Program Product

IBM System/360 Operating System PL/I Checkout Compiler General Information

Program Number 5734-PL2

This publication is a planning aid only. It is intended for use prior to the availability of the following IBM System/360 program products:

- OS PL/I Checkout Compiler, Program Product 5734-PL2
- OS PL/I Transient Library, Program Product 5734-LM5

Used in conjunction with the program product publication *IBM System/360 Operating System: PL/I Language Reference Manual (Preliminary),* Order No. SC33-0009, this publication enables installation managers, systems analysts, and programmers to plan and write PL/I programs that are to be compiled and executed upon availability of these program products.

IBM

Preface

This publication contains a description of the PL/I Checkout Compiler for the IBM System/360 Operating System.

The subjects covered include the compiler facilities, its conversational features, a summary of the PL/I language implemented, and the system environment. The appendixes contain a list of keywords, a comparison with the PL/I (F) Compiler, and a discussion of the PL/I Checkout Compiler and the PL/I Optimizing Compiler as a pair.

Recommended Publications

The PL/I language implemented by the PL/I Checkout Compiler is described in detail in the publication:

IBM System/360 Operating System: PL/I Language Reference Manual (Preliminary), Order No. SC33-0009

The following publications contain further information on subjects referred to in this publication:

IBM System/360 Operating System: Planning for Time Sharing Option, Order No. GC28-6698

The following publications contain other information of interest to the PL/I programmer or the programmer who is learning PL/I:

A PL/I Primer, Order No. SC28-6808

A Guide to PL/I for Commercial Programmers, Order No. SC20-1651

A Guide to PL/I for FORTRAN Users, Order No. SC20-1637

Introduction to the List Processing Facilities of PL/I, Order No. GF20-0015

Introduction to the Compile-Time Facilities of PL/I, Order No. SC20-1689

First Edition (July, 1970)

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems, refer to the latest IBM System/360 Bibliography SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

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

A form for reader's 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., Programming Publications, Hursley Park, Winchester, Hampshire, England.

Contents

Tables

The PL/I Checkout Compiler is a new, high-performance processor for use in either batch or conversational mode. For both types of user it offers substantially increased programmer productivity. It does so by:

- checking PL/I programs very thoroughly.
- providing clear and detailed diagnostic information, couched in PL/I terminology for ease of understanding.
- providing fast translation.

As a result, the task of debugging is made considerably easier, and the turnaround between debugging sessions is reduced.

Additionally, the conversational user can communicate with the compiler or the system during translation, and with his program or the system during interpretation. He can also send or receive data during interpretation. Translation-time diagnostic messages can be made available to the user at the end of the syntax check and at the end of the 'global' check (explained below). This allows the user to interrupt processing, correct the errors using a system update facility, and then restart translation. Execution-time diagnostics can be made available to the user during interpretation, or the user can interrupt interpretation. An interruption of either sort returns control to the terminal. The user then has the choice of:

- Supplying PL/I statements for immediate interpretation. This permits, for instance, checking the values of program variables.
- Resuming interpretation either at the point of interruption or at any other point in the program.
- Making temporary corrections to the program and resuming interpretation.
- Terminating interpretation, making a permanent correction to the program, and restarting translation.

These conversational facilities are provided by new language features (described later) and through the Time Sharing Option (TSO), an operating system feature that permits several users to have concurrent access to the computer.

The checkout compiler translates PL/I source statements into an intermediate code that is not suitable for execution by the machine. Execution of the code is performed by interpretation. However, the compiler does not supply all the code required to represent the source program; instead, it inserts references to subroutines that are stored in a library. These subroutines are loaded, executed, and discarded as required during program interpretation.

The library is not an integral part of the checkout compiler. The IBM program product is called the IBM System/360 Operating System PL/I Transient Library. In this manual, the term 'transient library' always refers to the IBM program product.

Translation with the checkout compiler is, on average, about four times faster than compilation with the PL/I (F) Compiler. Interpretation is performed, on average, at one fifth the speed of execution of the (F) compiler.

Complete programs can be translated and interpreted without the overheads associated with link-editing. A further saving on overhead time can be achieved by processing more than one program in a single job step.

The checkout compiler implements the same level of language as the (F) compiler, plus a number of extensions and improvements which are described in this publication.

Terminology

The PL/I Checkout Compiler, and the features of the IBM System/360 Operating System with which it is associated, may introduce a number of terms with which the reader is unfamiliar. Those occurring most often are defined below; others are defined as they occur within the text.

Background	Refers to the environment in which jobs submitted via SYSIN or through the TSO SUBMIT command are executed. One job step at a time occupies a region of main storage, and remains in storage until completion.

Background job	A job entered via SYSIN or the SUBMIT command.
Batch	Describes the processing of one job step at a time in a region of main storage. So called because jobs are submitted in batches via a SYSIN device.
Command	(Under TSO.) A command is a request from a remote terminal for the execution of a particular command processing program.
Conversational processing	Processing a program within an environment in which the user is able to carry on a dialogue with the system or his program. (The system response time is such that the user is able to maintain his train of thought.)
Foreground	Describes the environment in which programs invoked by commands are processed. Programs are swapped in and out of main storage as necessary to utilize main storage efficiently.
Global checking	The second phase of program translation in this compiler. It includes checking declarations for consistency of attributes, checking argument usage, and constructing the interpretable code. The interpretable output may, optionally, be put out onto a direct-access device.
Immediate PL/I statements	PL/I statements entered conversationally during the execution of a program. Such statements are executed immediately within the environment of the active block, and are not stored.
Interpretation	The processing of problem data against the interpretable code produced by the translator. This term corresponds to the term 'execution' used in other compilers.
Interpretive checking	Checking for programmer errors during interpretation. (For example, the use of uninitialized variables.)
Subcommand	(Under TSO.) A terminal input line recognized by a command processing program (e.g., a problem program) and which requests a particular action.
Swapping	(Under TSO.) To write an image of a user's main storage region onto auxiliary storage and to read in another user's main storage image.
Syntax checking	The first phase of program translation in this compiler. It includes the recognition and checking of options, enforcing the rules governing the spelling of keywords, positioning of delimiters and similar rules not concerned with program logic.
Time sharing	A method of using a computer system that allows a number of users to execute programs concurrently and to interact with them during execution.
Time Sharing Option (TSO)	An optional configuration of the operating system supporting conversational time sharing from remote terminals.
Translation	The first part of the processing cycle, combining syntax and global checking to produce the interpretable code used during interpretation. Translation corresponds to the compilation phase in other compilers.

Chapter 1: The PL/I Checkout Compiler

The checkout compiler is a high-performance language processor for developing and checking PL/I programs in both batch and conversational mode. It provides syntax checking, global checking, and checking at execution. Checking is very thorough, and the diagnostic messages are highly informative. All communication with the user which originates from the compiler is in PL/I terminology.

Compiler Performance

Translation Speed

On a System/360 Model 65 with a 2314 direct-access device, the compiler will translate about 2,500 PL/I statements per minute, with a system overhead time of about 3 seconds per program.

Interpretation Speed

On a System/360 Model 40, the compiler will interpret about 500 PL/I statements per second.

Batch Mode

In batch mode, about 600 small jobs can be processed per hour on a System/360 Model 65 if programs are processed in groups. This figure assumes a job size of about 25 PL/I source statements representing about 3,000 executed statements.

Main Storage Requirements

The checkout compiler is designed to function efficiently when 100K bytes of main storage are available to it. It will function in an absolute minimum of 80K, but at this level there may be some loss of efficiency.

Program Size

A source program should not contain more than about 10,000 PL/I statements.

