimination of Common Control Data

The Optimizing Compiler generates control information to describe certain program elements such as arrays. If there are two or more similar arrays, this descriptive information is generated once only.

Use of Registers

More efficient execution of loops can be achieved by maintaining in registers the values of variables that are subject to frequent modification during execution of the loops. When program flow of control permits, values can be kept in registers, and considerable efficiency can be achieved by dispensing with time-consuming load-and-store operations to reset the values of variables in their storage locations. If the latest value of a variable is known not to be required after a loop has been executed, the value is not assigned to the storage location of the variable when control passes out of the loop.

Register allocation can be more significantly optimized if REORDER is specified for the block. However, the values of variables that are reset in the block are not guaranteed to be the latest assigned values when a computational interrupt occurs, since the latest value of a variable may be present in a register but not in the storage location of the variable. If ORDER is specified, optimization of register allocation is impeded by the requirement that all values of variables reset in the block are guaranteed, and must therefore be assigned immediately to the storage locations of their respective variables.

Code for Program Branches

Base registers for branch instructions in the object program are allocated in accordance with the logical structure of the program. This minimizes the occurrence of program-addressing load instructions in the middle of deeply nested loops.

Also, the branch instructions generated for IF statements are arranged by the compiler to be as efficient as possible. For example, a statement such as:

```
IF condition THEN GOTO label;
```

is defined by the PL/I language as being a test of the condition followed by a branch on false to the statement following the THEN clause. However, when the THEN clause consists only of a GOTO statement, the statement is compiled as a branch on true to the label specified in the THEN clause.

Chapter 3: The PL/I Language Implemented

New language features, together with a number of extensions to the language implemented by version 5 of the PL/I (F) Compiler, have been included in the language implemented by the optimizing compiler. A list of the PL/I keywords implemented by the new compiler is given in appendix A.

In general, a PL/I source program written for version 5 of the PL/I (F) Compiler will produce identical results when compiled by the Optimizing Compiler. In a few cases, results may differ because the Optimizing Compiler restricts some language features to a different degree, and because there are minor incompatibilities between the implementations of the language by the two compilers. The most significant differences in implementation restrictions are given in figure 1, and the incompatibilities are listed in appendix B.

Language Features

The language implemented by the optimizing compiler includes the following features:

Data Types

- Character-string and bit-string data.
- Fixed-point binary and fixed-point decimal data (real and complex).
- Floating-point binary and floating-point decimal data (real and complex).
- Character and (real and complex) decimal picture data.
- Label and entry data.
- File data.
- Task and event data.
- Pointer and offset data.
- Area data.

Operations

- Assignment, with automatic conversion if necessary, between data variables.
- Element, array, and structure expressions.
- Arithmetic, comparison, logical (boolean), and string-manipulation operators.
- Built-in functions for mathematical and arithmetic computation, string manipulation, manipulation of based and controlled storage, manipulation of arrays, multitasking, and error handling.
- Pseudovariables for computation and error handling.
- Facilities for creating programmer-defined functions.

Data Aggregates

- Arrays of data elements
- Structures of data elements
- Arrays of structures

Program Management

- Preprocessing.
- Separate compilation of external procedures of the same program.
- Structuring of program into blocks to limit the scope of names and to permit flexible allocation of storage.
- Recursive invocations of a procedure with stacking and unstacking of generations of automatic data.
- Multitasking facilities.
- Maintenance of external procedures in auxiliary storage for dynamic loading into, and deletion from main storage during program execution.

Feature	PL/I (F) Compiler (Version 5)	PL/I Optimizing Compiler
AREA size	32,768	16,777,216
Arrays, max.number of dimensions	32	15
Structures	63 levels	15 levels
Floating-point data	short, long precision decimal, binary and numeric character modes	short, long, or extended[1] precision decimal, binary, and numeric character modes
Factored attributes: DECLARE statement	No limit	max. number: 16
Attribute factorization: nesting levels	max.number: 73	max.number: 16
Block nesting	max.depth: 50	max.depth: 50[2]
Storage limitations: STATIC INTERNAL and AUTOMATIC	maximum: 16M bytes	No limit
String repetition factor maximum number of characters:	5	15
maximum value of factor:	16,370	32,767
Depth of nesting of descriptor list in ENTRY statement	max.number: 49	max.number: 2
Identifiers in check list	max.number: 510	limited only by size of statement
Elements in data list	max.number: 320	max.number: 250
Length of string in REPLY option of DISPLAY statement	max.length: 100	max.length: 72

[1] Facilities will be provided for the simulation of the extended floating-point instruction set; the user can employ these simulation facilities should the system at his installation not contain the instruction set.
[2] There is a limit on the depth of block nesting imposed by the length of the block labels given by the user. If the average length of the labels exceeds eight characters, the maximum number of available nesting levels is reduced.

Figure 1. Differences in implementation restrictions

- Dynamic storage allocation.
- Processing of interrupts and error conditions.

Input/Output

- Stream-oriented input/output with automatic conversion to and from internal and external forms of data representation.
- Record-oriented input/output with both move and locate modes of operation.
- Sequential and direct-access processing modes.
- Message processing (teleprocessing) mode.

- Asynchronous input/output data transmission.

Language Extensions

Language features implemented by the Optimizing Compiler that are only partially implemented or not implemented at all by version 5 of the PL/I (F) Compiler are described in outline below.

Optimization

The Optimizing Compiler carries out extensive optimization of PL/I programs. Optimization is controlled by a compiler option and the use within the PL/I program of the ORDER and REORDER options for program blocks. Optimization is discussed in chapter 2.

Program Checkout

CHECK Condition Prefix: Variables specified in the name-list of the CHECK condition may be of any problem data type. They must not be iSUB defined variables or explicitly-qualified based variables. They may also be parameters. For example:

```
      (CHECK (A,B,C,D)):
   F: PROC OPTIONS(MAIN);
      DCL A FIXED STATIC,
          B FLOAT AUTO,
          C CHAR(10) CONTROLLED,
          D FIXED DECIMAL BASED(P);

      CALL X(D);

      (CHECK(E)):
   X: PROC(E);
      DCL E FIXED DECIMAL;
```

iSUB Beyond Range: The SUBSCRIPTRANGE condition is raised when an iSUB variable is outside the range given in the declaration of the iSUB-defined array. For example:

```
      DCL A(24) CHAR(4),
          B(5) CHAR(4) DEF A(4*1SUB);
```

SUBSCRIPTRANGE would be raised on reference to B(6).

DEFAULT Statement

Identifiers which are declared with only a partial set of attributes, or which are not explicitly declared, can derive any other attributes required from two sources:

The standard default attributes supplied by PL/I.

The default attributes specified in a DEFAULT statement.

A DEFAULT statement can supplement, but not override, the attributes of selected groups of:

Explicitly, contextually, or implicitly declared identifiers.

Parameter descriptors in ENTRY descriptor lists.

Values returned by functions.

If there is no DEFAULT statement, or if a set of required attributes is still incomplete after the DEFAULT statement has been applied, any attributes required are supplied by the standard defaults.

The DEFAULT statement cannot be used to specify a structure, although the attributes of structure elements can have defaults applied according to a RANGE specification.

External entry names cannot be specified in a DEFAULT statement; they must be declared explicitly.

Note: A DEFAULT statement may be given a label, but the label will be treated as if it were on a null statement.

Preprocessing Facilities

Assignment to Values: The RESCAN and NORESCAN options may be specified on %ACTIVATE and %DEACTIVATE statements. The NORESCAN option enables the programmer to specify that an activated preprocessor character-string variable or entry name appearing in the source text is to be replaced by its actual value; that is, the usual rescanning process does not take place. If the RESCAN option is specified, the variable is activated as though no option were specified; rescanning takes place at each replacement level. For example:

```
%DCL WRITE CHARACTER;
```

```
%WRITE = 'WRITE EVENT(E)';

%ACTIVATE WRITE NORESCAN;

WRITE FILE(F) FROM(X);
```

The WRITE statement will be modified to:

```
WRITE EVENT(E) FILE(F) FROM(X);
```

If the %ACTIVATE statement were omitted, replacement of WRITE would continue indefinitely, and the program would be meaningless.

