IBM

VS FORTRAN Version 2

GC26-4219-4

General Information

Release 3

$$\sinh 3x = \sinh x(4\cosh^2 x - 1)$$
$$\sinh 4x = \sinh x \cosh x(8\cosh^2 x - 4)$$
$$\sinh 5x = \sinh x(1 - 12\cosh^2 x + 16\cosh^4 x)$$
$$\cosh 3x = \cosh x(4\cosh^2 x - 3)$$
$$\cosh 4x = 1 - 8\cosh^2 x + 8\cosh^4 x$$
$$\cosh 5x = \cosh x(5 - 20\cosh^2 x + 16\cosh^4 x)$$

$$x = \frac{2y-b-a}{b-a}$$

$$f(1+a^2)^{1/2}$$

VS FORTRAN Version 2

GC26-4219-4

General Information

Release 3

Fifth Edition (November 1987)

This is a major revision of, and makes obsolete, GC26-4219-3.

This edition applies to Release 3 of VS FORTRAN Version 2, Licensed Programs 5668-805 and 5668-806, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

The changes for this edition are summarized under "Summary of Changes" following the preface, "About This Manual." Because the technical changes in this edition are extensive and difficult to localize, they are not indicated by vertical bars in the left margin.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

About This Manual

This manual is intended to help managers and technical personnel evaluate and plan for using the VS FORTRAN Version 2 licensed program. (It is not intended to be used as a specifications manual.)

Summary of Changes

Release 3, March 1988

Major Changes to the Product

- Enhancements to the vector feature of VS FORTRAN Version 2
  - Automatic vectorization of user programs is improved by relaxing some restrictions on vectorizable source code. Specifically, VS FORTRAN Version 2 can now vectorize MAX and MIN intrinsic functions, COMPLEX compares, adjustably dimensioned arrays, and DO loops with unknown increments.
  - Ability to specify certain vector directives globally within a source program.
  - Addition of an option to generate the vector report in source order.
  - Ability to collect tuning information for vector source programs.
    - Ability to record compile-time statistics on vector length and stride and include these statistics in the vector report.
    - Ability to record and display run-time statistics on vector length and stride. Two new commands, VECSTAT and LISTVEC, have been added to Interactive Debug to support this function.
    - Enhancements to Interactive Debug to allow timing and sampling of DO loops.
    - Inclusion of vector feature messages in the on-line HELP function of Interactive Debug.
  - Simplification of the VECTOR compile-time option.
- Enhancements to the language capabilities of VS FORTRAN Version 2
  - Ability to specify the file or data-set name on the INCLUDE statement.
  - Ability to write comments on the same line as the code to which they refer.
  - Support for the DO WHILE structured programming construct.
  - Support for the ENDDO statement as the terminal statement of a DO loop.
  - Enhancements to the DO statement so that the label of the terminal statement is optional.
  - Support for statements extending to 99 continuation lines or a maximum of 6600 characters.
  - Implementation of IBM's Systems Application Architecture (SAA) FORTRAN definition; support for a flagger to indicate source language that does not conform to the language defined by SAA.
  - Support for the use of double-byte characters as character data in source programs, I/O, and for Interactive Debug input and output.

  - Support for the use of a comma to indicate the end of data in a formatted input field, thus eliminating the need for the user to insert leading or trailing zeros or blanks.

- Enhancements to the programming aids in VS FORTRAN Version 2
  - Enhancements to the intercompilation analysis function to detect conflicting and undefined arguments.
  - Support for the Data-In-Virtual facility of MVS/XA.
  - Ability to allocate certain commonly used files and data sets dynamically.
  - Enhancements to the Multitasking Facility to allow large amounts of data to be passed between parallel subroutines using a dynamic common block.
  - Support for named file I/O in parallel subroutines using the Multitasking Facility.
- Enhancements to the full screen functions of Interactive Debug

Major Changes to This Manual

- Documentation of the major product enhancements has been added.
- Information presented in Chapter 2 of previous editions has been merged into Chapter 1 of this edition.

Release 2, June 1987

Major Changes to the Product

- Support for 31-character symbolic names, which can include the underscore (_) character.
- The ability to detect incompatibilities between separately-compiled program units using an intercompilation analyzer. The ICA compile-time option invokes this analysis during compilation.
- Addition of the NONE keyword for the IMPLICIT statement.
- Enhancement of SDUMP when specified for programs vectorized at LEVEL(2), so that ISNs of vectorized statements and DO-loops appear in the object listing.
- The ability of run-time library error-handling routines to identify vectorized statements when a program interrupt occurs, and the ability under Interactive Debug to set breakpoints at vectorized statements.
- The ability, using the INQUIRE statement, to report file existence information based on the presence of the file on the storage medium.
- Addition of the OCSTATUS execution-time option to control checking of file existence during the execution of OPEN statements, and to control whether files are deleted from their storage media.
- Under MVS, addition of a data set and an optional DD statement to be used during execution for loading library modules and Interactive Debug.
- Under VM, the option of creating during installation a single VSF2LINK TXTLIB for use in link mode in place of VSF2LINK and VSF2FORT.

- The ability to sample CPU use within a program unit using Interactive Debug. The new commands LISTSAMP and ANNOTATE have been added to support this function.
- The ability to automatically allocate data sets for viewing in the Interactive Debug source window.

Major Changes to This Manual

Documentation of the major product enhancements has been added.

Release 1.1, September 1986

Major Changes to the Product

- Addition of vector directives, including compile-time option (DIRECTIVE) and installation-time option (IGNORE)
- Addition of NOIOINIT execution-time option
- Addition of support for VM/XA System Facility Release 2.0 (5664-169) operating system

Major Changes to This Manual

Documentation of the above product enhancements has been added.

Contents

Chapter 1. The Features of VS FORTRAN Version 2

FORTRAN is a programming language developed for applications involving mathematical computations and other manipulation of numeric data. This makes it especially well suited to scientific and engineering applications. Because FORTRAN is simple and easily learned and produces efficient code, it is widely used.

Over the years, IBM has developed a number of successful large-system FORTRAN products. VS FORTRAN Version 2 is the latest in this series. It offers the proven facilities of predecessor FORTRANs, plus a number of new features—all of which help programmers develop applications easily and efficiently, and use the power of IBM's latest large systems.

Some of the highlights of VS FORTRAN Version 2 are:

- Vector Support
- Extensive Language Capabilities
- Programming Aids
- Interactive Debugging
- Other Debugging Aids

The following pages describe these features and advantages.

The Vector Feature

The VS FORTRAN Version 2 compiler can produce programs that use the speed of the IBM 3090 Vector Facility. When instructed by a compile-time option, the compiler transforms eligible statements in DO loops into vector instructions. Such instructions can result in significantly faster processing.

For nested DO loops, the VS FORTRAN Version 2 compiler examines eight levels of loops beginning with the innermost level, and selects for conversion to vector instructions the eligible loop that will yield the fastest running time of the entire nest.

A simple recompilation allows most existing programs to take advantage of this new vector feature. There is no new source language to learn, and, in general, no recoding is necessary. (Both 77-level and 66-level programs can be vectorized.)

How Vector Processing Works

The characteristic feature of vector instructions is that they process multiple array elements. In effect, they process iterations of a DO loop in groups, and can therefore reduce run time dramatically. Figure 1 illustrates this reduction in run time.

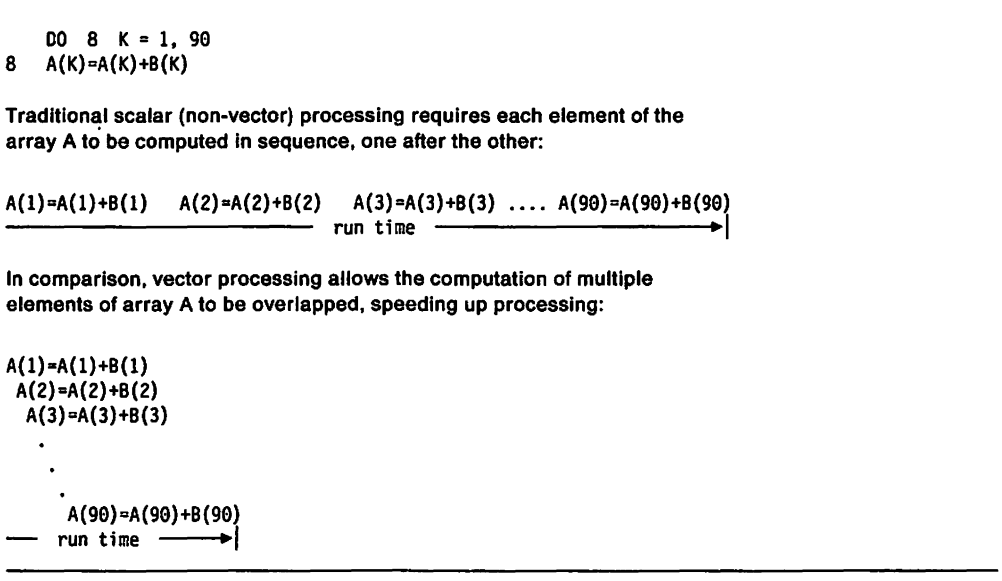

Figure 1. How Vector Processing Speeds Run Time

Compiler Versatility

VS FORTRAN Version 2 compiles source programs to produce either traditional nonvector code (usually called scalar code), or vector code. The compiler and library can operate on both scalar and vector processors. Programs that are candidates for vector processing can first be compiled in scalar mode and debugged on a scalar machine, and then recompiled in vector mode and transported to the vector machine for linking, loading, and run-time production.