When the compiler has 100K of main storage available to it, the total number of symbols and constants permitted is about 2,000. This can increase up to a limit of 8,000, provided that sufficient main storage is available.

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.
- Specify the margins for source statements.
- Control progress into load-and-go depending on the severity of the diagnostics.
- Produce object code specifically designed for a particular computer model.
- Specify whether full or short diagnostic messages are to be printed.
- List statement numbers of declaration (if applicable) and all attributes assigned to all identifiers in the program.
- List the External Symbol Dictionary.
- Specify the number of lines to be printed on each page of output.
- Print block level and iterative 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 source listing is to be formatted.
- Place a limit on the number of statements executed.
- In batch mode, suppress printing of diagnostic messages until a specified number of statements have been executed.
- Obtain a formatted dump when an error terminates processing.
- Specify the stringency of execution checking.
- Specify whether 4-byte or 16-byte pointers and offset variables are to be used.
- 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.
- Allow the use of extended precision machine instructions.
- Specify the number of branch statements and their targets to be printed when requested.

Processing Mode

The checkout compiler can process source programs either conversationally (in a foreground region), or in batch mode (in a background region). Alternatively, they may be initiated in a foreground region for subsequent batch processing.

The choice of processing mode, batch or conversational, has no effect on the action of the translator or interpreter phases; they perform their functions (provided that these functions are available in both modes) in the same way for either mode. The feature most affected by the choice of mode is the method of establishing the program environment.

In batch mode, job control statements provide information on the program environment. For example, parameters passed on the EXEC statement provide the means of selecting compiler options; DD statements associate PL/I files with data sets. If the linkage editor is required, it is invoked by the appropriate job control statements.

In conversational mode, job control statements for the program environment consist of pre-established DD statements; these are invoked by TSO commands that pass any parameter and ddnames required. If the linkage editor is required, it is invoked by a TSO command.

Batch Mode Processing

Input stream: This is submitted by means of a sequential data set. The input is a PL/I source program, and (optionally) a data set containing data to be processed by the source program.

Processing: The unit is the job; execution of a job is a task. A job is processed under the supervision of an operating system control program. (See 'Control Programs' in Chapter 4.)

Output: System output, including diagnostic messages, is collected on an output data set and is printed on a listing when the job has finished.

Note: Jobs submitted on a sequential data set are often grouped in batches for processing; this gives quicker turnaround and more efficient use of resources. This mode of processing is thus known as batch processing, even though a job submitted for batch processing may consist of only a single PL/I source program for compilation and execution. If each PL/I program is processed as a separate job, the compiler has to be invoked, and resources (that is System/360 devices) have to be allocated separately for every program. The system overhead in time that this method causes can be greatly reduced if several programs are compiled together in one job; there is only one invocation of the compiler and one allocation of resources. The technique of submitting several programs together for compilation is called batched compilation. Each program can be interpreted immediately following translation, or the output for each program can be executed by itself or

combined with the output from other programs for execution.

Conversational Mode Processing

Input stream: This is submitted or initiated at a terminal. The input is a series of commands identifying a data set (which may be the terminal itself) that contains a PL/I source program, and, optionally a data set that contains data to be processed by the program.

Note: A terminal in this context is frequently a keyboard device; in general terms, however, a terminal can vary from such a device to another CPU.

Processing: The unit is a session, which is the period of time a terminal is continuously under the control of one user (that is, the period between LOGON and LOGOFF). A session is processed under the supervision of the MVT control program, with the time sharing option.

Once a session has started, the conversational facility is available. By using the appropriate command the user can send or receive information to or from the PL/I program, which, similarly, can send or receive data. The user is able to make temporary alterations to the PL/I source program during processing. The program is allocated a partition or region, which under TSO may be 'swapped' with other users, or retained throughout the session.

Output: System output, including diagnostic messages, is put out as it occurs, and can be printed, if required, at the terminal. Program output data can also be sent to the terminal.

Program Checkout

Processing of PL/I source programs is handled in two phases by the checkout compiler. The Translator phase provides syntax and semantic (or 'global') checking and produces text for interpretation. The Interpreter phase interprets the code produced by the translator and provides checking during interpretation.

Syntax Checking

The syntax checking routines perform two main functions: checking PL/I source statements and producing a source program listing.

The checking performed is:

1. Recognition and checking of compiler options and file names.
2. Syntax checking within single statements.
3. Syntax checking across several statements, for example, that each DO group has a matching END statement.

The source program listing can, optionally, be formatted so that the relationships of the various blocks, groups and compound statements is displayed. Abbreviations in the source program are printed in full.

Diagnostic messages produced by the syntax checking routines are collected and (in conversational mode) printed at the terminal at the end of syntax checking. Using the TSO EDIT command, corrections can be made to the source program and translation restarted. In batch mode, the messages are collected for eventual output on the SYSPRINT file.

Global Checking

The global checking routines perform three main functions: checking PL/I source statements for semantic errors, producing text for execution, and producing information for the listing.

The checking performed includes:

1. Checking declarations for consistency of attributes.

2. Checking arguments for correct usage; missing labels are detected at this stage.

The global checker produces, as primary output, text intended for immediate interpretation.

As an optional alternative, it can produce:

1. An object module suitable for input to the linkage editor.

2. Tables of information which will be needed at execution time.

The information produced by the global checker for the listing is optional and includes an attribute and cross-reference table.

Diagnostic messages produced by the global checking routines are collected and (in conversational mode) printed at the terminal at the end of global checking. Using the TSO EDIT command, corrections may be made to the program. In batch mode, the messages are collected for eventual output on the SYSPRINT file.

Interpretation Checking

The interpretation routines perform two main functions: checking variables and storage allocation, and interpreting code produced by the translator.

Checks are made to detect:

1. The use of uninitialized variables.

2. Misuse of DEFINED variables and label variables.

3. Misuse of pointer references:

   a. to I/O buffers no longer available.

   b. to data with attributes that differ from those of the name specified.

4. Misuse of storage allocations:

   a. attempts to retrieve a value from storage that has previously been freed.

   b. attempts to free (by means of a pointer reference) part of a storage allocation (for example, part of a structure).

   c. attempts to free storage that has already been freed.

5. Illegal branches into DO groups.

6. Branches into inactive blocks.

Under certain circumstances, some of these checks may not be performed:

1. Uninitialized variables and misuse of label variables cannot be detected if these variables are part of a module produced by another compiler that has been link-edited with a module produced by the checkout compiler.

2. In pointer references to data, if separately compiled object modules have been link-edited together, then mismatch of attributes between the data reference and the identifier in the pointer reference cannot be detected if the data and the identifier are in separate compilations.

3. If compatibility with code from other compilers requires that the short form of the internal representation of pointers should be used, then checks on pointer references cannot be made.

In conversational mode, diagnostic messages produced by the interpreter are printed at the terminal immediately they occur. Temporary correction to the program can be made at this point, but these corrections are not permanently incorporated into the source program.

Uninitialized Variables

The following implementation restrictions on the initialization of variables should be noted.

Range of values: Checking for uninitialized variables is done by assigning a special value to the variable immediately after allocation. An attempt to use a variable which still has this special value raises the ERROR condition, and an ON-code relating to the error is set. (This code can be obtained by using the ONCODE built-in function in an appropriate ON-unit.) The special value is one which the programmer would not normally be expected to use. Should he do so intentionally, he must be prepared for the results described above. (Alternatively, he can disable checking for uninitialized

variables, by specifying a compiler option.) The special values which the programmer could conceivably use are:

FIXED BIN(15)	maximum negative number
FIXED BIN(31)	maximum negative number
CHAR non-varying	X'FE' in first byte
pictures	X'FE' in first byte

Static variables: Initialization of static variables in a module from another compiler takes place when the program module is loaded into main storage. If execution is recommenced, by means of a GO TO 0 statement, static variables will not be reinitialized if they are in a module from another compiler that has been link-edited with a module from the checkout compiler.