LENGTH and INDEX: The LENGTH and INDEX built-in functions may be used within a preprocessor statement.

Listing Control Statements

The listing control statements, %SKIP and %PAGE, allow the programmer to control the formatting and line spacing of the printed listing of his program. Although both these statements have the initial % sign, neither of them necessitates the use of the preprocessor.

%PAGE: The statement following the %PAGE statement in the program listing is printed on the first line of the next page. The statement is applied to both the insource listing (the input to the preprocessor) and the source listing (the preprocessed text).

%SKIP[(n)]: The specified number of lines following a %SKIP statement in the program listing are left blank. "n" may be any integer from 1 through 99. Omission of a value for "n" is equivalent to specifying the value of 1. The statement is applied to both the insource listing (the input to the preprocessor) and the source listing (the preprocessed text).

Storage Control

Subscripted or Based Locator: The pointer and offset variables associated with based variables can be subscripted, or based, or both.

Self-defining structures: Many of the restrictions imposed by the PL/I (F) implementation of the REFER option have been removed or modified in the language implemented by the Optimizing Compiler. The main changes are:

- The REFER option can be used more than once in a based structure.
- If used only once in such a structure, it need not be used with the last element.
- The REFER object need not be fixed binary.

For example:

```
DCL 1 X BASED,
      2 M,
      2 N,
      2 Y(10 REFER (M):
        S REFER (N)),
      2 (I,J),
      2 K,
        3 A AREA(S REFER (I)),
        3 STR CHAR(L REFER (J)) VAR,
      S FLOAT BINARY INIT(2000),
      L FIXED DECIMAL INIT(50);
```

Loading procedures dynamically: A procedure invoked by a CALL statement, by a CALL option of an INITIAL attribute, or by a function reference, is normally resident in main storage throughout the execution of the entire program. External procedures (other than main procedures) may, however, be maintained in auxiliary storage. Copies of such procedures can be dynamically loaded into, and deleted from, main storage as required by the program. Fetching and releasing of the procedures are initiated by the new PL/I statements, FETCH and RELEASE. The new feature is particularly useful when a called procedure is not necessarily invoked every time the calling procedure is executed, and when conservation of main storage is important.

ENVIRONMENT Attribute

Changes and extensions to the options of the ENVIRONMENT attribute specifying record format, block size, and record size, are as follows:

```
Record format:   [F|FB|FBS|U|V|VB|VBS|VS|
                  D|DB]

Block size:      [BLKSIZE(blocksize)]

Record size:     [RECSIZE(recordsize)]
```

The ENVIRONMENT attribute need not necessarily specify any of the above options. Missing information may be supplied in a data-set label, a DD statement, or by default (at OPEN time). This means, for instance, that the programmer can specify record size in his source program while leaving specification of block size until the program is actually executed, when the I/O devices are known.

The Optimizing Compiler will recognize and convert the previously-implemented forms of the above options as shown below, and will issue a message stating that they are obsolete.

Old form	Converted to
F(b)	F BLKSIZE(b) RECSIZE(b)
F(b,r)	F BLKSIZE(b) RECSIZE(r)
U(b)	U BLKSIZE(b) RECSIZE(b)
V(b)	V BLKSIZE(b) RECSIZE(b-4)
V(b,r)	V BLKSIZE(b) RECSIZE(r)
VBS(b,r)	VBS BLKSIZE(b) RECSIZE(r)
VS(b,r)	VS BLKSIZE(b) RECSIZE(r)

There are two new data-set organizations, TP(M) and TP(R), associated with teleprocessing. TP(M) implies the transmission of whole messages; TP(R) implies the transmission of records. Both are valid only for TRANSIENT files. These data-set organizations are equivalent to the options G(m) and R(r) available in version 5 of the PL/I (F) Compiler. The Optimizing Compiler will recognize and convert as follows:

Old form	Converted to
G(m)	V TP(M) RECSIZE(m)
R(r)	V TP(R) RECSIZE(r)

Variables in Options: Whenever a numeric value is required to complete the specification of an ENVIRONMENT option, the value may be expressed as a decimal integer constant or a static fixed binary variable of precision (31,0). The variable, if used, must be assigned a value before the file is opened.

REREAD Option: The REREAD option backspaces a magnetic tape to the start of the data set when the associated file is closed.

ASCII Option: The ASCII option specifies that American standard code for information interchange is used to represent data on the data set.

BUFOFF Option: The BUFOFF option specifies the length of the block prefix fields that may be associated with records of an ASCII data set.

SCALARVARYING Option: When storage is allocated for a varying-length string variable, the Optimizing Compiler adds two bytes, containing the current length of the string, to the declared length of the variable. The SCALARVARYING option determines how varying-length strings are transmitted to and from a data set.

The option must be specified when locate mode statements (LOCATE and READ SET) are used to create or access a data set with records of this type. During creation, the length field is transmitted with the string; when being accessed, the first two bytes of the record are recognized as a length field.

For a non-element varying-length string, for example, one that is an array or a member of a structure that is transmitted as a whole, the length field will always be transmitted both while being created and while being accessed.

The option must not be specified for use with varying-length string records in data sets created by version 5 of the PL/I (F) Compiler, as this compiler neither creates nor recognises such a length field.

The SCALARVARYING and CTLASA/CTL360 options cannot be specified for the same data set.

Record-Oriented Transmission

Record I/O Statements: The record variable specified in an INTO or FROM option can be DEFINED, or a parameter, provided the reference is to connected storage. The CONNECTED attribute is discussed later in this chapter. The reference may be a structure or array which contains varying-length strings. For example:

```
DCL A (10) CHAR (6) VAR,
    1 B,
      2 C CHAR (3) VAR,
      2 D CHAR (16) VAR,
    E(10) CHAR(6) DEFINED A;

READ FILE (X) INTO (A);
READ FILE (X) INTO (B);
WRITE FILE (X) FROM (E);

CALL Z(A);

Z:  PROC (Y);

   DCL Y (10) CHAR (6) VAR;
   READ FILE (X) INTO (Y);
```

ASCII Data Sets

Data sets on magnetic tape whose data is represented in American standard code for information interchange (ASCII) may be created and accessed by a PL/I program. The implementation supports F, FB, U, D, and DB record formats. F, FB, and U

formats are treated in the same way as data sets whose data is represented in EBCDIC (extended binary coded decimal interchange code). The D and DB formats are equivalent to V and VB records. With the exception of the ENVIRONMENT attribute the same PL/I program may be used to handle an ASCII data set as would be used for a data set in EBCDIC form. (See, "ENVIRONMENT Attribute" earlier in this chapter.)

Only character data may be written onto an ASCII data set. Varying-length strings cannot be transmitted to a data set using a SCALARVARYING file. Since an ASCII data set must be on magnetic tape it must be of CONSECUTIVE organization.

File Names

File-Name Expressions: File names can be specified, in input/output statements, as expressions:

For example:

```
DCL B(10) FILE,
    X(Q) FILE;
    .
    .
    .
GET FILE (FILEA)....;
PUT FILE (B(6))....;
READ FILE (X(N*3))....;
```

File-Name Constants and Variables: File names can be declared as file constants or

file variables. An identifier is assumed to be a file constant if:

1. It is declared with any file attribute other than FILE.

2. It is not explicitly declared but appears in the FILE option of an input/output statement, or in an ON statement for an input/output condition.

It is assumed to be a file variable if:

1. It has the FILE attribute and is an element of a structure.

2. It has the FILE attribute and any of the following additional attributes:

   STATIC|AUTOMATIC|BASED|CONTROLLED
   Dimension
   Parameter
   ALIGNED|UNALIGNED
   DEFINED
   INITIAL
   VARIABLE

Data Aggregates

Array and Structure Operations: A reference can be made to both an array and a structure in the same expression or assignment provided that the target is an array of structures.