Vector Language Considerations

Vectorization (the process of compiling DO loops into vector object code) is handled for the programmer by the VS FORTRAN Version 2 compiler. In general, no recoding is necessary to take advantage of vectorization.

VS FORTRAN Version 2 is able to process source statements with the following data types into vector instructions:

REAL*4
REAL*8
COMPLEX*8
COMPLEX*16
LOGICAL*4
INTEGER*4
INTEGER*2 (with some restrictions)

Vector versions of most of the VS FORTRAN Version 2 intrinsic mathematical functions are provided with the product. Thus, these mathematical calculations within DO loops are eligible to take advantage of the speed of vector processing. Because these vector mathematical functions have the same names as their scalar counterparts, programmers need make no coding changes. The compiler automatically selects the correct version.

Statements with the following data types and usage are ineligible for vectorization:

REAL*16
COMPLEX*32
CHARACTER
LOGICAL*1

Even if they satisfy the above data type requirements, not all statements are vectorizable. Statements and DO loops are ineligible if they contain:

Certain types of branches
I/O statements
External references (some intrinsic functions are eligible)
ASSIGN
ENTRY
PAUSE
RETURN
STOP
DO loop index variables of other than INTEGER*4 data type
Inner loops ineligible for any of the above reasons

The VS FORTRAN Version 2 compiler will examine each DO loop and select for vectorization only those statements that can be safely and effectively transformed. Statements that can not be safely vectorized, or would not run faster as vector instructions, are compiled into standard nonvector instructions.

Vector Directives

Directives are available which enable the user to provide additional information to affect vectorization performed by the compiler. Through these directives the user can:

- Tell the compiler what loop-count values are expected during processing, so that the compiler can better determine whether the loop can be advantageously vectorized.
- Indicate to the compiler that certain data relationships should not be considered in determining eligibility for vectorization. This makes certain ambiguous loops more likely to vectorize.
- Specify whether vector or scalar code is preferred.

The programmer can include individual directives for each DO loop or specify that certain directives be used globally within the source program.

Vector Report

In addition to producing vector code, the compiler can produce a vector report for the programmer. This lists the results of the compiler's source code analysis, and shows which loops have been vectorized. If requested, the compiler will generate compile-time statistics on the length and stride of each DO loop in the source program (both vectorized and non-vectorized); these statistics can be listed on the vector report. (Run-time statistics can also be generated by Interactive Debug.)

The report can be used to "tune" the source program; that is, the programmer can examine the source program for possible coding refinements. Certain sections that were not vectorized may be recoded to make them eligible for vectorization. The vector report can also be used to debug and to help understand the structure of the object code.

The report can be organized according to source order —just as the programmer coded the source— or according to the sequence in which the compiler placed the vectorized source code.

Vector Tuning

Interactive Debug provides the user with the ability to collect information needed to tune vector source programs for more efficient processing. IAD will record run-time statistics on the vector length and stride of each DO loop and display the results on the screen. The timing and sampling facilities provide information on the relative efficiency of each DO loop. By using the tuning information provided by IAD, the user can make decisions on how to modify the source code to improve the run-time performance of the program. For example, from a comparison of the compiler estimates of vector length and stride to the run-time statistics generated by IAD, the programmer can modify the vector directives in the program to more accurately reflect the vector lengths and strides.

Extensive Language Capabilities

VS FORTRAN Version 2 offers flexible language capabilities. It supports both FORTRAN 77 and FORTRAN 66 language standards. In addition, VS FORTRAN Version 2 implements IBM's Systems Application Architecture (SAA) FORTRAN definition.

Some of the highlights of the VS FORTRAN Version 2 language are described below. (For a complete listing of the elements of the language, see Appendix A, "The Language of VS FORTRAN Version 2" on page 25.)

Design Aids

VS FORTRAN Version 2 offers several facilities that ease program design:

The Block IF Statement: Lets programmers create blocks of procedural statements according to structured programming rules. Constructed with THEN, ELSE, and END IF, block IF allows one to easily identify each path of the IF and see its overall scope.

DO WHILE Statement: Lets programmers code DO loops according to the rules of structured programming. The DO WHILE statement processes groups of statements based on the evaluation of a logical expression, thus providing one of the basic structured programming constructs. Use of the DO WHILE and block IF statements makes programs easier to debug and maintain.

The INCLUDE Statement: Lets users insert a sequence of prewritten statements into a source program (at the location of the INCLUDE). This provides the ability to maintain a set sequence of FORTRAN statements in one location that need to be inserted at several places in a large program (for example, COMMON, DIMENSION, and EQUIVALENCE statements in multiple subprograms).

INCLUDE is also available in a conditional form (along with a compile-time option). This gives the ability to selectively include prewritten statements that can customize a single program for various applications, without requiring coding and maintaining a separate program for each.

Programmers can use either a ddname or the file or data-set name on the INCLUDE statement. Using the file or data-set name decreases the amount of job control language and file definition statements needed for the program.

Names for Constants: Lets users associate a name with a constant (through the PARAMETER statement), and then reference that constant by name in the remainder of the program unit.

End of Line Commentary: Lets programmers write comments on the same line as the code to which they pertain.

Free-Form Source Option

In VS FORTRAN Version 2, both free format and fixed format are available. Programmers can use whichever they prefer when coding new programs, and existing programs can always be recompiled without change to their source format.

Extended Statement Length

Programmers can write statements which extend up to 99 continuation lines or 6600 characters. By allowing longer statements in source programs, VS FORTRAN Version 2 facilitates the use of long FORMAT statements, initialization of large arrays, and the use of long names.

Comma in Formatted Input

Programmers can use a comma to indicate the end of data in a formatted input field. By using the comma, the programmer no longer has to insert leading or trailing blanks or zeros.

Standard Language Flaggers

VS FORTRAN Version 2 provides tools to assist programmers in creating source programmers that conform to the FORTRAN 77 standard or that adhere to the Systems Application Architecture FORTRAN definition. (Adherence to SAA FORTRAN helps to ensure portability of the source code.)

At the user's request, the compiler will flag source language that doesn't conform to the specified level of the FORTRAN 77 standard. Or, the user can request the compiler to identify source language used in the program that is not part of the SAA FORTRAN definition.

IMPLICIT NONE Statement

The keyword NONE, an extension of the IMPLICIT statement, enforces the good programming practice of assigning an explicit type to all variables, arrays, external functions, and named constants. IMPLICIT NONE precludes all default implicit typing except for the intrinsic functions. This extension provides a check on variable usage and identifies otherwise hard-to-find programming errors such as spelling mistakes.

Input/Output Capabilities

VS FORTRAN Version 2 offers three types of file access: sequential, direct, and keyed. This includes access to VSAM ESDS (entry sequenced data sets), RRDS (relative record data sets), and KSDS (key sequenced data sets).

Language features such as OPEN, CLOSE, INQUIRE, and IOSTAT offer programmers extensive control over I/O operations. In addition to the standard read/write operations, programmers can connect and disconnect files, retrieve useful information about files and records, and continue executing after I/O errors.

Character Data Type

VS FORTRAN Version 2 allows programmers to specify and process character data. Character data items (variables and array elements) can be up to 32767 characters in length.

For even more programming versatility, VS FORTRAN Version 2 allows users to identify and process portions (substrings) of character data items.

Long Symbolic Names

VS FORTRAN Version 2 allows symbolic names to be a maximum of 31 characters long and to include the underscore (_) character. The use of the underscore character improves the readability of long names.

Mathematical Subroutines

Mathematical subroutines supplied in VS FORTRAN Version 2 offer enhanced performance and accuracy over their VS FORTRAN Version 1 counterparts.

Many of the routines in VS FORTRAN Version 2 return a value that is never more than one low-order bit in error. Some always provide a correctly-rounded result. In contrast, the VS FORTRAN Version 1 routines are typically several low-order bits in error.

In addition to supplying this increased accuracy, many of the routines run faster than their Version 1 counterparts.

Bit String Manipulation

VS FORTRAN Version 2 supplies functions that allow programmers to view INTEGER*4 data as an ordered set of bits, where the set is a binary representation of an integer value. Programmers can perform logical operations on the bits, shift them left or right, set them, and test them.

Additional Supplied Subroutines and Functions

VS FORTRAN Version 2 also has built into it a great number of predefined functions and subroutines, in addition to the mathematical and bit manipulation routines described above. These functions and subroutines offer help in a wide variety of programming tasks. Among them are:

Character manipulation
Internal data conversion
Date and time recording
Error handling subroutines

Because these facilities are provided in the VS FORTRAN Version 2 product, users can achieve considerable programming power with minimal effort.

For a full list of the facilities provided, see Appendix B, "Supplied Functions" on page 29.

DBCS Support

Through its support for the double-byte character set, VS FORTRAN Version 2 allows programmers whose languages are ideographic (such as Japanese) to write programs that process some information in their own language.

Programmers can use double-byte characters in character constants, symbolic names, and comments within a source program. They can also include FORTRAN I/O statements for character data containing double-byte characters.

Programming Aids

VS FORTRAN Version 2 aids system and programmer performance in the following ways.