Compatibility with the PL/I (F) Compiler

Source programs written for the PL/I (F) Compiler can be compiled by the PL/I Checkout Compiler and will execute correctly without modification, except for a few minor differences which exist between these implementations of the language. These incompatibilities are described in Appendix B.

Object modules that have been compiled by the PL/I (F) Compiler, and modules from the PL/I (F) Compiler subroutine library, cannot be incorporated into programs compiled by the PL/I Checkout Compiler. This problem can be overcome by recompilation of all PL/I source programs with the checkout compiler.

A PL/I source program coded for use with the (F) compiler can be stored on a data set and invoked from the terminal for processing by the checkout compiler.

Interlanguage Communication

Facilities are available which allow procedures compiled by the PL/I Checkout Compiler to communicate at execution time with programs compiled by any COBOL or FORTRAN compiler produced by IBM for the System/360 Operating System.

Thus it is possible for existing COBOL and FORTRAN users to write new applications in PL/I while still utilizing existing COBOL and FORTRAN libraries; in addition, existing applications can be modified by the use of PL/I procedures.

Communication between programs written in different languages is specified in the usual way, by a CALL statement or, for FORTRAN and PL/I, by a function reference.

The interlanguage communication facilities are requested in the PL/I procedure by the COBOL or FORTRAN option of the OPTIONS attribute or option. The remapping of COBOL structures, and transposing of FORTRAN arrays, which would then normally take place, can be completely or partially suppressed by the NOMAP, NOMAPIN, and NOMAPOUT options. The INTER option can be used to specify that the COBOL and FORTRAN interrupts which would otherwise be handled by the system, are to be handled by the PL/I interrupt-handling facilities.

Chapter 2: Conversational Features of the Compiler

To use the checkout compiler in conversational mode, the programmer must first initiate the session at the terminal. He does so with the LOGON command. (Similarly, when the session is complete, he terminates it with the LOGOFF command.) After signing on, the next step is to invoke the required program(s) and/or data sets. To do this, the TSO commands described below are used.

The TSO Commands

The following TSO commands may be used by a programmer at a terminal in order to process his program.

EDIT creates or updates a sequential data set, or a member of a partitioned data set. This data set or member may contain either a source program for the checkout compiler, or data to be processed by a PL/I program.

PLI invokes the PL/I prompter and checkout compiler.

The PL/I prompter ensures that the parameters entered with a command are complete and correct. If they are not, it 'prompts' the user to supply the correct information.

LINK invokes the linkage editor.

CALL invokes the loader.

RUN similar to the PLI command, but can be used either as a command or as a subcommand.

LOADGO invokes a user-written program.

Conversational Processing

When using the checkout compiler in conversational mode, a user can intervene at the terminal during translation and interpretation. He can insert immediate PL/I statements during interpretation, and he can provide values for variables declared in his program. The program can use the terminal as a data set. In this case, output will be printed at the terminal whenever the program logic requires it, and input must be provided by the user in response to a message from the program.

To intervene in processing, the user must obtain control at the terminal, and then provide some input. To do so, he must be able to:

1. Recognize the circumstances under which control is passed to the terminal.
2. Understand any message which may be produced when the terminal receives control.
3. Provide suitable input, either immediate PL/I statements, or TSO commands or subcommands.

Receiving Control at the Terminal

The situations in which control is passed to the terminal are described below.

Attention: When the user strikes the attention key, the immediate response at the terminal is that an exclamation point is printed. If the terminal was printing data, printing ceases. The terminal then has control.

If he strikes the attention key again before the terminal receives control, operation of the checkout compiler is terminated and control passes to the control program.

Breakpoints: A user can cause control to be passed to the terminal during the execution of his program, either by including the HALT statement in his program, or by using the AT, ABOVE, or STEP subcommands (discussed later in this chapter).

Condition: If the ERROR condition is raised and there is no ON-unit execution stops and standard system action is performed. This passes control to the terminal.

Completion: At normal termination of execution, control passes to the terminal.

Messages

During the processing of a program in conversational mode, compiler-generated messages will be printed at the terminal. These may be either progress messages, indicating the current status of the program, or an indication that control has returned to the terminal. In the latter case, the reason for passing control will be given in the message.

Translator messages: One or more of the following messages will be put out:

Message	Explanation
SYNTAX CHECKING	The user has struck the attention key during syntax checking. Control is not passed to the terminal.
SYNTAX CHECKING COMPLETE	Syntax checking is complete and some diagnostic messages may have been produced. Control is passed to the terminal only if errors have been found.
GLOBAL CHECKING	The user has struck the attention key during global checking. Control is not passed to the terminal.
GLOBAL CHECKING COMPLETE	Global checking is complete and some diagnostic messages may have been produced. Control is passed to the terminal only if errors have been found.
NO SYNTAX ERRORS	This is a progress message only; control is not passed to the terminal.
NO GLOBAL ERRORS	This is a progress message only; control is not passed to the terminal.

Interpreter messages: Messages issued during interpretation will have the following form:

```
message-number  executed-count  reason
statement-number [procedure-name]
```

where 'executed-count' is the number of statements executed since execution last began.

'reason' indicates the error or other event which caused control to be passed.

'statement-number' is the number of the PL/I statement at which control was passed. (In each invocation of the checkout compiler, immediate-mode statements are numbered serially.)

'procedure-name' provides information identifying the procedure which was being executed at the time control is passed. This information is omitted if it is the same as in the last message of this type.

If interpretation has finished when control is passed, the message is:

```
EXECUTION FINISHED
```

System message: If the system is awaiting input and the user strikes the attention key, the following message will be issued:

```
INPUT MAY BE REQUIRED
```

Input at the Terminal

All input from the terminal must be in PL/I mode if it is made in response to messages generated during interpretation. This specifies that the end of a line constitutes the end of a statement unless a continuation character is used. Both the messages and the response are copied onto SYSPRINT (unless the terminal is designated as the SYSPRINT output device).

The input can be any of the following:

- A TSO PL/I subcommand, which allows temporary changes to be made to the program.
- One or more immediate PL/I statements.
- Data.

TSO PL/I Subcommands

These commands can be entered at the terminal during execution. They allow the user to interrupt or resume execution, or enter additional PL/I statements for immediate execution. The command statements are detailed below.

The AT and ABOVE Commands

These cause execution to be suspended immediately before the execution of a

specified statement (which can be identified by either a statement number or a label). Either command allows the user to insert statements for immediate execution, modifying or adding to the existing program; alternatively, the commands may be used simply to return control to the terminal.

The two commands differ only when the statement specified is entered as the result of a branch instruction. In that case, the ABOVE command has no effect.

It will be common for the user to require one or more additional PL/I statements to be executed at some point in his source program. This can be achieved by providing an AT-unit.

An AT-unit consists of one or more PL/I statements preceded by the keyword AT and a statement number. This statement, or group of statements, is then inseparably associated with the AT statement for the remainder of that session, unless canceled.

For example:

```
31            X=SUBSTR(FILEA,1,2);
32     LAB:   IF X ¬= 'AB' THEN DO;
```

The statement:

```
AT 32 DO;
         PUT DATA(X);
         GO;
       END;
```

would result in the current value of X being printed each time control passes through the label LAB.

The statement:

```
ABOVE 32;
```

causes control to be returned to the terminal immediately following the execution of statement 31. This command would not be effective following the execution of a statement GO TO LAB; at any other point in the program.