String Handling

STRINGSIZE Condition: If a string is assigned to a string shorter than itself, the source string is truncated on the right to the length of the target string, and the STRINGSIZE condition is raised. For example:

```
(STRINGSIZE): P: PROC;
  DCL A CHAR (20),
      B CHAR (14);
  B = A;  /* STRINGSIZE IS RAISED, A IS
           TRUNCATED ON ASSIGNMENT TO B */
```

BASED or DEFINED with a VARYING Attribute: A variable declared with the BASED or DEFINED attribute may also be given the VARYING attribute. For example:

```
DCL C CHAR(10) BASED(P) VARYING,
    B CHAR(12) VARYING,
    D CHAR(12) VARYING DEFINED B;
```

Data Attributes

The CONNECTED Attribute: This is a storage attribute of non-controlled data aggregate parameters. It specifies that the storage associated with the aggregate is contiguous, that is, not interleaved with storage for other variables. This allows the aggregate to be designated as a record variable or a base in string overlay defining. For example:

```
Q:  PROC (X,Y);
    DCL A FILE RECORD ENVIRONMENT
    (F BLKSIZE(40)RECSIZE(40)),
    X (10) CHAR (4) CONNECTED,
    1 Y CONNECTED UNALIGNED,
      2 R,
      2 S,
      3 T,
      3 U,
    G CHAR (40) DEF X;
      .
      .
    READ FILE (A) INTO (X);
      .
      .
    SUBSTR (G,28,8)=....;
      .
```

```
      .
    WRITE FILE (A) FROM (X);
      .
      .
    CALL B (Y);
      .
      .
B:  PROC (Z) RETURNS (DECIMAL);
    DCL 1 Z CONNECTED UNALIGNED,
       2 RR,
       2 SS,
       3 TT,
       3 UU;
      .
      .
      END Q;
```

Initialization by Expression or Function: The initial value of a non-STATIC variable declared with the INITIAL attribute can be derived from sources other than a constant. The source may be an expression or a function reference; on evaluation, the value derived is the initial value of the variable. For example:

```
DCL B AUTOMATIC INITIAL(F(X)),
    C INITIAL(SQRT(3));
```

Variables and function references used in these expressions must be known in the block in which the initialized item is declared.

Extended Precision Floating Point

Where the extended precision feature is available, the maximum precisions are:

Binary floating-point data	109
Decimal floating-point data	33

The default precision for floating-point data remains as for version 5 of the PL/I (F) Compiler, that is binary 21, decimal 6.

Subroutines and Functions

There is greater flexibility in the use of entry names. An entry name can be either an entry constant or an entry variable.

Entry Constant: An identifier specified as a label prefix to a PROCEDURE or ENTRY statement is an entry constant. For example:

```
X:  PROC;

Y:  ENTRY;
```

X and Y are entry constants. They can be explicitly declared with the ENTRY attribute only if external.

Entry Variable: An identifier specified with the ENTRY attribute and any of the following attributes, which must be explicitly declared, is an entry variable.

```
VARIABLE
STATIC|AUTOMATIC|BASED|CONTROLLED
Dimension
Parameter
ALIGNED|UNALIGNED
DEFINED
INITIAL
```

For example:

```
DCL X ENTRY (FIXED,BINARY,DECIMAL)
        VARIABLE,
    Y ENTRY (CHAR) AUTO,
    Z (10) ENTRY;
```

Entry variables may be assigned, passed as arguments, and used in CALL statements and function references.

Entry Names and Generic Names: The GENERIC attribute has been altered to permit the entry names specified in a generic declaration to be dissociated from the generic name. The specification of an identifier as an alternative in a generic declaration does not constitute a declaration of the identifier; the occurrence of the identifier is regarded simply as an entry expression that describes the entry name of the corresponding procedure. For each entry expression alternative, a descriptor list is specified by means of a WHEN clause. Defaults are not applied to the descriptors in WHEN clauses. As a result the generic specification is much more flexible in that:

1. An identifier can be specified as more than one alternative in a GENERIC declaration, or in more than one GENERIC declaration.

2. The descriptor list need only partially describe the argument that must be matched in the generic reference. It can indicate that any argument is acceptable.

For example:

```
DCL X GENERIC (A WHEN (FIXED, FIXED));
```

If the arguments to the generic reference X are both FIXED, then A is selected. If the specification had been

```
A WHEN (,FIXED)
```

then any type of argument would be acceptable as the first argument, but the second would have to be FIXED.

Generic selection is performed thus:

1. The alternatives are scanned from left to right until a WHEN clause is found which matches the arguments in the generic reference. "Matching" occurs when there are the same number of descriptors in the WHEN clause as there are arguments, and when each descriptor in the WHEN clause specifies a subset of the attributes of the corresponding argument.

2. The entry expression in the alternative that contains the appropriate WHEN clause is then invoked with the arguments in the generic reference. The invocation may involve the conversion of arguments and the creation of dummy arguments.

For example:

```
  DCL  X GENERIC (A WHEN (FIXED,FLOAT),
                  A WHEN (FIXED,FIXED),
                  B WHEN (FLOAT,COMPLEX)),
       Y GENERIC (A WHEN (FLOAT,FLOAT),
                  A WHEN (,FLOAT),
                  C WHEN (,)),
       Z GENERIC (A WHEN (FIXED,FIXED),
                  B WHEN (FLOAT,COMPLEX)),
       (L1,L2) FIXED,
       (M1,M2) FLOAT,
       S COMPLEX;
                  .
                  .
CALL X (L1,M1); /* A IS SELECTED */
CALL X (L1,L2); /* A IS SELECTED */
CALL X (M2,S);  /* B IS SELECTED */
CALL X (M1,M2); /* NO MATCHING ARGUMENTS,
                   NO SELECTION.
                   PROGRAM IS ERRONEOUS*/
CALL Y (M1,M2); /* A IS SELECTED */
CALL Y (L1,M1); /* A IS SELECTED */
CALL Y (S,L1);  /* C IS SELECTED */
M1 = Z (L1,L2); /* A IS SELECTED */
M2 = Z (L1,M2); /* NO MATCHING ARGUMENT,
                   NO SELECTION.
                   PROGRAM IS ERRONEOUS*/
M1 = Z (M2,S); /* B IS SELECTED */
```

Mathematical Built-In Functions

New mathematical built-in functions, ASIN(x) and ACOS(x), are available for computing arc sines and arc cosines. Only real arguments are allowed.

Comparison Operators

The = and ^= operators may be used to compare any data type other than area, condition, event, or task type.

Interlanguage Communication

The OPTIONS Attribute/Option: The interlanguage communication facilities are requested in a PL/I procedure by specifying the COBOL or FORTRAN option in the PROCEDURE or ENTRY statement or in the OPTIONS attribute in a DECLARE statement. OPTIONS(ASSEMBLER) can be specified in a DECLARE statement.

The remapping of COBOL structures, and the transposing of FORTRAN arrays, can be completely or partially suppressed by the NOMAP, NOMAPIN, or NOMAPOUT options.

The INTER option is used to specify that the assembler-language, COBOL and FORTRAN interrupts that would normally be handled by the operating system are to be handled by the PL/I interrupt handling facilities.

Chapter 4: System Requirements

Machine Requirements

COMPILATION

The minimum machine requirements for the OS PL/I Optimizing Compiler are supplied by an IBM System/360 Model 40 with a main storage capacity of 128K bytes. The compiler will operate in a 50K partition under MFT, in a 52K region under MVT, or in a 72K region with the Time Sharing Option (TSO). Under VS1 and VS2, the partition or region sizes are rounded up to a multiple of 64K. The compiler will also operate under the control of the Conversational Monitor System (CMS) of the Virtual Machine Facility/370 (VM/370) and needs 320K bytes of virtual storage for the CMS virtual machine.

The central processing unit of the machine must have the decimal and floating-point instruction sets. If timing information is required, the central processing unit must have the timer feature, and use of this feature must be specified at system generation.