Optimized Object Code

Programmers can request three levels of object code optimization. Optimized object code usually requires less storage and results in faster processing. The levels are:

1. Register and branch optimization.
2. Full text and register optimization, to the extent allowed while retaining interruption localizing. (Interruption localizing prohibits moving code out of a loop if it might cause an interruption that would not occur without optimization.)
3. Full text and register optimization, without retaining interruption localizing. (Although this optimization might result in unanticipated interruptions, incorrect answers will not be generated from a legal program.)

Reentrancy

VS FORTRAN Version 2 reduces use of main storage by using reentrancy in three different ways:

1. The compiler itself is reentrant.
2. The compiler can generate reentrant object code.
3. Many of the library routines are reentrant.

Reentrancy means that many people can use the compiler or run an application at the same time, and a single copy of the compiler, program, or program part serves them all. A separate copy for each user is not necessary.

Similarly, when different applications running at the same time need to use a reentrant FORTRAN library routine, a single copy of that routine can be shared. Each concurrent application does not need its own copy.

Reentrant application routines need not be shared among different users to produce savings. Because the shareable parts of infrequently-used reentrant programs need not be loaded unless they are actually used, main storage requirements may be reduced.

Dynamic Loading of Library Routines

Users can choose to have the VS FORTRAN Version 2 library routines linked with their object modules, or loaded dynamically at run time. Dynamic loading reduces auxiliary storage requirements for load modules, speeds link-editing, and—in an XA environment—allows many library routines to reside above the 16-megabyte line or to be shared by being placed in the extended link pack area.

Multiple System Support

VS FORTRAN Version 2 runs on both MVS and VM systems. (For a complete list, see "Software" on page 15.) This includes the XA environment, with its large amounts of virtual storage.

A VS FORTRAN Version 2 program can be compiled on one supported operating system and then link-edited or loaded and run on another supported system. This is convenient if an installation has central development systems and separate production systems.

Dynamic Common

The dynamic common (DC) compiler option defines the names of common blocks to be allocated at run time. This allows specification of very large common blocks that can reside in the additional storage space available in an XA environment. This also reduces storage requirements by allocating only those common blocks that are in subprograms actually referenced during processing.

Upward Compatibility

In general, VS FORTRAN Version 2 is compatible with Version 1 and with earlier IBM large-system FORTRANs. For details, see "Compatibility" on page 17.

Data-In-Virtual Support

VS FORTRAN Version 2 allows programmers to make use of the data-in-virtual facility on MVS/XA.

The data-in-virtual facility reduces the amount of overhead inherent in processing large amounts of data through traditional record-oriented access methods. By invoking a series of VS FORTRAN Version 2 callable routines, the user can access a VSAM linear data set, map it to a dynamic common, and process the data set as if it were a large array.

Intercompilation Analysis

By using the intercompilation analysis function, users can significantly reduce development time by detecting incompatibilities between program units before attempting to debug an application. This function detects conflicting external names, provides usage information for common blocks, and checks for disagreement between actual arguments and dummy arguments and for undefined arguments.

When the ICA(UPDATE) compiler option is in effect, VS FORTRAN Version 2 builds a file of information obtained from compilations. The user can request the compiler to reference and update this file when compiling individual program units to check the interfaces between currently-compiled and previously compiled program units.

Dynamic File Allocation

For certain types of files and data sets, VS FORTRAN Version 2 will allocate a file or data set—associate a file and device with a FORTRAN program— dynamically. This means that the user can allocate files or data sets as they are required by the program, rather than at the time the program is loaded into storage. Dynamic file allocation reduces the need for file definition and job control statements for certain common types of files or data sets.

The OPEN and INQUIRE statements have been extended to allow the programmer to specify a CMS file identifier or MVS data set name, rather than a ddname. A new callable utility, FILEINF, allows the programmer to optionally specify attributes of the file. If the programmer chooses not to specify the attributes of the file or data set, VS FORTRAN Version 2 provides default attributes through a default attribute table which can be changed at customization time. VS FORTRAN Version 2 will build a file definition statement for the file or data set based on the specified or default attributes.

Multitasking Facility

VS FORTRAN Version 2 allows MVS users to take advantage of the improved run-time performance available on tightly-coupled multiprocessors and attached-processor systems. This multitasking facility lets a single VS FORTRAN Version 2 application program use several processors in a multiprocessing system simultaneously.

The programmer writes multiple subroutines to handle areas of code that can be run independently. Each subroutine can then be executed simultaneously on a separate processor.

For example, if a program had a large array of 1000 elements, and the calculation to be performed on each element is independent of the results of the others, the programmer could write four subroutines that each process a portion of the array. On request, VS FORTRAN Version 2 would run the subroutines simultaneously, and considerably reduce elapsed time. With each subroutine processing 250 elements, the 1000-element array could be processed three-to-four times as quickly.

The multitasking facility of VS FORTRAN Version 2 handles the dispatching of the subroutines, so that the programmer need have no special knowledge of the operating system or of the coupled processors. The programmer controls the subroutines through normal FORTRAN CALL statements. (Such subroutines are not available to Interactive Debug.)

Because VS FORTRAN Version 2 multitasking is simple to use, it can be quickly designed into new applications, and also easily introduced into existing ones. Large amounts of data can be shared between the main task program and the parallel subroutines by using a dynamic common block; each subroutine can perform I/O functions for named files. With little coding effort, it allows users to take advantage of the power of two-way and four-way multiprocessing systems, and considerably reduce the run-time of certain types of programs. It is especially valuable for running large computation-intensive programs that operate on different data independently, and can run on dedicated or otherwise-idle machines.

Interactive Debug

Interactive Debug, a component of the VS FORTRAN Version 2 product, is a flexible, efficient tool for monitoring program processing. Use of Interactive Debug can increase programmer productivity during the development cycle by expediting the debugging and tuning of programs. It helps users find program bugs by allowing them to simultaneously view the source program listing and control program processing. They can debug programs in a convenient, conversational manner, monitoring the program's activities as it runs.

The programmer can:

- Stop and restart the program at selected points
- Examine and change values of variables, arrays, and array elements
- Trace program transfers
- Track processing frequency of statements
- Skip processing of sections of code
- Control the action taken for run-time errors
- Locate errors, repair the problem, and continue debugging before recompiling
- Identify the parts of the program that use the most CPU time

- Record and display run-time statistics on vector length and stride for each DO loop in the program

For a full list of all of the Interactive Debug commands, and a description of the functions they provide, see Appendix D, "Commands for Interactive Debugging."

Full use of Interactive Debug requires ISPF (IBM Interactive System Product Facility) Version 2 and ISPF/PDF (IBM Program Development Facility). For details, see Chapter 2, "Programming Requirements and Support" on page 15.

Source Listing and Display Animation

VS FORTRAN Version 2 Interactive Debug offers full screen support. Users can perform debugging on one part of the screen, while watching their program run on the other part.

When the program is processing, its source listing can be automatically scrolled through a portion (a window) of the screen, while highlighting the currently-processing line (color, blinking, and intensification may be used for highlighting). The effect is an animated picture of program processing. The speed of processing can also be reduced, so that a slow-motion version can be monitored. Furthermore, because the user can halt processing where desired, a stop-action or freeze-frame ability is also provided.

Because the user's source listing is moving in tandem with program processing, it is easier than ever to understand what is happening in the program, and to see the connection between debugging output and any specific source statement.

Additional Interactive Debug Features

In addition to above functions, VS FORTRAN Version 2 Interactive Debug offers additional capabilities for versatile program debugging.

An easy invocation procedure:
The user simply specifies the DEBUG option when invoking a program to be debugged. This reduces the need for planning, and also allows the debugging of previously-compiled programs with minimal effort.

Selective debugging of program units:
An optional run-time control file lets the user specify which program units, and portion of program units, to be debugged. Code not requiring examination can be run without the overhead of run-time debugging.

Online help information:
Information is provided for all Interactive Debug commands and functions, and for all vector feature messages. By entering HELP (or using the equivalent PF key), the user will be presented with the first of a set of screens containing explanatory information.

Menu screens:
The user is provided with various menus on the screen, which allow for an easy selection of a choice of action. Menus are available for the HELP facility, for the screen design facility (choices of color, highlighting, and so forth), and for source listing identification.

Program Sampling:
Interactive Debug enables users to identify the sections of the program that take the most CPU time to process. Sampling information is collected at regular intervals for each statement of every debuggable program unit, and for each DO loop in a program unit. Relative distributions are obtained for various program sections. With this knowledge, programmers can concentrate improvement efforts on the areas that will have the most effect on performance.

Timing:
Users can measure the CPU run time of the program units, or of the DO loops within a program unit, being monitored. Users can then judge the relative efficiency of performance improvements by comparing timing data collected before and after they are implemented.

Manipulation of external files while debugging:
Commands that are similar to FORTRAN I/O statements (for example, ENDFILE, BACKSPACE, CLOSE, and REWIND) allow a user to manipulate the program's external sequential files. Also, while remaining in debug mode, it is possible to browse or edit these files.

Ability to issue system commands while debugging:
Without terminating the debug session, the user can issue commands at the system level.

Support for the double-byte character set:
Interactive Debug allows programmers to do run-time debugging on a program unit that contains double-byte characters in the source, or that operates on double-byte data. The programmer can view both double-byte and EBCDIC characters on the Interactive Debug panels. Double-byte characters can be used in input entered on the command line of the main debugging panel or on the line-mode command entry.