The END Command

This terminates execution and returns control to the command level. To avoid confusion with a PL/I immediate mode statement, this command cannot be used in the same piece of input as any other statement.

The GO Command

This causes program execution to be resumed at the point where it was suspended.

The HELP Command

This is used to return the full text of a specified message.

The MONITOR/NOMONITOR Commands

The MONITOR command causes data being written onto SYSPRINT to be copied at the terminal.

In this implementation, if the user strikes the attention key while such data is being printed at the terminal, some of the output may be omitted from the terminal printout; the whole output will nevertheless appear on SYSPRINT.

The NOMONITOR command suppresses the action of the MONITOR command.

The OFF Command

This nullifies any AT or ABOVE breakpoints specified for a given statement. The OFF command without an argument nullifies all existing AT or ABOVE breakpoints.

The QUALIFY Command

The QUALIFY command has the effect of substituting the environment named in the command for the current environment, for name resolution purposes only; the action specified in the current environment would still obtain if, for example, an interrupt occurred as a result of the PL/I statements entered after the QUALIFY command.

For example:

```
QUALIFY A;
```

Any identifiers or immediate PL/I statements entered after this command are assumed to be within the scope of A, not that of the current block. The effect is canceled when control returns to the user's program.

The STEP Command

This causes program execution to be suspended after the execution of a specified number of statements.

For example:

```
STEP 50;
```

Program execution will be suspended, and control passed to the terminal, after the execution of the fifty PL/I statements following the STEP command. When execution is resumed this action will be repeated every fifty statements until the program terminates or another STEP command varies the response.

Immediate PL/I

The immediate PL/I language is described in Chapter 3. If one or more statements are entered which do not result in output at the terminal, the number of the last statement entered is printed at the terminal to indicate that the statement(s) have been received for processing.

If a sequence of immediate PL/I statements is grouped on one line, or in a DO group, performance will be improved, as the compiler will then be loaded only once instead of once for each statement.

Processing Example

The following example shows a typical sequence of events during the execution of a program initiated at a terminal.

The checkout compiler is invoked with the command PLI. Assume that default options are used and terminal listings are not required. The programmer types in:

```
PLI xyz
```

where 'xyz' represents the name of the data set member containing the program he wishes to execute.

The terminal returns a progress message:

```
V1 OS99 OS/360 CHECKOUT COMPILER   DATE....
TIME....
OPTIONS SPECIFIED....
```

which gives some accounting information and informs the programmer that translation of the program has started.

Assuming that no syntax or global errors are found, the terminal will show the progress messages:

```
IENxxxx  NO SYNTAX ERRORS

IENxxxx  NO GLOBAL ERRORS
```

Assume that the user is proceeding straight from translation to interpretation. Further assume that statement number 120 of his program is:

```
X = A/B;
```

and that on execution of the statement, B has the value zero. If this value is not allowed for in the program, interpretation will be interrupted and a message printed at the terminal:

```
IENxxxx  ZERODIVIDE A = 27  B = 0  X433
120  IN P

IENxxxx  ERROR nnn
```

where P is the name of the program (obtained from the data set xyz). X433 indicates that 433 statements had been executed before the error occurred.

The programmer may wish to know how control reached statement 120. To obtain this information he types:

```
PUT FLOW;
```

The statement numbers of the origin and destination of the previous 'n' branches are listed, with the latest branch listed first. ('n' is specified by a compiler option.) The number I1 will also be printed, indicating that the first immediate mode statement has been received for processing.

The programmer could decide to make a temporary amendment to the program by inserting a new branch instruction immediately before the statement:

```
X = A/B;
```

He types:

```
AT 120 DO;
      IF B=0 THEN GO TO ABC;
      GO;
   END;
```

The number I3 is printed to acknowledge these commands.

To restart the program at the beginning, the programmer would then type:

```
GO TO 0;
```

The response I4 acknowledges this statement.

Assuming that on this execution the program completes normally, the message:

```
EXECUTION FINISHED
```

informs the programmer, who then types an END command to complete the session. The message READY is printed indicating that the checkout compiler has completed processing and that TSO commands are available to the user.

Chapter 3: The PL/I Language Implemented by the Checkout Compiler

New language features, together with a number of improvements on the language implemented by Version 5 of the PL/I (F) Compiler, have been included in the language implemented by the checkout compiler. These features, which are described below, have been incorporated both to allow the use of conversational programming and to increase the function and flexibility of PL/I. Appendix A contains a complete list of the keywords implemented by the checkout compiler.

In general, source programs written for the (F) compiler can be translated and interpreted by the checkout compiler without amendment. However, there are some minor incompatibilities, and these are detailed in Appendix B.

Language Features

The language implemented by the PL/I Checkout Compiler contains the following programming features:

Data Types

- Character-string and bit-string data.
- Fixed-point binary and fixed-point decimal data.
- Floating-point binary and floating-point decimal data.
- Character and numeric picture data.
- Real and complex arithmetic data.
- Label and entry data.
- File data.
- Task and event data.
- Pointer and offset data.

Operations

- Assignment, with automatic conversion between data types if necessary.
- 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, multitasking, and error handling.
- Pseudo-variables for computation and error handling.
- Facilities for creating programmer-defined functions.
- Dynamic storage allocation.

Data Aggregates

- Arrays of data elements.
- Structures of data elements.
- Arrays of structures.
- Areas.

Program Management

- Separate compilation of external procedures of the same program.
- Structuring of program into blocks to limit the scope of names and permit flexible allocation of storage.
- Recursive invocations of a procedure with stacking and unstacking of generations of automatic data.
- Multitasking facilities.

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.

Conversational Features

- Trace of identifiers.
- Passing control to the terminal.
- Program modification.

Language Extensions

Language features implemented by the PL/I Checkout Compiler which are only partially implemented, or not implemented at all, by the PL/I (F) Compiler are described below.

CONVERSATIONAL PL/I

The ability of the checkout compiler to process programs in conversational mode has created a need for new PL/I language features. These features can be divided into two categories:

- Program check
- Immediate PL/I

New language in both these categories is described below. Conversational processing also requires the use of a series of commands which may be entered at the terminal during execution to act as instructions to the operating system. These are fully discussed under the heading 'TSO PL/I Subcommands' in Chapter 2.

Program Check

New language for program checking has two main functions:

- Printing information at the terminal concerning the status of program interpretation.
- Passing control to the terminal.

Details of program interpretation may be traced using the CHECK and FLOW statements.

The CHECK statement causes information about specified or assumed identifiers to be written on the SYSPRINT file whenever these identifiers appear during program interpretation. This information will continue to be printed until the program terminates or until a NOCHECK statement nullifies the action of the CHECK statement.

The FLOW statement causes information about the transfer of control during interpretation to be written onto the SYSPRINT file. The FLOW statement remains active until the termination of the task, or until a NOFLOW statement is executed in the same task.

Control is passed to the terminal when a HALT statement is encountered. Execution is suspended immediately following the execution of this statement, and resumed only when the appropriate terminal command is entered. The HALT statement has no effect in background processing.

Immediate PL/I

Statements used to modify a PL/I program during interpretation can be entered via a terminal. Such statements are known as 'immediate PL/I'. They are executed immediately on entry, but are effective only within the block whose execution has been suspended, and will be known only to that block. They are therefore subject to certain restrictions:

1. Some PL/I statements cannot be used in immediate mode. These are statements that change the block structure of the program, or that introduce declarations:

   An END statement cannot be entered on its own.

   Immediate mode statements cannot be labeled.