EXECUTION

The storage requirements, either real or virtual, for the execution of a PL/I program compiled by the optimizing compiler depend on the size and content of the source program. The real storage requirements may be reduced either by the use of the VS1 or VS2 control programs or by use of the overlay facilities for handling segmented object programs.

The machine must have the decimal instruction set. Where the extended precision floating point feature is available it must have also the floating-point instruction set. If timing information is required, the machine must have the timer feature, and use of this feature must be specified at system generation. (If the DELAY statement is used, the ability to handle the interval timer from the application program must also be specified at system generation.) If the DATE built-in function is used, the OS control program should have date facilities incorporated at system generation.

Full details of system generation are given in the publication System Generation, Order No. GC28-6554.

AUXILIARY STORAGE

Direct-access storage space is required in the link library for the optimizing compiler and its libraries as follows:

- Compiler: approximately 430 tracks on a 2311, 240 tracks on a 2314, or 130 tracks on a 3330.
- Resident library: approximately 60 tracks on a 2311, 33 tracks on a 2314, or 20 tracks on a 3330.
- Transient library: approximately 60 tracks on a 2311, 31 tracks on a 2314, or 18 tracks on a 3330.

The Optimizing Compiler requires direct-access storage space for working-storage only when adequate main storage is not available. The amount required depends upon the size of the source program and the amount of main storage available to the compiler. The data set identified by SYSUT1 is used for this auxiliary storage. This data set is also used when the preprocessor is invoked and when the PL/I 48-character set is used in the source program.

INPUT/OUTPUT DEVICES

At compile time and during subsequent link-editing, devices are required for the following types of input/output:

Source program input

Printed listings

Object module in relocatable format

Object module in relocatable card-image format

The ddname associated with a particular class of compiler input/output, and the permitted device types for each, are shown in figure 2.

Function	ddname	Device Type	When Required
Input	SYSIN	DASD Magnetic tape Card reader Paper-tape reader	Always
Print	SYSPRINT	DASD Magnetic tape Printer	Always
Output to Linkage Editor	SYSLIN	DASD Magnetic tape	When link-editing follows compilation in the same job.
Output to Linkage Editor (card deck)	SYSPUNCH	DASD Magnetic tape Card Punch	When link-editing takes place in a subsequent job.
Compiler Spill File	SYSUT1	DASD	When adequate main storage is not available
Source Statement Library	SYSLIB	DASD	When preprocessor %INCLUDE is used with either MACRO or INCLUDE option

Figure 2. Compiler input/output devices

Operating System Requirements

The Optimizing Compiler is a component of the IBM Operating System. The control programs that the compiler will run under, and the data management facilities that may be required, are detailed below.

CONTROL PROGRAMS

The Optimizing Compiler can be used under the following control programs:

Multiprogramming with a Fixed number of Tasks (MFT): The number of tasks that can be processed at any one time is determined by the number of partitions (segments of main storage) that exist at that time. Each task is associated with one partition, and receives a share of the available resources.

Virtual Storage (VS1): This control program is the virtual-storage equivalent of the MFT control program described above. In general, a source program that compiles and executes successfully under MFT will do the same under VS1.

The PL/I multitasking facilities should not be used under an MFT or VS1 control program.

Multiprogramming with a Variable number of Tasks (MVT): The number of tasks that can be processed at any one time is determined by the number of regions (segments of main storage) plus the number of tasks created dynamically in all the regions. Each region is associated with a task and with all the subtasks it creates dynamically. Each task receives a share of the resources; subtasks use the resources allocated to the task that created them.

Virtual Storage (VS2): This control program is the virtual-storage equivalent of the MVT control program described above. In general, a source program that compiles and executes successfully under MVT will do the same under VS2.

Other operating system options under which the compiler can run are:

Time Sharing Option (TSO): This option is available only to MVT or VS2 users and requires a region approximately 20K bytes larger than for MVT. The option provides:

- The ability to use subcommands that allow interaction with a PL/I program during processing.
- Facilities for several users to share a region of main storage for the concurrent execution of their programs. Each terminal in session is granted exclusive use of the region for a series

of short time slices.

Multiprocessing (M65MP): Provides support for multiprocessing with two Model 65s.

|Conversational Monitor System (CMS): see |appendix D for further details.

DATA MANAGEMENT FACILITIES

Object programs compiled by the optimizing compiler make use of the operating system data management facilities. These facilities are dependent on the I/O statements used and include:

Basic Sequential Access Method (BSAM)

Queued Sequential Access Method (QSAM)

Basic Indexed Sequential Access Method (BISAM)

Queued Indexed Sequential Access Method (QISAM)

Basic Direct Access Method (BDAM)

Telecommunication Access Method (TCAM)

| Virtual Storage Access Method (VSAM)

SORT FACILITIES

The PL/I program may contain statements to invoke the operating system Type 1 Sort/Merge program (360S-SM-023) or the program product Sort/Merge program (5734-SM1) and pass records to be sorted, or receive sorted records, or both pass and receive records.

CHECKPOINT/RESTART FACILITIES

The PL/I program may contain a statement to invoke the checkpoint facility to record the current status of a program and its data in auxiliary storage. The checkpoint data can be used by the restart facility to restart the program at the point in execution when the checkpoint was taken.

System/370 Facilities

|The following control programs, operating |systems, input/output devices, and access |method can be used only with System/370:
|
| VS1 (Virtual Storage) control program.
|
| VS2 (Virtual Storage) control program.
|
| Conversational Monitor System (CMS).
|
| The 3330 Direct-Access Storage Device.
|
| The 3505 Card Device.
|
| The 3525 Card Device.
|
| VSAM (Virtual Storage Access Method).
|
| The availability of the optimizing |compiler with these features is stated in |the publication IBM Program Product Design |Objectives, Order No. GC33-0036.

Appendix A: Keywords and their Abbreviations

The following is a complete list of the PL/I and implementation-defined keywords implemented, with exceptions, by both the OS PL/I Optimizing Compiler and the OS PL/I Checkout Compiler. The keywords not implemented by the Optimizing Compiler are indicated by an asterisk. Each of the keywords is described in detail in the publication OS PL/I Checkout and Optimizing Compilers: Language Reference Manual.

Keyword	Abbreviation	Use of Keyword
A[(w)]		format item
ABS(x)		built-in function
ACOS(x)		built-in function
%ACTIVATE	%ACT	preprocessor statement
ADD(x_1,x_2,x_3[,x_4])		built-in function
ADDBUFF		option of ENVIRONMENT attribute
ADDR(x)		built-in function
ALIGNED		attribute
ALL [(character-string-expression)] *		option of PUT statement
ALL(x)		built-in function
ALLOCATE	ALLOC	statement
ALLOCATION(x)	ALLOCN	built-in function
ANY(x)		built-in function
AREA		condition
AREA[(size)]		attribute
ARGn		option of NOMAP, NOMAPIN, NOMAPOUT options
ASCII		option of the ENVIRONMENT attribute
ASIN(x)		built-in function
ASSEMBLER	ASM	option of OPTIONS option/attribute
ATAN(x_1[,x_2])		built-in function
ATAND(x_1[,x_2])		built-in function
ATANH(x)		built-in function
ATTENTION *	ATTN	condition
AUTOMATIC	AUTO	attribute
B[(w)]		format item
BACKWARDS		attribute, option of OPEN statement
BASED[(locator-expression)]		attribute
BEGIN		statement
BINARY	BIN	attribute
BINARY(x_1[,x_2[,x_3]])	BIN(x_1[,x_2[,x_3]])	built-in function
BIT[(length)]		attribute
BIT(x_1[,x_2])		built-in function
BLKSIZE(block-size)		option of ENVIRONMENT attribute
BOOL(x_1,x_2,x_3)		built-in function
BUFFERED	BUF	attribute
BUFFERS(n)		option of ENVIRONMENT attribute
BUFOFF[(n)]		option of ENVIRONMENT attribute
BUILTIN		attribute
BY		option of DO statement, option of repetitive input/output specification
BY NAME		option of assignment statement
C(real-format-item [,real-format-item])		format item
CALL		statement, option of INITIAL attribute
CEIL(x)		built-in function
CHAR(x_1[,x_2])		built-in function
CHARACTER[(length)]	CHAR[(length)]	attribute
CHECK *		statement
CHECK[(name-list)]		condition, condition prefix
CLOSE		statement