Logging:
Interactive Debug allows users to place the output from some of the commands in the print data set for later examination. All debug sessions can be logged in a data set. This data set can subsequently be used as input to Interactive Debug to recreate a previous debugging session.

Debugging of vectorized and optimized code: Programs compiled with the VECTOR option or at any optimization level can be debugged (with restrictions).

Batch-mode support:
Debugging sessions can be run in batch mode, by creating an input file of debugging commands. Debugging output will be placed in a file for later examination.

Other Debugging Aids

In addition to the Interactive Debug component, VS FORTRAN Version 2 offers the following analysis tools:

Extensive Diagnostics

Both the compiler and the run-time library provide diagnostic information.

Compile-Time Messages explain the cause of the error, indicate its severity, and identify its location (if known). Both error messages and the statements in error can be displayed at the programmer's terminal. Messages can also be printed on the source listing, either grouped separately or inserted inline at the locations of the errors.

MAP and XREF output on the compilation listing can be viewed at the terminal (in 72-column format) if so desired. One feature of XREF is that it will identify any variables that have been referenced but not initialized. XREF can also provide SET and FETCH information for each variable reference.

Run-Time Messages not only include an explanation, but identify the offset and routine of the erroneous statement, and can also identify the line number or internal statement number (ISN). In addition, the contents of the program status word (PSW) are printed in the case of program interrupts and abends (plus register contents for abends).

Other Run-Time Information can be obtained by calling the many service subroutines supplied with VS FORTRAN Version 2 (for example, OVERFL and DVCHK). The product supplies additional subroutines that allow a program to dynamically control its handling of errors.

Static Debug

VS FORTRAN Version 2 programmers can specify debugging packets at the beginning of the source program. Using these debugging packets, they can:

- Check the validity of array subscripts
- Trace the order of processing of all or part of the program
- Display array or variable values each time they change during program processing, or whenever desired

Additional Tools

Should a program abnormally terminate, VS FORTRAN Version 2 automatically provides two diagnostic aids:

- A traceback map that shows the sequence of called routines up to that point
- A symbolic dump of the program's variables and array elements

The program can also print a traceback map or a dump at any time during processing by calling a service routine provided by VS FORTRAN Version 2.

Chapter 2. Programming Requirements and Support

Hardware

To run programs compiled with the vectorization option specified, the IBM 3090 with the Vector Facility is required. All VS FORTRAN Version 2 programs can be compiled, and all scalar programs will run, on any IBM System/370 43xx or 30xx processor, where supported by the operating systems listed under "Software."

The processor must have sufficient real storage to meet the combined storage requirements of the host operating system and access methods used. Terminal support is provided by the host subsystem.

On MVS, VS FORTRAN Version 2 can use any I/O devices supported by the BSAM, BDAM, or VSAM access methods. On VM, VS FORTRAN Version 2 can use any I/O device supported by OS simulation under CMS.

Software

VS FORTRAN Version 2 supports vector and scalar compilation, and scalar processing, under:

- MVS/System Product Version 1 (5740-XYN or 5740-XYS) —all current releases, with or without TSO/E (5665-285)
- MVS/XA: MVS/System Product Version 2 (5665-291 or 5740-XC6) —all current releases and MVS/XA DFP Version 2 (5665-XA2)—all current releases, with or without TSO/E (5665-285)[1]
- VM/System Product (5664-167)—Release 4 or later, with or without VM/SP HPO (5664-173) —Release 4 or later[2,4]
- VM/XA System Product (5664-308)—Release 1, with bimodal CMS[3]
- VM/XA System Facility (5664-169)—Release 2

VS FORTRAN Version 2 supports vector processing under:

- MVS/XA: MVS/SP Version 2 Release 1.3 Vector Facility Enhancement or Release 1.7 (5665-291 or 5740-XC6) or later, and MVS/XA DFP Version 2 (5665-XA2)—all current releases, with or without TSO/E (5665-285)[1]
- VM/System Product (5665-167)—Release 4 or later, with VM/SP HPO (5664-173)—Release 4.2, with Vector Facility Support[2,4]
- VM/XA System Facility (5664-169)—Release 2
- VM/XA System Product (5664-308)—Release 1, with bimodal CMS[3]

Notes

1 Use of Data-in-Virtual requires MVS/SP Version 2 Release 2 Vector Facility Enhancement and MVS/XA DFP Version 2 Release 3.

2 Use of the double-byte character set requires VM/System Product Release 5.

3 VS FORTRAN Version 2 Release 3 will support VM/XA System Product Release 1 with bimodal CMS concurrent with the availability of ISPF support

4 Processing of VSAM files under VM/SP, and under VM/XA in 370 compatibility mode, requires VSE/VSAM (5746-AM2) Release 1, 2, or 3

Interactive debugging requires TSO/E (on MVS) or CMS (on VM). Interactive debugging in full-screen mode also requires:

- Under MVS, ISPF Version 2 (5665-319), with or without ISPF/PDF Version 2 for MVS (5665-317). For enhanced full-screen functions, 5665-319 is required.
- Under VM, ISPF Version 2 for VM (5664-282), with or without ISPF/PDF Version 2 for VM (5664-285). For enhanced full-screen functions, 5664-282 is required.

In the requirements given above for interactive debugging in full-screen mode, the appropriate ISPF/PDF product must be selected in order to use the following capabilities:

PDF browse and edit facilities in split-screen mode
Automatic browse of the debug print file and log at session end
Start of debugging using Interactive Debug's foreground invocation panel

Virtual Storage

The compiler requires approximately 1770K bytes of virtual storage (below the 16-megabyte line) to process a typical 100-statement source program. This storage estimate is for the compiler and its work areas only, and does not include space for operating system overhead or other programs. Because VS FORTRAN Version 2 does no page fixing, the minimum real storage for initiating a job is the only real storage requirement. No auxiliary storage is required since no work files are used. Because the compiler is reentrant, a single copy can support multiple users.

The Interactive Debug component requires approximately 400K bytes to begin processing (plus the storage required to load the program that will be debugged). Interactive Debug also acquires additional dynamic storage during processing. The amount varies according to the program being debugged and the type and quantity of debugging commands issued.

Installation

To install and customize the VS FORTRAN Version 2 product under MVS, the System Modification Program (SMP or SMP/E) is required. VS FORTRAN Version 2 provides the System Modification Program control statements and sample installation jobs, and the customization jobs needed to install and customize the product.

For installing and customizing under VM, EXECs are provided as part of VS FORTRAN Version 2.

For customizing VS FORTRAN Version 2 on an XA system, Assembler H Version 2 is required.

Compatibility

In general, VS FORTRAN Version 2 is compatible with Version 1 and with earlier large-system IBM FORTRANs.

Source Program Compatibility

Valid source programs that compiled correctly with VS FORTRAN Version 1 will compile correctly with VS FORTRAN Version 2.

Users who want to convert source programs written in earlier IBM FORTRANs (FORTRAN G1, HX, or 66-level VS FORTRAN) to FORTRAN 77 can use the IBM FORTRAN Language Conversion Program (5668-864). The converted programs can be compiled at a FORTRAN 77 level by the VS FORTRAN Version 2 compiler.

With Version 2 Release 3, comment lines may not precede an @PROCESS statement that includes the following compiler options:

- DBCS
- SAA
- FIPS
- LANGLVL
- CHARLEN
- NAME
- VECTOR
- FREE | FIXED
- DIRECTIVE

Vector Considerations — Subscript Values and Array Bounds: The VS FORTRAN Version 2 Compiler assumes that subscripts remain inside array dimensions. Programs conforming to the FORTRAN 77 standard require that every subscript be within its corresponding dimension declaration. The FORTRAN 66 standard requires only that the array element finally selected reside within the boundary of the total array; in particular, subscripting such as the following is permitted by the FORTRAN 66 standard:

```
      REAL A(10,10)
      DO 1 I = 1,20
1        A(I,2) = A(I,1)
```

However, if vectorization is specified, the loop above will be vectorized with no check for array bounds exceeded. Thus, it is possible that a program that runs correctly in scalar mode will not run correctly when vectorized if it does not conform to the FORTRAN 77 standard when referencing elements in an array.

Input/Output Considerations: For programs compiled on VS FORTRAN Version 2 Release 1.1 or earlier, compatibility of input and output source statements may be affected by semantics changes. For more information, see "Input/Output Compatibility" on page 18.

Object Module Compatibility

Existing object modules compiled by VS FORTRAN Version 1 Release 3 or later, OS FORTRAN G1, and OS FORTRAN H Extended can be link-edited together with those compiled by VS FORTRAN Version 2. The only exception is the following: if the object modules were produced by the VS FORTRAN Version 1 compiler prior to Release 3, and if they are programs that pass character arguments to subprograms or are subprograms that receive character arguments, then they must be recompiled by the VS FORTRAN Version 2 compiler or Version 1 Release 3 or later compiler to be compatible.

Object modules may be affected by input/output semantics changes. See "Input/Output Compatibility" for more information.

Library Compatibility

The VS FORTRAN Version 2 Release 3 library must be used to create (that is, to link-edit) an executable load module from the object module generated by the VS FORTRAN Version 2 Release 3 compiler.