   An unlabeled DO group may be entered, but will not be executed until the END statement has been entered.

   BEGIN, PROCEDURE, ENTRY, DECLARE, DEFAULT, FORMAT, REVERT, and ON statements cannot be used.

2. All identifiers used in immediate PL/I statements must be known in the current block. However, the scope of an identifier in immediate PL/I can be changed by the QUALIFY command (see 'TSO PL/I Subcommands' in Chapter 2).

Apart from these restrictions, all other PL/I statements may be used.

GO TO statement: The GO TO statement, in immediate mode, may specify its target statement by statement number.

For example:

```
GO TO 37;
```

Within a source program, however, the target statement must still be identified by a label constant or element-label-variable. This form of target identification may also be used in immediate mode.

NON-CONVERSATIONAL EXTENSIONS

The CHECK/NOCHECK and FLOW/NOFLOW statements described above are equally applicable in either conversational or batch mode. Other language extensions, which are similarly applicable in either mode of processing, are described below.

Program Checkout

Variables in program checkout: Variables specified in the name-list of the CHECK condition may be of any problem data type and any storage class. 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) RETURNS (FIXED DECIMAL);
      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) DEF A (4*ISUB);
```

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

PUT statement: Options of the PUT statement are available that can cause a list of active procedures, or the current values of specified variables, to be written on the specified file. The options available, and the effects of their use are:

- The SNAP option: returns, to the designated file, a calling trace indicating the current status of the program.
- The FLOW option: specifies that a more detailed calling trace is to be printed on a designated file. The output from this option includes the statement numbers for source and target statements in the previous 'n' transfers of control (where 'n' is defined by compiler option).
- The ALL option: includes all the other options; in addition, returns the values of all program variables and of the CNCHAR, ONCODE, ONCOUNT, ONFILE, ONKEY, ONLOC, and ONSOURCE built-in functions.

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 explicitly or contextually declared for:

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.

Certain attributes or types of identifier require explicit declaration and cannot be specified in a DEFAULT statement. They are:

Structuring

External entry names

A DEFAULT statement consists of a list of default specifications separated by commas. A default specification consists of a range specification, which indicates the range of applicability of the DEFAULT statement, and a list of default attributes for identifiers or descriptors in the specified range. The forms of range specification are as follows:

RANGE	Designates a particular range of identifiers (determined by their initial letters) to be influenced by the default specification.
DESCRIPTORS	Designates that non-null descriptors in explicit ENTRY attributes are to be influenced by the default specification.

Default values for precision, string lengths, and area sizes may be established by use of the VALUE clause in a default specification.

A DEFAULT statement may be specified in any block. It applies only to identifiers declared within that block or any inner block. A DEFAULT statement in an internal block overrides the effect of DEFAULT statements in outer blocks.

The semantics of the DEFAULT statement are illustrated by the following examples.

1. DEFAULT RANGE (ALPHA)....;

   The default attributes apply only to identifiers that begin with the letters ALPHA.

2. DEFAULT RANGE (A:C)....;

   Only identifiers beginning with the letters A, B, or C are subject to the DEFAULT statement.

3. DEFAULT RANGE (*)....;

   All identifiers are subject to the DEFAULT statement.

4. DEFAULT RANGE (L:O) VALUE (FIXED DECIMAL (7), FIXED BINARY (10), FLOAT DECIMAL (3), CHAR (16));

   ```
   DCL L FIXED DEC,
       M CHAR;
   ```

   The above identifiers, together with the implicitly-declared identifiers N and O have attributes and precisions (or string lengths) as follows:

   ```
   L FIXED DECIMAL (7)
   M CHAR (16)
   N FIXED BINARY (10)
   O FLOAT DECIMAL (3)
   ```

5. DCL X ENTRY (FIXED,BINARY,DECIMAL);

   The standard defaults would result in the following descriptors:

   ```
   FIXED DECIMAL REAL
   FLOAT BINARY REAL
   FLOAT DECIMAL REAL
   ```

   The statement:

   ```
   DEFAULT DESCRIPTORS BINARY FIXED
   COMPLEX;
   ```

   would cause the descriptors for the parameters of X to be:

   ```
   FIXED BINARY COMPLEX
   FIXED BINARY COMPLEX
   FIXED DECIMAL COMPLEX
   ```

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.

Storage Control

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

Self-defining structures: 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);
```

Note: The PL/I (F) implementation of REFER options in based variable declarations is subject to a great number of restrictions. Many of these have been removed or modified in the language implemented by the checkout compiler.

The ENVIRONMENT Attribute

Changes and extensions to the options specifying record format, block size, and record size, give greater flexibility. The new options are:

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

Block size: [BLKSIZE(blocksize)]

Record size: [RECSIZE(recordsize)]

Use of one of these options of the ENVIRONMENT attribute no longer requires that the other two be specified. Missing information may be supplied in a data-set label, a DD statement, or by default. 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 checkout 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 checkout 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/REWIND option: The REREAD option is the checkout compiler equivalent of REWIND in the PL/I (F) implementation. The checkout compiler recognizes either keyword.

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 reference may be a structure or array which contains VARYING 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);
```

File Names

File-name expressions: File names can be specified, in input/output statements, as expressions.

For example:

```
DCL B(10) FILE STREAM,
    X(Q)  FILE RECORD;

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.

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 an array or 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:

```
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 P POINTER STATIC INITIAL(NULL),
    B AUTOMATIC INITIAL(F(X)),
    C INITIAL(SQRT(3));
```

Note: 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, a compiler option for extended floating-point precision may be selected. This gives maximum precisions of:

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

The default precision for floating-point data remains as for 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 constants: 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 need be explicitly declared with the ENTRY attribute only if external.

Entry variables: 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  (BINARY,FIXED DECIMAL)
        VARIABLE,
     Y  ENTRY  (CHAR) AUTO,
     Z (10) ENTRY;
```

Entry variables may be assigned, passed as arguments, returned from procedures, 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 as an entry name; it is merely an entry expression that describes the entry name of the procedure to be invoked in specified circumstances. 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 */
A:  PROC....;
B:  PROC....;
C:  PROC....;
```

Arc Sine and Arc Cosine Computation

New mathematical built-in functions are available, for computing arc sines and arc cosines. They are ASIN(x) and ACOS(x) respectively.

1. ASIN(x)

   Argument: Real or complex. If real, the absolute value of x must not be greater than 1.

   Result: Arc sine, expressed in radians. If real, it lies within the range:

   $-\pi/2 \leq \text{ASIN}(x) \leq \pi/2$

   If the argument was complex, the result is complex.

2. ACOS(x)

   Argument: Real or complex. If real, the absolute value of x must not be greater than 1.

   Result: Arc cosine, in radians. If real, it lies within the range

   $0 \leq \text{ACOS}(x) \leq \pi$

   If the argument was complex, the result is complex.

Comparison of Labels

A comparison of label constants and label variables can be made. The comparison is limited to the use of the = and ¬= operators.

Chapter 4: System Requirements

Machine Requirements

The minimum machine requirements for the PL/I Checkout Compiler are an IBM System/360 Model 40, with a main storage capacity of 128K bytes, of which at least 80K bytes must be available for the compiler. For fully efficient use, the compiler requires a minimum of 100K bytes. The central processing unit must have the decimal and floating-point instruction sets and, if timing information is required, it must have the timer feature.

If the conversational features of the compiler are to be used, the time sharing option (TSO) of the operating system must be available. This has a minimum requirement of an IBM System/360 Model 50 with a main storage capacity of 512K bytes.

Compiler Residence

The compiler will occupy approximately 700K bytes of direct-access storage space. In addition, 75K bytes will be required for the transient library used by the compiler.