Keyword	Abbreviation	Use of Keyword
COBOL		option of ENVIRONMENT attribute, or OPTIONS option/attribute
COLUMN(n)	COL(n)	format item
COMPLETION(x)	CPLN(x)	built-in function, pseudovariable
COMPLEX	CPLX	attribute
COMPLEX(x_1,x_2)	CPLX(x_1,x_2)	built-in function, pseudovariable
CONDITION	COND	attribute
CONDITION(name)	COND(name)	condition
CONJG(x)		built-in function
CONNECTED	CONN	attribute
CONSECUTIVE		option of ENVIRONMENT attribute
%CONTROL *		listing control statement
CONTROLLED	CTL	attribute
CONVERSION	CONV	condition, condition prefix
COPY[(file-expression)]		option of GET statement
COS(x)		built-in function
COSD(x)		built-in function
COSH(x)		built-in function
COUNT(file-expression)		built-in function
CTLASA		option of ENVIRONMENT attribute
CTL360		option of ENVIRONMENT attribute
D		option of ENVIRONMENT attribute
DATA		option of GET or PUT statement
DATAFIELD		built-in function
DATE		built-in function
DB		option of ENVIRONMENT attribute
%DEACTIVATE	%DEACT	preprocessor statement
DECIMAL	DEC	attribute
DECIMAL(x_1[,x_2[,x_3]])	DEC(x_1[,x_2[,x_3]])	built-in function
DECLARE	DCL	statement
%DECLARE	%DCL	preprocessor statement
DEFAULT	DFT	statement
DEFINED	DEF	attribute
DELAY		statement
DELETE		statement
DESCRIPTORS		option of DEFAULT statement
DIM(x_1,x_2)		built-in function
DIRECT		attribute
DISPLAY		statement
DIVIDE(x_1,x_2,x_3[,x_4])		built-in function
DO		statement, repetitive input/output data specification
%DO		preprocessor statement
E(w,d[,s])		format item
EDIT		option of GET or PUT statement
ELSE		clause of IF statement
%ELSE		clause of %IF statement
EMPTY		built-in function
END		statement
%END		preprocessor statement
ENDFILE(file-expression)		condition
ENDPAGE(file-expression)		condition
ENTRY		attribute, statement
ENVIRONMENT	ENV	attribute, option of CLOSE statement
ERF(x)		built-in function
ERFC(x)		built-in function
ERROR		condition
EVENT		attribute, option of CALL, DELETE, DISPLAY, READ, REWRITE, and WRITE statements
EXCLUSIVE	EXCL	attribute
EXIT		statement
EXP(x)		built-in function

Keyword	Abbreviation	Use of Keyword
EXTERNAL	EXT	attribute
F(w,[,d[,s]])		format item
F		option of ENVIRONMENT attribute
FB		option of ENVIRONMENT attribute
FBS		option of ENVIRONMENT attribute
FETCH		statement
FILE		attribute
FILE(file-expression)		option of CLOSE, DELETE, GET, LOCATE, OPEN, PUT, READ, REWRITE, UNLOCK, and WRITE statements
FINISH		condition
FIXED		attribute
FIXED(x_1[,x_2[,x_3]])		built-in function
FIXEDOVERFLOW	FOFL	condition, condition prefix
FLOAT		attribute
FLOAT(x_1[,x_2])		built-in function
FLOOR(x)		built-in function
FLOW[1]		statement, option of PUT statement
FORMAT		statement, option of %CONTROL statement
FORTRAN		option of OPTIONS option/attribute
FREE		statement
FROM(variable)		option of WRITE or REWRITE statements
FS		option of ENVIRONMENT attribute
GENERIC		attribute
GENKEY		option of ENVIRONMENT attribute
GET		statement
GO TO	GOTO	statement
%GO TO	%GOTO	preprocessor statement
HALT[1]		statement
HBOUND(x_1,x_2)		built-in function
HIGH(x)		built-in function
IF		statement
%IF		preprocessor statement
IGNORE(n)		option of READ statement
IMAG(x)		built-in function, pseudovariable
IN(element-area-variable)		option of ALLOCATE and FREE statements
%INCLUDE		preprocessor statement
INDEX(x_1,x_2)		built-in function
INDEXAREA [(index-area-size)]		option of ENVIRONMENT attribute
INDEXED		option of ENVIRONMENT attribute
INITIAL	INIT	attribute
INPUT		attribute, option of OPEN statement
INTER		option of OPTIONS option/attribute
INTERNAL	INT	attribute
INTO(variable)		option of READ statement
IRREDUCIBLE	IRRED	attribute
KEY(file-expression)		condition
KEY(x)		option of READ, DELETE, and REWRITE statements
KEYED		attribute, option of OPEN statement
KEYFROM(x)		option of WRITE statement
KEYLENGTH(n)		option of ENVIRONMENT attribute
KEYLOC(n)		option of ENVIRONMENT attribute
KEYTO(variable)		option of READ statement
LABEL		attribute
LBOUND(x_1,x_2)		built-in function
LEAVE		option of ENVIRONMENT attribute
LENGTH(x)		built-in function
LIKE		attribute
LINE(n)		format item, option of PUT statement
LINENO(x)		built-in function
LINESIZE(expression)		option of OPEN statement

Keyword	Abbreviation	Use of Keyword
LIST		option of GET or PUT statement
LOCATE		statement
LOG(x)		built-in function
LOG2(x)		built-in function
LOG10(x)		built-in function
LOW(x)		built-in function
MAIN		option of OPTIONS option
MAX(x_1,x_2...x_n)		built-in function
MIN(x_1,x_2...x_n)		built-in function
MOD(x_1,x_2)		built-in function
MULTIPLY(x_1,x_2,x_3[,x_4])		built-in function
NAME(file-expression)		condition
NCP(n)		option of ENVIRONMENT attribute
NOCHECK *		statement
NOCHECK[(name-list)]		condition prefix
NOCONVERSION	NOCONV	condition prefix
NOFIXEDOVERFLOW	NOFOFL	condition prefix
NOFLOW *		statement
NOFORMAT		option of %CONTROL statement
NOLOCK		option of READ statement
NOMAP[(arg-list)]		option of OPTIONS attribute
NOMAPIN[(arg-list)]		option of OPTIONS attribute
NOMAPOUT[(arg-list)]		option of OPTIONS attribute
NOOVERFLOW	NOOFL	condition prefix
NORESCAN		option of %ACTIVATE statement
NOSIZE		condition prefix
NOSTRINGRANGE	NOSTRG	condition prefix
NOSTRINGSIZE	NOSTRZ	condition prefix
NOSUBSCRIPTRANGE	NOSUBRG	condition prefix
NOUNDERFLOW	NOUFL	condition prefix
NOWRITE		option of ENVIRONMENT attribute
NOZERODIVIDE	NOZDIV	condition prefix
NULL		built-in function
OFFSET[(area-name)]		attribute
OFFSET(x_1,x_2)		built-in function
ON		statement
ONCHAR		built-in function, pseudovariable
ONCODE		built-in function
ONCOUNT		built-in function
ONFILE		built-in function
ONKEY		built-in function
ONLOC		built-in function
ONSOURCE		built-in function, pseudovariable
OPEN		statement
OPTIONS(list)		attribute, option of ENTRY and PROCEDURE statements
ORDER		option of BEGIN and PROCEDURE statements
OUTPUT		attribute, option of OPEN statement
OVERFLOW	OFL	condition, condition prefix
P 'picture specification'		format item
PAGE		format item, option of PUT statement
%PAGE		listing control statement
PAGESIZE(w)		option of OPEN statement
PASSWORD		option of ENVIRONMENT attribute
PENDING(file-expression)		condition
PICTURE	PIC	attribute
POINTER	PTR	attribute
POINTER(x_1,x_2)	PTR(x_1,x_2)	built-in function
POLY(x_1,x_2)		built-in function
POSITION (expression)	POS (expression)	attribute
PRECISION(x_1,x_2[,x_3])	PREC(x_1,x_2[,x_3])	built-in function
PRINT		attribute, option of OPEN statement
PRIORITY(x)		option of CALL statement
PRIORITY[(x)]		built-in function, pseudovariable