Input/Output Compatibility

Programs compiled on VS FORTRAN Version 2 Release 1.1 or earlier may be affected by an improvement in input/output statement semantics.

To assist in file existence verification, the semantics for OPEN, CLOSE, and INQUIRE have been changed, and the run-time options OCSTATUS and NOOCSTATUS have been added.

These changes allow the program to determine file existence on the basis of a file's presence in a storage medium, to coordinate file existence with OPEN statement processing, and to report and enforce file existence and connectedness properties.

Some considerations involving input/output compatibility under the new semantics are as follows:

- A preconnected file can be implicitly opened only once in a program, since the file loses its preconnection when a CLOSE statement is given or when another file is opened on the same unit.
- An OPEN statement cannot be issued for a currently open file except to change the BLANK specifier.
- INQUIRE can be used to determine the properties of a file that has never been opened.
- INQUIRE specifiers SEQUENTIAL, DIRECT, KEYED, FORMATTED, and UNFORMATTED will return values dependent on how the files could potentially be connected.
- If run-time option OCSTATUS is in effect:
  - File existence is checked for consistency with the OPEN statement specifiers STATUS='OLD' and STATUS='NEW'.
  - File deletion occurs when the CLOSE statement specifier STATUS='DELETE' is given (on devices which allow deletion).
- If run-time option NOOCSTATUS is in effect:

- File existence is not checked for consistency with the OPEN statement specifiers STATUS='OLD' and STATUS='NEW'.
- File deletion does not occur with the CLOSE statement specifier STATUS='DELETE'.

Load Module Compatibility

Existing VS FORTRAN Version 1 and VS FORTRAN Version 2 load modules are compatible with the VS FORTRAN Version 2 Release 3 library as follows:

- For Version 1 Releases 2, 3, and 3.1
  - Load modules using the MVS reentrant I/O library load module (IFYVRENT) will run if the VS FORTRAN Version 2 library is made available during processing. Load modules created with Version 1 prior to Release 2 that use the reentrant I/O library load module must be relinked with the VS FORTRAN Version 2 Release 3 library.
- For Version 1 Release 4 or later and Version 2 Release 1 or later
  - Load modules that ran in load mode will continue to run if the VS FORTRAN Version 2 library is made available during processing.
  - Load modules with nonshareable portions of object modules will run if the load modules that contain the corresponding shareable portions of those object modules are made available during processing. No relinking is necessary.
- For Version 1 all releases and Version 2 Release 1 and 1.1
  - Load module compatibility may be affected by input/output semantics changes. For more information, see "Input/Output Compatibility" on page 18.

Arithmetic Results Compatibility

Because of architectural differences between the vector and scalar hardware, bit-wise identical results generally occur but are not guaranteed between scalar processing and vector processing of programs compiled and run under VS FORTRAN Version 2. Bit-wise identical results can be obtained by using the VECTOR(NOREDUCTION) (or VECTOR(NOINTRINSIC) if the Version 1 library is used) compiler option; when this option is used, however, some vectorization may be inhibited.

The instructions used by the VS FORTRAN Version 2 Release 3 compiler to generate more efficient vectorized code may produce different results from the same program compiled with the Version 2 Release 2 compiler.

Data Set Compatibility

Data sets created by standard-conforming VS FORTRAN Version 1 programs can be used by VS FORTRAN Version 2 programs.

When a program is compiled with the LANGLVL(66) option, VS FORTRAN Version 2 provides support for all the file processing capabilities of IBM System/360 and System/370 FORTRAN IV products: sequential, direct, and asynchronous.

When a program is compiled with the LANGLVL(77) option, VS FORTRAN Version 2 provides support for VSAM ESDS (entry sequenced data set), RRDS (relative record data set), and KSDS (key sequenced data set), as well as all the same capabilities provided with the LANGLVL(66) option.

Interactive Debug Compatibility

VS FORTRAN Version 2 Interactive Debug is designed for use with programs compiled by VS FORTRAN Version 2, and by VS FORTRAN Version 1 Release 4 or later.

Programs compiled with releases subsequent to Version 1 Release 4 (including Version 2 releases) require link editing with the corresponding library release or a later library. For example, programs compiled on the Release 4.1 compiler would require link editing with the Release 4.1 library or a later library.

Batch-mode debugging is available only if the program is using the VS FORTRAN Version 2 library.

Program sampling is available only if the program is using the VS FORTRAN Version 2 Release 2 or later library.

DO loop analysis is available only to programs compiled on the Version 2 Release 3 compiler using the IVA suboption of the VECTOR option.

AFFON files for debugging jobs run on Release 2 or earlier must be modified to run on Release 3.

License

Separate licenses for the Compiler and Library and Interactive Debug (5668-806) or the Library only (5668-805) are required for each system on which the licensed program materials will be used.

Program Services

Central Service, including the IBM support center, will be available until discontinued by IBM upon twelve months' written notice.

Warranty

The VS FORTRAN Version 2 product will conform, when shipped to the Customer, to its Licensed Program Specifications which are in effect for it at that time, provided it is properly used in a Specified Operating Environment.

If the Customer believes there is a defect in the program such that it does not meet its Licensed Program Specifications, the Customer must notify IBM while Program Services are available for the program.

IBM does not warrant that the functions contained in the program will meet the Customer's requirements or will operate in the combinations which may be selected for use by the Customer, or that the operation of the program will be uninterrupted or error free or that all program defects will be corrected.

THE FOREGOING WARRANTIES ARE IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Licensed Program Specifications may be updated from time to time and such updates may constitute a change in specifications.

For Distributed Systems License Option (DSLO) Licenses, warranty service, if any, will be provided only through the Basic License location.

Following the discontinuance of all program services, the program will be distributed on an "As Is" basis without warranty of any kind either express or implied.

Any other documentation with respect to this licensed program, including any documentation referenced herein, is provided for reference purposes only and does not extend or modify these specifications.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Appendixes

Appendix A. The Language of VS FORTRAN Version 2

VS FORTRAN Version 2 is designed according to the specifications of the following external standards:

- *American National Standard Programming Language FORTRAN, ANSI X3.9-1978,* and *International Organization for Standardization ISO 1539-1980 Programming Languages--FORTRAN.* (These two standards are technically equivalent, and are referred to informally as the 77 level.)
- *American Standard FORTRAN, X3.9-1966,* and *International Organization for Standardization ISO R 1539-1972 Programming Languages--FORTRAN.* (These two standards are technically equivalent, and are referred to informally as the 66 level.)

With Release 3, VS FORTRAN Version 2 supports the Systems Application Architecture Common Programming Interface FORTRAN definition.

VS FORTRAN Version 2 not only supports SAA and both of the above standards, but also includes many IBM language extensions. The table in Figure 2 on page 26 lists the major elements of the VS FORTRAN Version 2 language. The table shows whether:

- An element is supported at the 77 and the 66 levels
- An element is standard (Std) or an IBM extension (Ext) to either the 77 or 66 standard,
- An element is included in the Systems Application Architecture FORTRAN definition.

Only high-level aspects of the language are shown. (For complete information, refer to the *Language and Library Reference*).

Statement or Feature	Use	In 77 Level?	In 66 Level?	In SAA?
Ampersand (&)	Allowed as special character	No	Yes - Ext	No
ASSIGN	Assigns GOTO targets	Yes - Std	Yes - Std	Yes
	Assigns FORMAT labels	Yes - Std	No	Yes
Assignment statements	Assign values to arithmetic and logical data items	Yes - Std	Yes - Std	Yes
	Assign values to character data items	Yes - Std	No	Yes
Asynchronous I/O	Read/write in asynchronous mode	Yes - Ext	Yes - Ext	No
AT	Specifies beginning of debugging packet	Yes - Ext	Yes - Ext	No
BACKSPACE	Repositions file at previous record	Yes - Std	Yes - Std	Yes
BLOCK DATA	Identifies a data subprogram	Yes - Std	Yes - Std	Yes
CALL	Transfers control to a subroutine	Yes - Std	Yes - Std	Yes
Character data type	Allows character (string) data	Yes - Std	No	Yes
CLOSE	Disconnects file from a program	Yes - Std	No	Yes
Columns 1 to 5	Can be non-blank on continuation line	Yes - Ext	Yes - Ext	No
COMMON	Defines storage shared between programs	Yes - Std	Yes - Std	Yes
	Allows both character and noncharacter data in one block	Yes - Std	No	Yes
Complex data type	Complex numbers of single precision	Yes - Std	Yes - Std	Yes
	Complex numbers of double and extended precision	Yes - Ext	Yes - Ext	Yes
CONTINUE	Nonoperational executable statement for programming convenience	Yes - Std	Yes - Std	Yes
Currency symbol ($)	Can be used in names	Yes - Ext	Yes - Ext	No
DATA	Initializes variables and array elements	Yes - Std	Yes - Std	Yes
	Initializes variables and arrays with implied DO loops if desired	Yes - Std	No	Yes
DEBUG and END DEBUG	Delimit the debugging packet portion of a program	Yes - Ext	Yes - Ext	No
DEFINE FILE	Specifies a direct-access file	No	Yes - Ext	No
DELETE	Deletes record from a KSDS file	Yes - Ext	No	No
DIMENSION	Defines arrays of up to three dimensions	Yes - Std	Yes - Std	Yes
	Defines arrays of up to seven dimensions	Yes - Std	Yes - Ext	Yes
	Defines arrays with adjustable size	Yes - Std	Yes - Std	Yes
	Defines arrays with explicit lower bounds (which can be positive or negative)	Yes - Ext	No	Yes
Direct-access I/O	Read/write by record number	Yes - Std	Yes - Ext	Yes
DISPLAY	Displays data within a debugging packet	Yes - Ext	Yes - Ext	No
DO	Gives a convenient way to program loops (using integer DO variables)	Yes - Std	Yes - Std	Yes
	Real and double precision DO variables are allowed; negative incrementation parameter is allowed	Yes - Std	No	Yes
DO WHILE	Initiates processing of program loops based on evaluation of a logical expression	Yes - Ext	No	No