Working Storage

The compiler requires direct-access storage space for overflow storage areas. The amount of space required depends on the size of the program and the amount of main storage available to the compiler. The data set identified by SYSUT2 is used for this auxiliary storage. In addition, the compiler may require direct-access storage space for the work files SYSUT3, SYSUT4, and SYSUT5, the functions of which are described below.

Input/Output Devices

During translation and interpretation, devices are required for the following types of input/output:

Source program input

Printed listings

Output of interpretable code from translator (input to interpreter).

The ddnames associated with particular functions, and the permitted device types for each, are shown in Table 1.

Operating System Requirements and Facilities

The PL/I Checkout Compiler is a component of the IBM System/360 Operating System. The control programs that the compiler will run under, and the data management facilities which may be required, are detailed below.

Control Programs

The checkout 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.

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.

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

Time Sharing Option (TSO): this option must be available if the conversational features of the compiler are to be used. It provides:

The system commands that allow interaction with a PL/I program during translation and interpretation.

Table 1. Compiler Input/Output Devices

Function	ddname	Device Type	When Required
Source program input and, optionally, data	SYSCIN	DASD Magnetic tape Card reader Paper tape reader Terminal	See note below
Printed output	SYSPRINT	DASD Magnetic tape Printer Terminal	Always
Work file	SYSUT2	DASD	Always
Output from translator; input to interpreter	SYSUT3	DASD	If production of an object module is specified
Work file used to retain formatted source program after translation	SYSUT4	DASD	If a formatted source listing is specified
Work file for preprocessor	SYSUT5	DASD	If preprocessing is specified
Source program input and/or data	SYSIN	as SYSCIN	See note below
Output to linkage editor	SYSLIN	DASD Magnetic tape	If production of an object module is specified

Note: Source program input to the compiler may be on SYSCIN or SYSIN. If translation and interpretation are included in the same job step, however, the following conventions must be observed.

If SYSCIN is used for the source program, data for processing by the translated source program may either follow the source program on SYSCIN or it may be on SYSIN.

If SYSIN is used for the source program, data for processing by the translated source program may also be on SYSIN (following the source program), but SYSCIN must not be used.

- 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 System/360 Model 65s.

Data Management Facilities

Object programs compiled by the checkout compiler make use of the operating system data management facilities. These facilities include:

Basic Sequential Access Method (BSAM)

Queued Sequential Access Method (QSAM)

Basic Partitioned Access Method (BPAM)

Basic Indexed Sequential Access Method (BISAM)

Queued Indexed Sequential Access Method (QISAM)

Basic Direct Access Method (BDAM)

Telecommunication Access Method (TCAM)

Appendix A: Summary of Keywords

The following is a complete list of the PL/I and implementation-defined keywords implemented by the checkout compiler. Each of these keywords is described in detail in the publication IBM System/360 Operating System: PL/I Language Reference Manual (Preliminary), which also lists the keyword abbreviations.

Keyword	Use of Keyword
ABS(x)	Built-in function
ACOS(x)	Built-in function
%ACTIVATE	Preprocessor statement
ADD(x,y,p[,q])	Built-in function
ADDBUFF(n)	Option of ENVIRONMENT attribute
ADDR(x)	Built-in function
ALIGNED	Attribute
ALL	Option of PUT statement
ALL(x)	Built-in function
ALLOCATE	Statement
ALLOCATION(x)	Built-in function
ANY(x)	Built-in function
AREA	Condition
AREA[(size)]	Attribute
ASIN(x)	Built-in function
ATAN(x[,y])	Built-in function
ATAND(x[,y])	Built-in function
ATANH(x)	Built-in function
AUTOMATIC	Attribute
BACKWARDS	Attribute, option of OPEN statement
BASED[(locator-expression)]	Attribute
BEGIN	Statement
BINARY	Attribute
BINARY(x[,p[,q]])	Built-in function
BIT[(length)]	Attribute
BIT(expression[,size])	Built-in function
BLKSIZE(expression)	Option of ENVIRONMENT attribute
BOOL(x,y,w)	Built-in function
BUFFERED	Attribute
BUFFERS(n)	Option of ENVIRONMENT attribute
BUILTIN	Attribute
BY	Clause of DO statement
BY NAME	Option of the assignment statement
CALL	Statement, or option of INITIAL attribute
CEIL(x)	Built-in function
CHAR(expression[,size])	Built-in function
CHARACTER[(length)]	Attribute
CHECK[(name-list)]	Condition
CHECK	Statement
CLOSE	Statement
COBOL	Option of ENVIRONMENT attribute, or the OPTIONS option/attribute
COLUMN(w)	Format item
COMPLETION(event-name)	Built-in function, pseudo-variable
COMPLEX	Attribute
COMPLEX(a,b)	Built-in function, pseudo-variable
CONDITION(name)	Condition
CONJG(x)	Built-in function
CONNECTED	Attribute
CONSECUTIVE	Option of ENVIRONMENT attribute
CONTROLLED	Attribute
CONVERSION	Condition
COPY	Option of GET statement
COS(x)	Built-in function
COSD(x)	Built-in function

Keyword	Use of Keyword
COSH(x)	Built-in function
COUNT(file-expression)	Built-in function
CTLASA	Option of ENVIRONMENT attribute
CTL360	Option of ENVIRONMENT attribute
DATA	STREAM I/O transmission mode
DATAFIELD	Built-in function
DATE	Built-in function
%DEACTIVATE	Preprocessor statement
DECIMAL	Attribute
DECIMAL(x[,p[,q]])	Built-in function
DECLARE	Statement
%DECLARE	Preprocessor statement
DEFAULT	Statement
DEFINED	Attribute
DELAY(n)	Statement
DELETE	Statement
DESCRIPTORS	Option of DEFAULT statement
DIM(x,n)	Built-in function
DIRECT	Attribute
DISPLAY	Statement
DIVIDE(x,y,p[,q])	Built-in function
DO	Statement
%DO	Preprocessor statement
EDIT	STREAM I/O transmission mode
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 or statement
ENVIRONMENT	Attribute
ERF(x)	Built-in function
ERFC(x)	Built-in function
ERROR	Condition
EVENT	Attribute
EVENT(event-name)	Option of CALL, DELETE, DISPLAY, READ, REWRITE, and WRITE statements
EXCLUSIVE	Attribute
EXIT	Statement
EXP(x)	Built-in function
EXTERNAL	Attribute
F	Option of ENVIRONMENT attribute
FB	Option of ENVIRONMENT attribute
FBS	Option of ENVIRONMENT attribute
FILE	Attribute
FILE(file-expression)	Option of I/O statements
FINISH	Condition
FIXED	Attribute
FIXED(x[,p[,q]])	Built-in function
FIXEDOVERFLOW	Condition
FLOAT	Attribute
FLOAT(x[,p])	Built-in function
FLOOR(x)	Built-in function
FLOW	Statement, and option of PUT statement
FORMAT(format-list)	Statement
FORTRAN	Option of the OPTIONS option/attribute
FREE	Statement
FROM(variable)	Option of WRITE and REWRITE statements
GENERIC	Attribute
GENKEY	Option of ENVIRONMENT attribute
GET	Statement
GO TO	Statement
%GO TO	Preprocessor statement
HALT	Statement
HBOUND(x,n)	Built-in function