Keyword	Abbreviation	Use of Keyword
PROCEDURE	PROC	statement
%PROCEDURE	%PROC	preprocessor statement
PROD(x)		built-in function
PUT		statement
R(x)		format item
RANGE		option of DEFAULT statement
READ		statement
REAL		attribute
REAL(x)		built-in function, pseudovariable
RECORD		attribute, option of OPEN statement
RECORD(file-expression)		condition
RECSIZE(record-length)		option of ENVIRONMENT attribute
RECURSIVE		option of PROCEDURE statement
REDUCIBLE	RED	attribute
REENTRANT		option of OPTIONS option
REFER(element-variable)		option of BASED attribute
REGIONAL(1|2|3)		option of ENVIRONMENT attribute
RELEASE		statement
REORDER		option of BEGIN and PROCEDURE statements
REPEAT(x_1,x_2)		built-in function
REPLY(c)		option of DISPLAY statement
REREAD		option of ENVIRONMENT attribute
RESCAN		option of %ACTIVATE statement
RETURN		statement, preprocessor statement
RETURNS(attribute-list)		attribute, option of PROCEDURE statement
REVERT		statement
REWRITE		statement
ROUND(x_1,x_2)		built-in function
SCALARVARYING		option of ENVIRONMENT attribute
SEQUENTIAL	SEQL	attribute
SET(locator-variable)		option of ALLOCATE, LOCATE, and READ statements
SIGN(x)		built-in function
SIGNAL		statement
SIN(x)		built-in function
SIND(x)		built-in function
SINH(x)		built-in function
SIZE		condition
SKIP[(n)]		format item, option of GET and PUT statements
%SKIP		listing control statement
SNAP *		option of ON and PUT statements
SQRT(x)		built-in function
STATIC		attribute
STATUS(x)		built-in function, pseudovariable
STOP		statement
STREAM		attribute, option of OPEN statement
STRING(x)		built-in function, pseudovariable
STRING(string-name)		option of GET and PUT statements
STRINGRANGE	STRG	condition, condition prefix
STRINGSIZE	STRZ	condition, condition prefix
iSUB		dummy variable of DEFINED attribute
SUBSCRIPTRANGE	SUBRG	condition
SUBSTR(x_1,x_2[,x_3])		built-in function, pseudovariable
SUM(x)		built-in function
SYSIN		name of standard system input file
SYSPRINT		name of standard system output file
SYSTEM		option of ON or DECLARE statements
TAN(x)		built-in function
TAND(x)		built-in function
TANH(x)		built-in function
TASK		attribute, option of OPTIONS option
TASK[(task-name)]		option of CALL statement

Keyword	Abbreviation	Use of Keyword
THEN		clause of IF statement
%THEN		clause of %IF statement
TIME		built-in function
TITLE(element-expression)		option of OPEN statement
TO		option of DO statement, option of repetiti input/output specification
TOTAL		option of ENVIRONMENT attribute
TP(M|R)		option of ENVIRONMENT attribute
TRANSIENT		attribute
TRANSLATE(x_1,x_2[,x_3])		built-in function
TRANSMIT(file-expression)		condition
TRKOFL		option of ENVIRONMENT attribute
TRUNC(x)		built-in function
U		option of ENVIRONMENT attribute
UNALIGNED	UNAL	attribute
UNBUFFERED	UNBUF	attribute, option of OPEN statement
UNDEFINEDFILE (file-expression)	UNDF (file-expression)	condition
UNDERFLOW	UFL	condition, condition prefix
UNLOCK		statement
UNSPEC(x)		built-in function, pseudovariable
UPDATE		attribute, option of OPEN statement
V		option of ENVIRONMENT attribute
VALUE		option of DEFAULT statement
VARIABLE		attribute
VARYING	VAR	attribute
VB		option of ENVIRONMENT attribute
VBS		option of ENVIRONMENT attribute
VERIFY(x_1,x_2)		built-in function
VS		option of ENVIRONMENT attribute
|VSAM		option of ENVIRONMENT attribute
WAIT		statement
WHEN(generic-descriptor-list)		option in GENERIC declaration
WHILE		option of DO statement
WRITE		statement
X(w)		format item
ZERODIVIDE	ZDIV	condition, condition prefix

Appendix B: Compatibility with (F) Compiler

Some features of the PL/I Optimizing Compiler implementation are incompatible with version 5 of the PL/I (F) Compiler implementation. The most significant incompatibilities are listed below. Except where stated, the description given is of the Optimizing Compiler implementation. Programs which were written for the (F) compiler and which use any of these features should be reviewed before compiling them with the Optimizing Compiler to ensure that they will return the same results.

These, and other differences, are also given in the programmer's guide for this compiler. This general information manual also gives some of the implementation limitations and restrictions of this compiler. The language reference manual for this compiler gives full details of the implementation of each language feature.

Areas

The (F) compiler holds the length of an area in the first 16 bytes of the area. The optimizing compiler holds the length of the area in a descriptor outside the area.

Arrays and Structures

- The maximum number of dimensions in an array is 15.
- The maximum depth of a structure is 15.

Built-In Functions

- Built-in functions are recognized on the basis of context only, so that all programmer-defined external procedures must be declared explicitly. Built-in functions without arguments, such as TIME and DATE, must also be declared explicitly with the BUILTIN attribute, or contextually with a null argument list, for example: TIME( ).
- For a variable to be a valid argument to the ADDR built-in function it must be connected and its left extremity must not lie within bytes that contain data belonging to other variables.
- The ALLOCATION built-in function returns a fixed-binary value giving the number of generations of the argument that exist in the current task.
- The NULLO built-in function is not implemented in the Optimizing Compiler. The NULL built-in function can be used for offset variables as well as for pointer variables.
- The PROD built-in function accepts arguments that are arrays of either fixed-point or floating-point elements. The value returned has the same scale as the argument given, except for fractional fixed-point arguments for which the result is in floating-point.
- The SUBSTR built-in function returns a non-varying string.
- The SUM built-in function accepts arguments that are arrays of either fixed-point or floating-point elements. The value returned has the same scale as the argument given.

The DISPLAY Statement

- The maximum length of the reply is 72 characters.

Dumps from PL/I Programs

- The object-time dump facility of the (F) compiler, IHEDUMP, requires a DD statement with the ddname PL1DUMP. Its equivalent in the Optimizing Compiler, PLIDUMP, requires a DD statement with the ddname PLIDUMP. The Optimizing Compiler will attempt to use PL1DUMP if PLIDUMP is not available.

Entry Names, Parameters, and Returned Values

- In general, an entry name in parentheses causes a dummy variable to be created; for the function to be invoked, a null argument list is required. However, an

entry name argument in parentheses, or an entry name without arguments, will be invoked if passed to a procedure whose parameter descriptor for the corresponding argument specifies an attribute other than ENTRY.

- External entry names must always be explicitly declared.
- Area and string extents in the RETURNS attribute or option must be represented by a decimal integer constant.
- The maximum depth of nesting in a descriptor list in the ENTRY attribute is 2.
- An aggregate expression involving strings may not be passed as an argument unless there is a corresponding parameter descriptor in which all string lengths are specified as decimal integer constants.
- An internal entry constant cannot be declared in a DECLARE statement. REDUCIBLE and IRREDUCIBLE may be specified on PROCEDURE and ENTRY statements. A scalar cannot be passed to an array parameter of an internal entry constant if the parameter's bounds are specified by asterisks.

The ENVIRONMENT Attribute

- For changes in the ENVIRONMENT attribute see "The ENVIRONMENT Attribute" in chapter 3.