Figure 2 (Part 1 of 3). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In 77 Level?	In 66 Level?	In SAA?
EJECT	Starts new page of source listing	Yes - Ext	Yes	No
END	Marks end of program unit	Yes - Std	Yes - Std	Yes
	Terminates program processing	Yes - Std	No	Yes
END DO	Terminates processing of a DO or DO WHILE loop	Yes - Ext	No	No
end of line commentary	The "!" indicates the beginning of a comment	Yes - Ext	Yes - Ext	No
ENDFILE	Writes end-of-file record	Yes - Std	Yes - Std	Yes
ENTRY	Specifies alternate entry points into subprograms	Yes - Std	Yes - Std	Yes
EQUIVALENCE	Defines shared storage	Yes - Std	Yes - Ext	Yes
	Can relate character and noncharacter data	Yes - Std	No	Yes
Explicit type statements	Define data types of specific variables	Yes - Std	Yes - Std	Yes
Expressions	Manipulate arithmetic, relational, or logical items, or other expressions	Yes - Std	Yes - Std	Yes
	Manipulate character items or arithmetic double precision or complex items	Yes - Std	No	Yes
EXTERNAL	Defines linked subprograms	Yes - Std	Yes - Std	Yes
FIND	Locates next input record	No	Yes - Ext	No
FORMAT	Defines record formats	Yes - Std	Yes - Std	Yes
	Character constants and run-time formats allowed	Yes - Std	No	Yes
Free-form source	Relaxes format rules for source program	Yes - Ext	Yes - Ext	No
FUNCTION	Identifies a function subprogram	Yes - Std	Yes - Std	Yes
GENERIC	Allows automatic function selection	No	Yes - Ext	No
GO TO	Specifies transfers of control	Yes - Std	Yes - Std	Yes
Hexadecimal constants	For initializing data values	Yes - Ext	Yes - Ext	No
Hollerith constants	For initializing integer variables	Yes - Ext	Yes - Std	No
	As arguments in CALL statements	No	Yes - Ext	No
	As character strings in FORMAT statements	Yes - Ext	Yes - Std	Yes
IF	Specifies alternate paths of processing, using arithmetic and logical IF versions	Yes - Std	Yes - Std	Yes
	Block IF version, using ELSE, ELSE IF, and END IF	Yes - Std	No	Yes
IMPLICIT	Types groups of variables	Yes - Std	Yes - Ext	Yes
INCLUDE	Copies prewritten source statements into program	Yes - Ext	No	Yes
Integer data type	Allows integer numbers	Yes - Std	Yes - Std	Yes
INQUIRE	Retrieves information about a file	Yes - Std	No	Yes
Internal files	Allow easy data conversion	Yes - Std	No	Yes
Intrinsic functions	Supply arithmetic and generic functions	Yes - Std	Yes - Std	Yes
	Supply character and bit functions	Yes - Std	No	Yes
INTRINSIC	Explicitly defines intrinsic functions	Yes - Std	No	Yes
I/O status indicator	Determine success of input/output statement	Yes - Std	No	Yes

Figure 2 (Part 2 of 3). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In 77 Level?	In 66 Level?	In SAA?
Keyed I/O	Read/write by record key value	Yes - Ext	No	No
Length fields	Optional specification for data types	Yes - Ext	Yes - Ext	Yes
List-directed I/O	Read/write formatted data without FORMAT statement	Yes - Std	Yes - Ext	Yes
'Literal constants'	Literal constants enclosed in apostrophes	Yes - Std	Yes - Ext	Yes
Logical data type	Allows true/false values	Yes - Std	Yes - Std	Yes
Mixed-mode expressions	Allow mixing of data types	Yes - Std	Yes - Ext	Yes
NAMELIST	Read/write referencing named list	Yes - Ext	Yes - Ext	No
OPEN	Connects files to a program; error routines can be specified	Yes - Std	No	Yes
PARAMETER	Establishes names for constants	Yes - Std	No	Yes
PAUSE	Suspends program processing temporarily	Yes - Std	Yes - Std	Yes
PRINT	Installation-dependent write statement	Yes - Std	Yes - Ext	Yes
PROGRAM	Names a main program	Yes - Std	No	Yes
PUNCH	Installation-dependent write statement	No	Yes - Ext	No
Quotation mark	Double quote (") allowed as special character	Yes - Ext	No	No
READ	Reads a record from a file	Yes - Std	Yes - Std	Yes
Real data type	Single precision floating-point numbers	Yes - Std	Yes - Std	Yes
	Double precision floating-point numbers	Yes - Std	Yes - Std	Yes
	Extended precision floating-point numbers	Yes - Ext	Yes - Ext	No
Real subscripts	Expressions with floating-point numbers can be used as subscripts	Yes - Ext	No	No
RETURN	Returns control to a calling program	Yes - Std	Yes - Std	Yes
REWIND	Repositions to beginning of file	Yes - Std	Yes - Std	Yes
REWRITE	Rewrites record in a KSDS file	Yes - Ext	No	No
SAVE	Saves values after a called program completes executing	Yes - Std	No	Yes
Sequential I/O	Read/write sequential files	Yes - Std	Yes - Std	Yes
Statement functions	Allow convenient programming of expressions	Yes - Std	Yes - Std	Yes
STOP	Terminates program processing	Yes - Std	Yes - Std	Yes
SUBROUTINE	Identifies a subroutine subprogram	Yes - Std	Yes - Std	Yes
Symbolic names	May be 31 characters long	Yes - Ext	No	Yes
TRACE ON/OFF	Traces specific portions of a program	Yes - Ext	Yes - Ext	No
Underscore character (_)	Can be used in names	Yes - Ext	No	Yes
VSAM I/O	Supports ESDS, RRDS, and KSDS files	Yes - Ext	No	No
WRITE	Writes a record into a file	Yes - Std	Yes - Std	Yes

Figure 2 (Part 3 of 3). Major Elements of the VS FORTRAN Version 2 Language

Appendix B. Supplied Functions

The VS FORTRAN Version 2 program product supplies you with a great number of predefined, built-in functions and subroutines. These offer you help in a wide variety of programming tasks—mathematical calculation, character and bit manipulations, error-handling operations, and others.

This appendix lists the provided functions and subroutines.

Mathematical Functions

Most of the mathematical functions are available in multiple versions to support a variety of data types and precisions. For example, the square root is available in six versions, so that both real and complex numbers (each in single, double, and extended precision) can be computed.

General Type of Function	Specific Function	Function Name
Logarithmic	Natural logarithm	LOG, ALOG, DLOG, QLOG, CLOG, CDLOG, CQLOG
	Common logarithm	LOG10, ALOG10, DLOG10, QLOG10
Exponential	*e* raised to the power X	EXP, DEXP, QEXP, CEXP, CDEXP, CQEXP
Square Root	Principal square root	SQRT, DSQRT, QSQRT, CSQRT, CDSQRT, CQSQRT
Trigonometric	Sine	SIN, DSIN, QSIN, CSIN, CDSIN, CQSIN
	Cosine	COS, DCOS, QCOS, CCOS, CDCOS, CQCOS
	Tangent	TAN, DTAN, QTAN
	Cotangent	COTAN, DCOTAN, QCOTAN
	Arcsine	ASIN, DASIN, QARSIN
	Arccosine	ACOS, DACOS, QARCOS
	Arctangent	ATAN, DATAN, QATAN, ATAN2, DATAN2, QATAN2
Hyperbolic	Hyperbolic sine	SINH, DSINH, QSINH
	Hyperbolic cosine	COSH, DCOSH, QCOSH
	Hyperbolic tangent	TANH, DTANH, QTANH

Figure 3 (Part 1 of 2). Mathematical Functions Supplied

General Type of Function	Specific Function	Function Name
Miscellaneous Mathematical	Absolute value	ABS, IABS, DABS, QABS, CABS, CDABS, CQABS
	Error function for normal curve	ERF, DERF, QERF
	Error function complement for normal curve	ERFC, DERFC, QERFC
	Gamma function	GAMMA, ALGAMA, DGAMMA, DLGAMA
	Imaginary part of complex argument	IMAG, AIMAG, DIMAG, QIMAG
	Conjugate of a complex argument	CONJG, DCONJG, QCONJG
	Truncation of a real number	AINT, DINT, QINT
	Nearest whole number	ANINT, DNINT
	Nearest integer	NINT, IDNINT
	Remainder	MOD, AMOD, DMOD, QMOD
	Transfer of sign	ISIGN, SIGN, DSIGN, QSIGN
	Positive difference	IDIM, DIM, DDIM, QDIM
	Double precision product	DPROD
	Largest value	MAX, MAX0, AMAX0, MAX1, AMAX1, DMAX1, QMAX1
	Smallest value	MIN, MIN0, AMIN0, MIN1, AMIN1, DMIN1, QMIN1