Keyword	Use of Keyword
HIGH(i)	Built-in function
IF	Statement
%IF	Preprocessor statement
IGNORE(n)	Option of READ statement
IMAG(x)	Built-in function, pseudo-variable
IN(area)	Option of ALLOCATE and FREE statements
%INCLUDE	Preprocessor statement
INDEX(string,config)	Built-in function
INDEXAREA[(size)]	Option of ENVIRONMENT attribute
INDEXED	Option of ENVIRONMENT attribute
INITIAL(expression)	Attribute
INPUT	Attribute, option of OPEN statement
INTER	Option of the OPTIONS option/attribute
INTERNAL	Attribute
INTO(variable)	Option of READ statement
IRREDUCIBLE	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
LENGTH(string)	Built-in function
LBOUND(x,n)	Built-in function
LEAVE	Option of ENVIRONMENT attribute
LIKE	Attribute
LINE(w)	Format item, option of PUT statement
LINENO(file-expression)	Built-in function
LINESIZE	Option of OPEN statement
LIST	STREAM I/O transmission mode
LOCATE	Statement
LOG(x)	Built-in function
LOG2(x)	Built-in function
LOG10(x)	Built-in function
LOW(i)	Built-in function
MAIN	Option of the 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,p[,q])	Built-in function
NAME(file-expression)	Condition
NCP(n)	Option of ENVIRONMENT attribute
NOCHECK[(name-list)]	Condition prefix identifier, statement
NOCONVERSION	Condition prefix identifier
NOFIXEDOVERFLOW	Condition prefix identifier
NOFLOW	Statement
NOLOCK	Option of READ statement
NOMAP	Option of the OPTIONS option/attribute
NOMAPIN	Option of the OPTIONS option/attribute
NOMAPOUT	Option of the OPTIONS option/attribute
NOOVERFLOW	Condition prefix identifier
NORESCAN	Option of %ACTIVATE statement
NOSIZE	Condition prefix identifier
NOSTRINGRANGE	Condition prefix identifier
NOSTRINGSIZE	Condition prefix identifier
NOSUBSCRIPTRANGE	Condition prefix identifier
NOUNDERFLOW	Condition prefix identifier
NOWRITE	Option of ENVIRONMENT attribute
NOZERODIVIDE	Condition prefix identifier
NULL	Built-in function
OFFSET(area-name)	Attribute
OFFSET(p,a)	Built-in function
ON	Statement
ONCHAR	Built-in function, pseudo-variable
ONCODE	Built-in function

Keyword	Use of Keyword
ONCOUNT	Built-in function
ONFILE	Built-in function
ONKEY	Built-in function
ONLOC	Built-in function
ONSOURCE	Built-in function, pseudo-variable
OPEN	Statement
OPTIONS(list)	Option of PROCEDURE statement, attribute
ORDER	Option of PROCEDURE and BEGIN statements
OUTPUT	Attribute, option of OPEN statement
OVERFLOW	Condition
PAGE	Format item, option of PUT statement
PAGESIZE(w)	Option of OPEN statement
PENDING(file-expression)	Condition
PICTURE	Attribute
POINTER	Attribute
POINTER(n,a)	Built-in function
POLY(a,x)	Built-in function
POSITION(expression)	Attribute
PRECISION(x,p[,q])	Built-in function
PRINT	Attribute, option of OPEN statement
PRIORITY(x)	Option of CALL statement
PRIORITY(task-name)	Built-in function
PROCEDURE	Statement
%PROCEDURE	Preprocessor statement
PROD(x)	Built-in function
PUT	Statement
RANGE	Option of DEFAULT statement
READ	Statement
REAL	Attribute
REAL(x)	Built-in function, pseudo-variable
RECORD	Attribute, option of OPEN statement
RECORD(file-expression)	Condition
RECSIZE(expression)	Option of ENVIRONMENT attribute
RECURSIVE	Option of PROCEDURE statement
REDUCIBLE	Attribute
REENTRANT	Option of PROCEDURE statement
REFER	Option of BASED attribute
REGIONAL(1|2|3)	Option of ENVIRONMENT attribute
REORDER	Option of PROCEDURE and BEGIN statements
REPEAT(string,i)	Built-in function
REPLY(c)	Option of DISPLAY statement
REREAD	Option of ENVIRONMENT attribute
RESCAN	Option of %ACTIVATE statement
RETURN	Statement
RETURNS	Attribute, option of PROCEDURE statement
REVERT	Statement
REWRITE	Statement
ROUND(exp,n)	Built-in function
SCALARVARYING	Option of ENVIRONMENT attribute
SEQUENTIAL	Attribute
SET(pointer-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[(x)]	Format item, option of GET and PUT statements
SNAP	Option of ON and PUT statements
SQRT(x)	Built-in function
STATIC	Attribute
STATUS[(event-name)]	Built-in function, pseudo-variable
STOP	Statement
STREAM	Attribute, option of OPEN statement
STRING(x)	Built-in function, pseudo-variable
STRING(string-name)	Option of GET and PUT statements
STRINGRANGE	Condition

Keyword	Use of Keyword
STRINGSIZE	Condition
iSUB	Dummy variable of DEFINED attribute
SUBSCRIPTRANGE	Condition
SUBSTR(string,i[,j])	Built-in function, pseudo-variable
SUM(x)	Built-in function
SYSIN	Name of standard system input file
SYSPRINT	Name of standard system output file
SYSTEM	Option of ON and DECLARE statements
TAN(x)	Built-in function
TAND(x)	Built-in function
TANH(x)	Built-in function
TASK	Attribute
TASK(task-name)	Option of CALL statement and OPTIONS option
THEN	Clause of IF statement
%THEN	Clause of %IF statement
TIME	Built-in function
TITLE(x)	Option of OPEN statement
TO	Clause of DO statement
TP(M|R)	Option of ENVIRONMENT attribute
TRANSIENT	Attribute
TRANSLATE(string, replacement,[position])	Built-in function
TRANSMIT(file-expression)	Condition
TRKOFL	Option of ENVIRONMENT attribute
TRUNC(x)	Built-in function
U	Option of ENVIRONMENT attribute
UNALIGNED	Attribute
UNBUFFERED	Attribute, option of OPEN statement
UNDEFINEDFILE(file-expression)	Condition
UNDERFLOW	Condition
UNLOCK	Statement
UNSPEC(x)	Built-in function, pseudo-variable
UPDATE	Attribute, option of OPEN statement
V	Option of ENVIRONMENT attribute
VALUE	Clause of DEFAULT statement
VARIABLE	Attribute
VARYING	Attribute
VB	Option of ENVIRONMENT attribute
VBS	Option of ENVIRONMENT attribute
VERIFY(string1,string2)	Built-in function
VS	Option of ENVIRONMENT attribute
WAIT	Statement
WHEN	Used in GENERIC declaration
WHILE	Clause of DO statement
WRITE	Statement
ZERODIVIDE	Condition

Appendix B: Compatibility with the PL/I (F) Compiler

Features of the PL/I Checkout Compiler implementation which are incompatible with the PL/I (F) Compiler implementation are listed alphabetically below. In every case, the description given is of the checkout 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 checkout compiler to ensure that they will return the same results.

ALLOCATION Built-In Function

The ALLOCATION built-in function returns a fixed-binary value giving the number of generations of the argument that exist in the current task.

Array Dimensions

The maximum number of dimensions in an array is 15.

Arrays of Pictures

Defined items in arrays of pictures must match the base elements exactly. The PL/I (F) Compiler requires only that the base elements should be pictures of character strings.

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 be also declared explicitly with the BUILTIN attribute, or contextually with a null argument list, for example: TIME( ).

DISPLAY Statement

The maximum length of the string to be displayed is 72 characters.

ENTRY Attribute

The maximum depth of nesting in a descriptor list in the ENTRY attribute is 2.

ENTRY Declarations

ENTRY declarations for internal procedures are not allowed.