Error Correction

- The error correction logic differs from that used by the PL/I (F) compiler. Invalid programs that are compiled and corrected by the (F) compiler may not give the same results on the optimizing compiler.

INITIAL Attribute

- The limitations on the length of DECLARE statements impose some restrictions on the use of the INITIAL attribute. These restrictions are described under "Statement Length" later in this section.

LIKE Attribute

- The Optimizing Compiler does not permit a substructure to have the LIKE attribute when another substructure within the major structure is the object of a further LIKE attribute. For example:

```
DCL 1 A LIKE C,
    1 B,
      2 C,
        3 D,
        3 E,
      2 (F) LIKE X,
    1 X,
      2 Y,
      2 Z;
```

In this example, the structure A has the LIKE attribute which refers to substructure C in structure B. B also contains substructure (F) with the LIKE attribute.

Link-Editing

- Programs compiled by the optimizing compiler cannot be link-edited with object modules produced by the PL/I (F) compiler.

Operating System Facilities

- The operating system facilities for sorting, for checkpoint/restart, for generating a return code, and for obtaining a dump are all invoked by means of a CALL statement with the appropriate entry-point name; for example CALL PLISRTA. The entry-point names, which are listed below, have the BUILTIN attribute and need not be declared explicitly.

Facility	Entry-point Name
Sort	PLISRTx
Checkpoint/Restart	PLICKPT
	PLIREST
	PLICANC
Return Code	PLIRETC
Dump	PLIDUMP

The Optimizing Compiler does not recognize the entry names used by the PL/I (F) compiler, that is, IHESRTx, IHESARC, IHETSAC, IHEDUMx, IHECKPx, and IHERESx.

Pictures

- Sterling picture data is not implemented. Therefore the following picture characters are not allowed:

  G, H, M, P, 6, 7, 8.

Preprocessor

- Text replaced by preprocessor statements does not have blanks appended to either end of the replacement value.
- A parameter descriptor list is not allowed in the declaration of a preprocessor variable with the ENTRY attribute.
- The RETURNS attribute may not be specified in a preprocessor DECLARE statement.

Pseudovariables

- For a varying string, the first 16 bits of the value of the UNSPEC pseudovariable represent the current length of the string.
- The pseudovariables COMPLETION, COMPLEX, PRIORITY, and STRING are not allowed as the control variable of a DO statement.

Record I/O

- READ, REWRITE, and DELETE statements are invalid for REGIONAL DIRECT OUTPUT files.
- There is no default record format for RECORD files. If the record format is not specified, the UNDEFINEDFILE condition is raised.

Redundant Expression Elimination

- The process of eliminating redundant expressions could give rise to an incompatibility for programs written for the PL/I (F) compiler that are recompiled by the optimizing compiler. For example, in the expression:

  IF (A=D)|(C=D) THEN ...

  if the condition (A=D) is satisfied, the condition (C=D) is ignored. This latter condition, however, might contain a function, which if not evaluated, could give rise to error.

Return Codes

- The PL/I(F) Compiler return code facility (IHESARC or IHETSA) allowed values greater than 999 to be set. The optimizing compiler facility (PLIRETC) allows a maximum of 999 to be set. All values above this limit are reduced to 999 and a warning message issued.

REWIND Option

- The Optimizing Compiler does not implement the REWIND option. Programs that use this option should be modified. Note that the function of the REREAD option implemented by the Optimizing Compiler is different from that of the REWIND option. The main difference is that the REREAD option overrides any DISP parameter and controls the positioning of the magnetic tape volume, whereas the REWIND option specifies that the DISP parameter is to control the positioning of the magnetic tape volume according to the subparameter specified.

Statements

- The approximate maximum number of statements in a program is 10,000.

Statement Labels

- A label on a DECLARE statement is treated as if it were on a null statement.

Statement Lengths

- The optimizing compiler has a restriction that any statement must fit into the work area of the compiler. The maximum size of this work area varies with the amount of space available to the compiler. The limitations on the length of statements are:

Space Available | Max. Statement Lengths

50K - 60K	1012 characters
61K - 78K	1600 characters
Over 78K	3400 characters

Under VS1 or VS2 or when the 3330 is used for the spill file the limitations on the length of statements are:

78K - 83K	3400 characters
Over 83K	3960 characters

The DECLARE statement is an exception in that it can be regarded as a sequence of separate statements, each of which starts wherever a comma occurs that is not contained within parentheses. For example:

```
DCL 1 A
    2 B(10,10) INIT(1,2,3,...),
    2 C(10,100) INIT((1000) 0),
    2 (D,E) CHAR(20) VAR, ...
```

In this example, each line can be treated by the compiler as a separate DECLARE statement in order to accommodate it in the work area. The compiler will also treat in the same way the INITIAL attribute when it is followed by a list of items separated by commas that are not contained within parentheses. Each item may contain initial values that, when expanded, do not exceed the maximum length. The above also applies to the use of the INITIAL attribute in a DEFAULT statement.

The (F) compiler can use up to 90K bytes for its work area. It is possible that programs with large DECLARE statements will not compile successfully on the optimizing compiler although they had compiled successfully on the (F) compiler. The following techniques are suggested to overcome this problem:

1. Increase the main storage available to the compiler, unless the space available already gives the largest statement length.

2. Simplify the DECLARE statement so that the compiler can treat the statement in the manner described above.

Stream Transmission

- A filemark (or end of string mark for the STRING option) is a valid item delimiter for list- and data-directed input. An item thus delimited is processed intact, and ENDFILE (ERROR for string) is raised on attempting to read a further item from the file or string.

- After a GET LIST statement has been processed, a file is positioned to the next non-blank character or, if this is a comma, to the character following the comma. A GET EDIT statement following a GET LIST on the same file must take into account the position of the file.

- If the variable on the left-hand side of an item of data-directed input is based or controlled, and is not currently allocated, the ERROR condition is raised. The NAME condition is raised for all other errors (including out of range subscripts) detected on the left-hand side of data-directed input items.

- The following sequence:

```
DCL SYSPRINT FILE;
    .
    .
    .
OPEN FILE(SYSPRINT);
```

will cause the UNDEFINEDFILE condition to be raised. Omission of either or both statements will result in correct execution. The file should be declared implicitly or with the attributes STREAM and OUTPUT.

Varying-length strings

- The (F) compiler initializes a varying-length string to a null string, the equivalent of:

```
STR = '';
```

whenever such a string is allocated. The Optimizing Compiler does not perform any initialization of varying-length strings, unless the INITIAL attribute is used.

Appendix C: Bibliography

The following publications describe, in more detail, the facilities of the OS PL/I Optimizing Compiler, the resident library, and the transient library.

The availability of a publication is indicated by its use key, the first letter in the order number. The use keys are:

G General: available to users of IBM systems, products, and services without charge, in quantities to meet their normal requirements; can also be purchased by anyone through IBM branch offices.

S Sell: can be purchased by anyone through IBM branch offices.

L Licensed: property of IBM; available only to licensees of the related program products under the terms of the license agreement.

OS PL/I Checkout and Optimizing Compilers: Language Reference Manual
Order No. SC33-0009

The primary source of information on the language implemented by the OS PL/I Optimizing and Checkout Compilers. It is a reference manual rather than a tutorial text, and the reader is assumed to have some previous knowledge of PL/I.

The manual contains information on how to write a PL/I source program. For information on how to compile, link-edit, and execute the program, reference should be made to the appropriate programmer's guide.

OS PL/I Checkout and Optimizing Compilers: Keyword and Terminal Commands Reference Card
Order No. SX33-6002

Contains a full list of the PL/I Keywords implemented by the OS PL/I Optimizing Compiler. The keywords are arranged in alphabetical order and each has a brief description of its use.

OS PL/I Optimizing Compiler: Programmer's Guide
Order No. SC33-0006

Companion volume to OS PL/I Optimizing and Checkout Compilers: Language Reference Manual (Order No. SC33-0009), and TSO: OS PL/I Optimizing Compiler: User's Guide (Order No. SC33-0029). The three manuals form a guide to the writing and execution of PL/I programs using the Optimizing Compiler and the associated resident and transient libraries in the batch and time-sharing environments of the IBM Operating System.