Figure 3 (Part 2 of 2). Mathematical Functions Supplied

Additional Functions

General Type of Function	Specific Function	Function Name
Character Manipulation	Convert character to integer value of position in collating sequence	ICHAR
	Convert integer to corresponding character in collating sequence	CHAR
	Length of character item	LEN
	Location of character string	INDEX
	Lexically greater than or equal to	LGE
	Lexically greater than	LGT
	Lexically less than or equal to	LLE
	Lexically less than	LLT
Bit Manipulation	Logical AND	IAND
	Logical OR	IOR
	Logical exclusive OR	IEOR
	Logical complement	NOT
	Left or right shift	ISHFT
	Bit test	BTEST
	Bit set to 1	IBSET
	Bit set to 0	IBCLR
Internal Data Conversion	To integer	INT, IFIX, IDINT, IQINT, HFIX
	To real	REAL, FLOAT, SNGL, SNGLQ, DREAL, QREAL, DFLOAT, QFLOAT
	To single precision	SNGL, SNGLQ
	To double precision	DFLOAT, DBLE, DBLEQ
	To extended precision	QEXT, QEXTD, QFLOAT
	To complex	CMPLX, DCMPLX, QCMPLX

Figure 4. Additional Functions Supplied

Service Subroutines

Service subroutines give VS FORTRAN Version 2 programmers control over certain mathematical exceptions and over program termination when unusual conditions occur. In addition, they provide date and time information, allow the programmer to manipulate double-byte character strings, and to specify certain attributes for dynamically allocated files or data sets. (All these prewritten subroutines are accessed through a CALL statement.)

The following subroutines are provided:

ASSIGNM	Moves a character string that contains both single- and double-byte characters to a character variable, substring, or array element.
CDUMP|CPDUMP	Dynamically dumps a specified area of storage.
CLOCK|CLOCKX	Provides the current time.
DATIM|DATIMX	Provides the current day and time.
DVCHK	Tests for a divide-check exception and returns a value indicating the condition that exists.
DUMP|PDUMP	Dumps a specified area of storage and either terminates (DUMP) or continues processing (PDUMP).
EXIT	Terminates processing by returning control to the operating system.
FILEINF	Sets up file characteristics before processing an OPEN or INQUIRE statement.
OVERFL	Tests for an exponent overflow or underflow exception and returns a value indicating the condition that exists.
SDUMP	Provides a symbolic dump of variables.
SYSABD|SYSABN	Causes abnormal termination of the job, either with or without a dump.
SYSRCS	Sets the return code to a value.
SYSRCT	Examines the current return code value.
SYSRCX	Halts the program (equivalent to EXIT), and passes the current return code to the operating system.

Error-Handling Subroutines

During program processing, VS FORTRAN Version 2 issues messages if the following errors occur:

Operation
Fixed-point divide
Decimal divide
Floating-point divide
Exponent overflow
Exponent underflow
Specification exception for boundary alignment (vector)
Unnormalized operand (vector)

VS FORTRAN Version 2 also issues messages if it detects various abnormal conditions and if I/O errors occur.

Through the run-time error-handling subroutines provided in, VS FORTRAN Version 2 you can dynamically control:

The number of times an error can occur before the program is terminated
The maximum number of times a message is printed
Whether a traceback map is to be printed with the message
Whether a user error routine is to be called

(During installation, you can set the default values for these controls in the error option table.)

Dynamic control is available by calling the following predefined subroutines:

ERRSAV Obtains a copy of an error option table entry.

ERRSTR Stores an updated entry in the error option table.

ERRSET Changes parameters in the option table (for example, the number of errors permitted or number of messages to be printed).

ERRTRA Requests a trace of program processing.

ERRMON Prints an error message.

Appendix C. Options

Compile-Time Options

VS FORTRAN Version 2 provides you with many options for controlling the operation and output of the compiler.

These options are:

AUTODBL (*value*)
Requests that the precision of floating-point items in the program be increased (single to double, double to extended). Padding can also be requested, to preserve the size relationship of any items sharing storage.

CHARLEN (*number*)
Specifies the maximum length permitted for any character variable, character array element, or character function.

CI (*number1,number2,...*)
Specifies the identification numbers of the INCLUDEs to be processed.

DBCS | NODBCS
Indicates that the source file may contain double-byte characters.

DC (*name1,name2,...*)
Defines the names of common blocks to be allocated at run time. This option allows specification of large common blocks that can reside in the additional storage space available in an XA environment.

DECK | NODECK
Specifies whether the object module in card image format is to be produced.

DIRECTIVE (*trigger-constant*) **| NODIRECTIVE [**(*trigger-constant*)**]**
Specifies a string which permits processing of selected comments as directive statements. The DIRECTIVE option can be specified only in an @PROCESS statement.

FIPS (S | F) | NOFIPS
Specifies whether standard language flagging is to be performed, and, if it is, the standard language flagging level (subset or full).

FLAG (I | W | E | S)
Specifies the level of diagnostic messages to be written.

FREE | FIXED
Denotes whether the input source program is in free format or in fixed format.

GOSTMT | NOGOSTMT
Specifies whether internal sequence numbers (for traceback purposes) are to be generated for a call sequence to a subprogram.

ICA (USE(*name1,name2,...***) UPDATE | UPD(***name***) MXREF | NOMXREF CLEN | NOCLEN CVAR | NOCVAR MSG(NEW | NONE | ALL)) | NOICA**
Specifies whether intercompilation analysis is to take place. Only NOICA can be specified in an @PROCESS statement.

IL (DIM | NODIM)
Specifies whether the code for adjustably-dimensioned arrays is to be placed in-line or done via library call.

LANGLVL (66 | 77)
Specifies the language level in which the input source program is written.

LINECOUNT (*number*)
Specifies the maximum number of lines on each page of the printed source listing.

LIST | NOLIST
Specifies whether the object module listing is to be produced.

MAP | NOMAP
Specifies whether a table of symbolic names and statement labels is to be written.

NAME (*name*)
Specifies, for FORTRAN 66 programs only, the name to be given to a main program.

OBJECT | NOOBJECT
Specifies whether the object module is to be produced.

OPTIMIZE (0 | 1 | 2 | 3) | NOOPTIMIZE
Specifies the optimizing level to be used during compilation.

RENT | NORENT
Specifies whether the compiler should produce reentrant object code.

SAA | NOSAA
Specifies whether the compiler should flag all language elements in a source program which are not part of Systems Application Architecture.

SDUMP (ISN | SEQ) | NOSDUMP
Specifies whether symbol table information will be printed if the program executes a CALL SDUMP statement or abnormally terminates at run time. Also specifies whether internal statement numbers or sequence numbers are to be generated for use by Interactive Debug.

SOURCE | NOSOURCE
Specifies whether the source listing is to be produced.

SRCFLG | NOSRCFLG
Controls the inserting of error messages in the source listing.

SXM | NOSXM
Specifies whether the MAP and XREF output should be 72 columns wide, to allow convenient viewing on a terminal.

SYM | NOSYM
Invokes the production of SYM cards in the object text file. These cards contain location information for variables within a FORTRAN program.

TERMINAL | NOTERMINAL
Specifies whether error messages and compiler diagnostics are to be written on the output data set and whether a summary of error messages is to be printed.

TEST | NOTEST
Generates calls to VS FORTRAN Version 2 Interactive Debug at each statement boundary.

TRMFLG | NOTRMFLG
Causes the FORTRAN source statement in error (if applicable) and its associated error messages (formatted for the terminal being used) to be displayed at the terminal.

VECTOR (REPORT(LIST | XLIST | TERM | SOURCE |STAT) | NOREPORT INTRINSIC | NOINTRINSIC IVA | NOIVA REDUCTION | NOREDUCTION SIZE(ANY | LOCAL | *number*)) | NOVECTOR
Invokes the vectorization process, which produces programs that can exploit the speed of the IBM 3090 Vector Facility. The user can control the scope of vectorization to be done, the type of report desired, the vector section size to be used, and whether statistics should be generated on the vector lengths and strides and included on the vector report.

XREF | NOXREF
Specifies whether a cross-reference listing is to be produced. The cross-reference listing includes all variables and labels used in the source program.

Run-Time Options

Options are available to allow you to control certain aspects of a program's processing:

ABSDUMP | NOABSDUMP
Specifies whether, when an abnormal termination is scheduled, the post-ABEND symbolic dump information is to be printed.

AUTOTASK (*loadmodname,numtasks*) | NOAUTOTASK
Specifies control information when using the multitasking facility (MTF).

DEBUG | NODEBUG
Specifies whether VS FORTRAN Version 2 Interactive Debug is to be invoked. Using this component, a programmer can debug a program as it executes in either a CMS or a TSO environment.

DEBUNIT (*unitnumbers*) | NODEBUNIT
To avoid conflict when using interactive debug, identifies any terminal devices the program will use for its own I/O.

IOINIT | NOIOINIT
Specifies whether the error message unit is to be opened or not during initialization of the run-time environment.

OCSTATUS | NOOCSTATUS
Controls whether file existence will be checked for consistency with the OPEN specifiers STATUS='OLD' and STATUS='NEW' and, for files not dynamically allocated by the VS FORTRAN Version 2 program, controls whether file deletion will occur with the CLOSE specifier STATUS='DELETE'.