Entry Names as Arguments

An entry name argument in parentheses, or an entry name without arguments, causes a dummy variable to be created; for the function to be invoked, a null argument list is required. (In the PL/I (F) Compiler an entry name argument in parentheses, or an entry name without arguments, is taken to be a function statement.)

Error Correction

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

Expressions in Parameter Extents

Expressions in parameter extents for variables that do not have the CONTROLLED attribute are not allowed.

File Parameters

A file parameter can be declared with the FILE attribute only; all other attributes are inherited from the argument. If additional attributes are declared, they are ignored and an informatory message is issued.

GENERIC Attribute

When using the GENERIC attribute, the entry must be declared explicitly and the keyword WHEN must also be specified.

KEY Option

If READ...KEY is used with a sequential data set and no record with the specified key exists in the data set, the KEY condition is raised and the file is positioned at the next record in ascending sequence.

KEYFROM Option

If an embedded key in a record is not identical to that specified in a WRITE...KEYFROM or LOCATE statement, the latter is moved into the record.

Labels on DECLARE Statements

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

Link-Editing of Object Modules

Object modules produced by the checkout compiler cannot be link-edited with object modules produced by the PL/I (F) Compiler.

ONKEY Built-In Function

When using REGIONAL(1) organization, the value returned by the ONKEY built-in function for a specification error consists of the last eight bytes of the source key, padded on the right with blanks if necessary. This value is returned for all I/O conditions other than ENDFILE, or other than ERROR raised as standard system action for an I/O condition.

In a RECORD I/O statement with the KEY or KEYFROM option, the ONKEY built-in function returns a null string when the ERROR condition is raised.

In a RECORD I/O statement referring to a KEYED file (but with no KEY, KEYFROM, or KEYTO option specified) the ONKEY built-in function returns the recorded key.

Picture Characters

The field of a drifting picture character will be blank when a zero is assigned to it.

Preprocessor Variables

A parameter descriptor list is not allowed in the declaration of a preprocessor variable with the ENTRY attribute.

PROD Built-In Function

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 that for fractional fixed-point arguments the result is in floating point.

Statements

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

Sterling Pictures

Sterling picture data is not implemented. Therefore the following picture characters are not allowed: G, H, M, P, 6, 7, 8.

Structures

The maximum depth of a structure is 15.

SUM Built-In Function

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.

Operating System Facilities

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

Facility	Entry-Point Name
Sort	PLISORT
Checkpoint/Restart	PLICKPT
Return Code	PLIRETC
Dump	PLIDUMP

The checkout compiler does not recognize the entry names used by the PL/I (F) Compiler, that is, IHESRTx, IHECKPT, IHESARC, IHEDUMx.

Appendix C: Complementary Use of the PL/I Checkout and Optimizing Compilers

The PL/I Checkout Compiler and the OS PL/I Optimizing Compiler (Program Product 5734-PL1) have been designed as a pair. They are compatible, and, because of their entirely different approaches, they offer many complementary advantages.

The primary aim of the checkout compiler is to reduce the time and effort spent on program checkout. The primary aim of the optimizing compiler is to increase system throughput when the program is in production use. Thus, used together, the two compilers can increase the efficiency of both programmer effort and machine usage.

Compatible Features

Source language: The same language is implemented by the two compilers, except that:

1. The conversational features of the PL/I language will be diagnosed for syntax errors, but otherwise ignored by the optimizing compiler.

2. The optimization feature of the PL/I language will be diagnosed for syntax errors, but otherwise ignored by the checkout compiler.

Object modules: The object modules produced by the two compilers can be link-edited and executed together as a single program.

Compiler options: The same option list can be specified for the two compilers. Any options which are not relevant to a particular compiler are recognized by it, but ignored.

Program results: The same results will be produced by a program irrespective of whether it is processed on the checkout or the optimizing compiler.

Execution trace: The same statement number trace can be performed with both compilers.

Complementary Features

Checkout compiler: The time and cost of developing a program is divided between programmer and machine. The checkout compiler aims to reduce substantially the programmer time and cost by providing a higher level of diagnostic information. This might sometimes result in a moderate increase in machine time and cost, caused by the interpretive style of processing necessary to achieve this higher level of diagnostic information.

Optimizing compiler: When a program has reached operational status, reduction of machine time and cost is of primary importance. The optimizing compiler reduces machine time and cost by producing a high performance object program. This high performance is achieved by the use of extensive optimization.

Review of Compiler Use

The following is a review of the advantages of the two compilers for program checkout and production use.

Program Checkout

The checkout compiler simplifies and speeds up program checkout in the following ways:

It produces more diagnostic information for syntax, global, and in particular, execution-time errors. In most cases, this will considerably reduce the time required to remove errors. In fact, in conversational mode, many programs will be fully debugged in one session at the terminal.

It can translate a source program several times faster than the optimizing compiler can compile it.

For some installations and programs, it may be advantageous to transfer to the optimizing compiler before checkout is complete, as it takes several times longer to interpret the text produced by the translation process of the checkout compiler than it does to execute the optimized code produced by the optimizing compiler. However, the following four compensating factors should be taken into consideration.

1. The checkout compiler will usually reduce the number of executions needed to debug the program. Thus, the difference in total machine time will not be as great as a strict comparison of performance would indicate.

2. The time required to perform an I/O operation is approximately the same irrespective of which compiler is used. Thus, the greater the number of I/O operations performed by the program, so the smaller the difference, proportionally, in elapsed time.

3. Link-editing or loading is not normally required before the interpretation of a module produced by the checkout compiler, but one of these is required before the execution of a module produced by the optimizing compiler. Thus, the time for this system overhead can be saved when using the checkout compiler.

4. The checkout compiler's ability to translate and execute several programs in one job step can result in a significant saving of system overhead time.

Production Use

The optimizing compiler will normally be used to generate the object program that will be used in a production environment. This is because the code produced by the optimizing compiler can be executed several times faster than the checkout compiler can interpret the text produced by the translation process.

However, in some cases, the number of executions may be so small that it does not justify recompiling a program after checkout on the checkout compiler.

Mixing Checkout-Compiled and Optimizing-Compiled Procedures

With programs constructed by the modular principle, it is likely that during development some external procedures will become suitable for optimizing earlier than others. These can be recompiled with the optimizing compiler and then executed with the procedures which are still in checkout-compiled form. The different procedures can be link-edited in the normal way.

Program control: The checkout compiler must be available to control the execution. Only one program can be executed in a job step. Either type of procedure can be the main procedure.

PL/I Resident Library: The user can avoid the inclusion of routines from the PL/I Resident Library (Program Product 5734-LM4) at execution time by specifying SYS1.PLIMIX in the SYSLIB DD statement for the linkage editor step. PLIMIX is a library of 'bootstrap' routines. They are included by the linkage editor with the optimized procedures instead of the full routines. They invoke the checkout compiler with a request to execute its corresponding routine.

Execution trace: An execution trace will be performed for procedures that have been compiled with the FLOW option.

Job control statements: Apart from the optional use of SYS1.PLIMIX, described above, the same job control statements are required as if the procedures were not of mixed types.

Index

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

READER'S COMMENT FORM

IBM System/360 Operating System
PL/I Checkout Compiler
General Information

Order No. GC33-0003

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

  As a reference source:

Very Good	Good	Fair	Poor	Very Poor
.....	.....	.....	.....	.....

  As a text:

Very Good	Good	Fair	Poor	Very Poor
.....	.....	.....	.....	.....

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . . .

This SRL manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note : Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Cut Along Line

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N.Y. 10601

Attention: Department 813 (HP)

fold fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM System/360 Printed in U.S.A. GC33-0003-0