The programmer's guide is concerned with the relationships between a PL/I program, the Optimizing Compiler, and the operating system. It explains how to compile, link-edit, and execute a PL/I program in a batch environment, and it introduces job control language, the linkage editor, data management, and other operating system features that may be required by a PL/I programmer.

OS PL/I Optimizing Compiler: Messages
Order No. SC33-0027

Lists all the messages that may be issued by the OS PL/I Optimizing Compiler and the associated transient library during processing of a PL/I program. Both the long and short forms of each message are listed where applicable.

The messages are in three groups:

- Compile-time messages (generated by the compiler during compilation of a PL/I source program).
- Execution-time messages (generated by the transient library during execution of the compiled program).
- Prompter messages (generated by the compiler when it is used in a time-sharing environment).

Where appropriate, explanations of the messages and suggested programmer responses are included.

OS PL/I Optimizing Compiler: Execution Logic
Order No. SC33-0025

Describes the object module produced by the OS Optimizing Compiler, and explains how

the compiled code uses subroutines from the associated resident and transient libraries.

The topics covered include program initialization, storage management, input/output, error handling, and interlanguage communication. The use of storage dumps for debugging is also explained.

The manual is intended primarily for programmers concerned with maintenance of the compiler and its libraries. It will also be useful for applications programmers who require an understanding of the processes of execution.

OS PL/I Optimizing Compiler:
Program Logic
Order No. LY33-6007

The internal design of the OS PL/I Optimizing Compiler is described in this manual, which is written for use by programmers responsible for maintenance of the compiler. The manual is intended primarily as a guide to analysis of the program listings by people who are generally familiar with the compiler. However, overall and component descriptions are provided for use by readers who have no prior knowledge of the compiler design.

OS PL/I Resident Library:
Program Logic
Order No. LY33-6008

The OS PL/I Resident Library consists of standard subroutines that are link-edited with object programs generated by the OS PL/I Optimizing Compiler. The library is used in program management, input/output, conversion, and interlanguage communication, and also includes many computational subroutines.

This publication is intended primarily for use by programmers concerned with the maintenance of the resident library. It summarizes, under functional headings, the internal logic of the libary subroutines, and gives brief details of the relationships between them.

OS PL/I Transient Library
Program Logic
Order No. LY33-6009

The OS PL/I Transient Library consists of standard subroutines that are loaded during execution of programs compiled by the OS PL/I Optimizing compiler, or the OS PL/I Checkout Compiler. These subroutines are used in error handling, program management, and input/output, and to provide storage dumps.

This publication is intended primarily for use by programmers concerned with the maintenance of the transient library. It summarizes, under functional headings, the internal logic of the library subroutines, and gives brief details of the relationships between them.

OS PL/I Optimizing Compiler:
Installation
Order No. SC33-0026

Intended primarily for programmers responsible for installing the OS PL/I Optimizing Compiler and the associated resident and transient libraries. It is divided into four parts that give detailed information on installation procedures and storage estimates.

TSO:
OS PL/I Optimizing Compiler: User's Guide
Order No. SC33-0029

This manual is a companion volume to OS PL/I Checkout and Optimizing Compilers: Language Reference Manual (Order No. SC33-0009) and OS PL/I Optimizing Compiler: Programmer's Guide (Order No. SC33-0006). The three manuals form a guide to the writing and execution of PL/I programs using the Optimizing Compiler and the associated resident and transient libraries in the batch and time-sharing environments of the IBM Operating System.

The manual comprises two parts:

- Guide: An explanation of how to use the compiler in a time-sharing (TSO) environment, and a description of the conversational I/O feature of the compiler.
- Command Language: A description of the PLI command and its operands.

The two parts are designed so that it can be used independently or its two parts included in the TSO publications Terminal User's Guide (Order No. GC28-6763) and Command Language Reference (Order No. GC28-6732), respectively.

Appendix D: Planning Information for CMS

The OS PL/I Optimizing Compiler and its libraries can be used with the Conversational Monitor System (CMS) of the Virtual Machine Facility/370 (VM/370).

The optimizing compiler supports those I/O devices that it supports under the operating system, provided that these devices are supported by VM/370. A list of the devices supported under VM/370 is given in the VM/370 Planning and System Generation Guide.

A minimum of 320K bytes of virtual storage is required for the CMS virtual machine for execution of the optimizing compiler under CMS.

All PL/I source programs that can be compiled by the optimizing compiler under the IBM Operating System will be accepted and compiled by the optimizing compiler under CMS.

However, object programs produced by the optimizing compiler to be executed under CMS cannot use the following facilities:

- Multitasking
- Teleprocessing. The TCAM data management facility and the TRANSIENT file attribute are included in this restriction.
- INDEXED and VSAM files.
- The PL/I sort facility.
- The PL/I checkpoint/restart facility.
- ASCII data sets.
- The BACKWARDS attribute for magnetic tape files.
- Spanned (VS-format or VBS-format) records (BDAM).
- PL/I FETCH and RELEASE statements.

These restrictions are execution-time restrictions under CMS; they do not apply to programs compiled under CMS and executed under the IBM Operating System.

For the availability of the optimizing compiler under CMS refer to IBM Program Product Design Objectives, Order No. GC33-0036.

Index

IBM / **Technical Newsletter**

This Newsletter No.	GN33-6095
Date	June 1973
Base Publication No.	GC33-0001-2
File No.	S360/S370-29
Previous Newsletters	GN33-6085

OS
PL/I Optimizing Compiler:
General Information

This Technical Newsletter, a part of version 1, release 2, modification 0 of the IBM Operating System PL/I Optimizing Compiler, Program Product 5734-PL1, provides replacement pages for the subject publication. These replacement pages remain in effect unless specifically altered. Pages to be inserted and/or removed are:

Title page, ii	21, 22
1, 2	23, 24
3, 4	27, 28
5, 6	29, 30
19, 20	33, 34

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

The attached pages contain minor additions or changes to the announcement information published previously for version 1, release 2, modification 0 of the OS PL/I Optimizing Compiler.

Note: *Please file this cover letter at the back of the manual to provide a record of changes.*

IBM United Kingdom Laboratories Ltd, Publications Dept, Hursley Park, Winchester, Hants, England.

Printed in U.S.A.

IBM / **Technical Newsletter**

This Newsletter No. GN33-6085
Date April 1973

Base Publication No. GC33-0001-2
System S360/S370-29

Previous Newsletters None

OS
PL/I Optimizing Compiler:
General Information

This Technical Newsletter provides replacement pages for the subject publication. These replacement pages remain in effect unless specifically altered. Pages to be inserted and/or removed are:

Title page, ii
37

A change to the text is indicated by a vertical line to the left of the change.

Summary of Amendments

Updated planning information for the use of the PL/I Optimizing Compiler under CMS is included.

Note: *Please file this cover letter at the back of the manual to provide a record of changes.*

IBM United Kingdom Laboratories Ltd, Publications Dept, Hursley Park, Winchester, Hants, England.

Printed in U.S.A.

OS
PL/I Optimizing Compiler:
General Information

Order No. GC33-0001-2

READER'S COMMENT FORM

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

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? _____

Number of latest Technical Newsletter (if any) concerning this publication: _____.

Please indicate in the space below if you wish a reply.

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

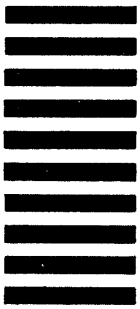
Your comments, please . . .

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

Cut or Fold Along Line

Fold Fold

First Class
Permit 40
Armonk
New York



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

Postage will be paid by:

International Business Machines Corporation
Department 813(HP)
1133 Westchester Avenue
White Plains, New York 10604

Fold Fold

IBM

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

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

OS PL/I Optimizing Compiler: General Information (File No. S360/S370-29) Printed in U.S.A GC33-0001-2