SPIE | NOSPIE
Specifies whether, when a program interrupt occurs, the run-time environment should take control.

STAE | NOSTAE
Specifies whether, when an abnormal termination is pending, the run-time environment should take control.

XUFLOW | NOXUFLOW
Specifies whether, when exponent underflow occurs, the program will be interrupted or continue with a hardware-provided fixup.

Appendix D. Commands for Interactive Debugging

Through an extensive command set, the Interactive Debug component of VS FORTRAN Version 2 provides the opportunity to examine and modify a program as it runs. These commands are:

ANNOTATE Copies source listings to the AFFPRINT file and controls the source listing window. (Use of the ON, OFF, and TOGGLE options require ISPF.)

AT Sets a breakpoint (the place at which processing of the user's program is suspended) at one or more statements. The AT command may specify a list of Interactive Debug commands to be processed whenever the breakpoint is reached.

AUTOLIST Automatically displays the contents of requested variables in the monitor window during animation or when program is suspended. (Use of this command requires ISPF.)

BACKSPACE Positions a sequentially-accessed external file at the beginning of the preceding record. BACKSPACE may be used where a sequential file is being read or written. This command allows the user to move backward in the file to rewrite or reread a record. Its use is similar to the FORTRAN BACKSPACE statement.

CLOSE Disconnects a sequentially accessed external file from an input or output unit. CLOSE may be used where a sequential file is being written or read. Its use is similar to the FORTRAN CLOSE statement.

COLOR Specifies color and highlighting choices for various fields of the debugging screen. (Use of this command requires ISPF.)

DBCS Indicates that the source file and listings may contain double-byte characters and that variables which contain double-byte characters can be entered and displayed during a full-screen or line-mode debugging session.

DESCRIBE Displays data types of variables, and dimensions of arrays.

ENDDEBUG Terminates all debugging activity and allows program processing to continue as though Interactive Debug had not been invoked. Optionally initiates program sampling.

ENDFILE Writes an end-of-file mark to a sequentially accessed external file. This command permits the user to edit or browse through the sequential file. Its use is similar to the FORTRAN ENDFILE statement.

ERROR Determines whether messages are received for run-time errors and specifies how library-detected errors are to be handled. Corrective action may be taken by the library, or control may be given to the user, allowing the specification of corrective action using the FIXUP command.

FIXUP Assigns new values for the arguments to a library function when the original arguments were in error. When no arguments are specified, standard corrective action is performed.

GO Resumes program processing after it has been suspended.

HALT Terminates processing of a command list, or causes processing to be suspended at the start of every statement, following every branch, or at entry to or exit from a new routine.

HELP Provides on-line information about using Interactive Debug and about messages printed in the vector report listing. HELP can be used to view the correct syntax and usage of a command or to receive a detailed tutorial on resolving an error message.

IF Processes a given Interactive Debug command when a logical expression is true. IF can be used to conditionally process Interactive Debug commands within a list of commands automatically processed at an AT breakpoint.

LIST Displays the values of variables, array elements, and arrays at the user's terminal or in a print data set. These can be displayed in a variety of formats.

LISTBRKS Lists all breakpoints currently set by the AT command and all WHEN conditions that are currently defined. LISTBRKS will also list the current HALT command status.

LISTFREQ Lists the number of times statements in the currently qualified program have been processed or lists statements that have never been processed.

LISTINGS Displays the listing data set panel. (Use of this command requires ISPF.)

LISTSAMP Lists sampling counts by statement, DO loop, or by program unit.

LISTSUBS Displays information about all debuggable program units in the load module.

LISTTIME Displays current elapsed processing time for DO loops and program units.

LISTVEC Displays vector length and stride information for DO loops, both vectorized and non-vectorized.

MOVECURS Moves the cursor between the command line and the source listing window. (Use of this command requires ISPF.)

NEXT Sets a temporary breakpoint (will be processed only once) at the next statement to be processed.

OFF Removes breakpoints set with the AT command.

OFFWN Suspends monitoring established with the WHEN command.

POSITION Positions the cursor in either the log file at the desired log line or the source window at the desired ISN, sequence number, or monitor line number. (Use of this command requires ISPF.)

PREVDISP Redisplays the previous panel displayed by the application program, if it was displayed via ISPF. (Use of this command requires ISPF.)

PROFILE Displays a panel which the user can use to specify desired defaults for later debugging sessions. (Use of this command requires ISPF.)

PURGE Cancels output from the currently-executing Interactive Debug command.

QUALIFY Specifies the default program unit that applies to statement and variable, array element, and array references.

QUIT	Stops all testing, exits Interactive Debug, and returns program control to the invoking program, usually the operating system.
RECONNECT	Tells the library to reset a file to its original (preconnected) state so that it can be implicitly opened again.
REFRESH	Gives users control over the degree of screen refresh when panels are displayed. (Use of this command requires ISPF.)
RESTART	Restarts the debugging session without erasing the current log. (Use of this command requires ISPF.)
RESTORE	Restores the source window to the last point of execution. (Use of this command requires ISPF.)
RETRIEVE	Retrieves the last command issued from the command line. (Use of this command requires ISPF.)
REWIND	Positions a sequentially accessed file at the beginning of the first record. REWIND allows the user to position at the beginning of the file so that it may be rewritten or reread. Its use is similar to the FORTRAN REWIND statement.
SEARCH	Locates an ISN or string in the source program. Locates a line number in the log. (Use of this command requires ISPF.)
SET	Assigns a value to a VS FORTRAN Version 2 variable or array element. Multiple elements of the array may be altered.
SIZE	Enlarges or reduces the size of the windows on the main debugging panel. (Use of this command requires ISPF.)
STEP	Provides an animated execution of the program (equivalent to repeated NEXT-GO pairs).
SYSCMD	Processes system commands during Interactive Debug processing.
TERMIO	Specifies the I/O routines to be used for the FORTRAN program terminal I/O. These can be the normal VS FORTRAN Version 2 library I/O routines or special Interactive Debug routines that allow I/O to be captured in the log of the debugging session. It can also specify whether to send output to a user while in batch mode.
TIMER	Initiates timing for specified DO loops or program units.
TRACE	Provides tracing of control transfers in the program or tracing of entries to, and exits from, subroutines.
VECSTAT	Records the average length and stride of vectors in the FORTRAN program.
WHEN	Permits suspending execution of the FORTRAN program when a given condition is satisfied or when a given variable or array element changes value.
WHERE	Displays the location within the FORTRAN program at which execution has been suspended. Optionally, WHERE displays the calling sequence leading to the currently executing program unit and the last 10 branches within the program.

WINDOW Opens, closes, and reconfigures the windows on the main debugging panel. (Use of this command requires ISPF.)

ZOOM Allows you to switch between allocating the entire screen to one window and sharing the screen among the windows as defined by the WINDOW panel. (Use of this command requires ISPF.)

Appendix E. Publications

The publications supporting the VS FORTRAN Version 2 licensed program are:

VS FORTRAN Version 2: Installation and Customization For VM (SC26-4339)—describes how to install VS FORTRAN and customize it to the needs of your installation site under VM.

VS FORTRAN Version 2: Installation and Customization For MVS (SC26-4340)—describes how to install VS FORTRAN and customize it to the needs of your installation site under MVS.

VS FORTRAN Version 2: Language and Library Reference (SC26-4221) —gives the rules for coding source programs using FORTRAN 77 and IBM extensions, and describes the library routines provided with VS FORTRAN Version 2.

VS FORTRAN Version 2: Programming Guide (SC26-4222)—gives guidance information on designing, coding, compiling, executing, and debugging VS FORTRAN programs.

VS FORTRAN Version 2: Interactive Debug Guide and Reference (SC26-4223)—gives guidance on invoking and executing VS FORTRAN Interactive Debug.

VS FORTRAN Version 2: Reference Summary (SX26-3751)—a convenient pocket-sized reference booklet summarizing source language and compiler options.

VS FORTRAN Version 2: Diagnosis Guide (LY27-9516)—gives information on diagnosing compiler and run-time library errors and instructions on submitting APARs.

VS FORTRAN Version 2: Licensed Program Specifications (GC26-4225)—a brief description of the VS FORTRAN Version 2 Release 2 Program Product, which serves as the basis for the warranty.

VS FORTRAN Version 2 Release 3 supports the Systems Application Architecture FORTRAN interface definition. The FORTRAN interface definition is documented in *Systems Application Architecture Common Programming Interface FORTRAN Reference*, SC26-4257.

Index

G

H

I

K

L

M

N

O

Reader's Comment Form

VS FORTRAN Version 2
General Information
GC26-4219-4

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Chapter/Section _____

_____ Page No. _____

Comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name _____ Phone No. (____)_____

Company _____

Address _____

_____

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

GC26-4219-4

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

back of the title page)

Program Number
5668-805
5668-806

File Number
S370-20

The VS FORTRAN Version 2 Library

Diagnosis Guide	LY27-9516
General Information	GC26-4219
Installation and Customization for MVS	SC26-4340
Installation and Customization for VM	SC26-4339
Interactive Debug Guide and Reference	SC26-4223
Language and Library Reference	SC26-4221
Licensed Program Specifications	GC26-4225
Programming Guide	SC26-4222
Reference Summary	SX26-3751

GC26-4219-4

Printed in U.S.A.