VS FORTRAN Version 2

General Information

Release 6

GC26-4219-10

VS FORTRAN Version 2

General Information

Release 6

GC26-4219-10

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

| **Eleventh Edition (October 1993)**

This edition replaces and makes obsolete the previous edition, GC26-4219-9.

This edition applies to VS FORTRAN Version 2 Release 6, Program Numbers 5668-805, 5688-087, 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 “About This Book,” and are indicated by a vertical bar to the left of a change.

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, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to: IBM Canada Ltd. Laboratory, Information Development, 2G/345/1150/TOR, 1150 Eglinton Avenue East, North York, Ontario, Canada, M3C 1H7.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

Trademarks and Service Marks

About This Book

This book is intended to help managers and technical personnel evaluate and plan for using the VS FORTRAN Version 2 licensed program. It contains high-level evaluation information.

How This Book Is Organized

This book is organized as follows:

Chapter 1, "The Features of VS FORTRAN Version 2," describes the high performance capabilities of the parallel and vector features, the language capabilities, programming aids, interactive debug, and other debugging aids provided by VS FORTRAN Version 2.

Chapter 2, "Programming Requirements and Support," describes the hardware and software requirements necessary to run VS FORTRAN Version 2, as well as its compatibility with previous versions of FORTRAN. In addition, this chapter provides information about IBM* program services and the VS FORTRAN Version 2 license and warranty.

Appendix A, "Summary of the VS FORTRAN Version 2 Language," shows the high-level aspects of the statements and features of VS FORTRAN Version 2.

Appendix B, "Supplied Functions and Subroutines," lists the supplied mathematical, character manipulation, bit manipulation, and internal data conversion functions of VS FORTRAN Version 2, as well as the supplied service and error-handling subroutines.

Appendix C, "Options," lists and briefly defines the compile-time and run-time options of VS FORTRAN Version 2.

Appendix D, "Commands for Interactive Debugging," lists and briefly defines the VS FORTRAN Version 2 interactive debug commands.

Appendix E, "Publications," lists the publications that support the VS FORTRAN Version 2 licensed program.

Publications

Figure 1 lists the VS FORTRAN Version 2 publications and the tasks they support.

Figure 1 (Page 1 of 2). VS FORTRAN Version 2 Publication Library

Task	VS FORTRAN Version 2 Publication	Order Number
Evaluation and Planning	*General Information* *Licensed Program Specifications*	GC26-4219 GC26-4225
Installation and Customization	*Installation and Customization for CMS* *Installation and Customization for MVS*	SC26-4339 SC26-4340

* IBM is a trademark of the International Business Machines Corporation.

Figure 1 (Page 2 of 2). VS FORTRAN Version 2 Publication Library

Task	VS FORTRAN Version 2 Publication	Order Number
Application Programming	*Language and Library Reference*	SC26-4221
	Programming Guide for CMS and MVS	SC26-4222
	Interactive Debug Guide and Reference	SC26-4223
	Reference Summary	SX26-3751
Library Reference	*Master Index and Glossary*	SC26-4603
Diagnosis	*Diagnosis Guide*	LY27-9516
Migration	*Migration from the Parallel FORTRAN PRPQ*	SC26-4686

Industry Standards

The VS FORTRAN Version 2 compiler and library are designed according to the specifications of the following industry standards, as understood and interpreted by IBM as of December 1990.

The following two standards are technically equivalent. In the publications, references to **FORTRAN 77** are references to these two standards:

- American National Standard Programming Language FORTRAN, ANSI X3.9-1978 (also known as FORTRAN 77)
- International Organization for Standardization ISO 1539-1980 Programming Languages—FORTRAN

The bit string manipulation functions are based on ANSI/ISA-S61.1.

The following two standards are technically equivalent. References to **FORTRAN 66** are references to these two standards:

- American Standard FORTRAN, X3.9-1966
- International Organization for Standardization ISO R 1539-1972 Programming Languages—FORTRAN

At both the FORTRAN 77 and the FORTRAN 66 levels, the VS FORTRAN Version 2 language also includes IBM extensions. References to **current FORTRAN** are references to the FORTRAN 77 standard, plus the IBM extensions valid with it. References to **old FORTRAN** are references to the FORTRAN 66 standard, plus the IBM extensions valid with it.

Documentation of IBM Extensions

In addition to the statements available in FORTRAN 77, IBM provides “extensions” to the language. In the *VS FORTRAN Version 2 Language and Library Reference*, these extensions are printed in color.

Summary of Changes

Release 6, October 1993

Major Changes to the Product

Support for AIX/370 is not included in VS FORTRAN Version 2 Release 6. Support for AIX/ESA* is found in the AIX VS FORTRAN/ESA product.

- Printable copies of certain of the VS FORTRAN Version 2 Release 6 publications are provided on the product tape.
- Support for additional language has been added, much of which address previous incompatibilities between XL FORTRAN and VS FORTRAN
  - Additional constant and operator support:
    - Quote-delimited literal character constants
    - Maximum negative integer literal value
    - Typeless constants (binary, octal and hexadecimal)
    - Named constants in complex constants
    - .XOR. logical operator
    - <> graphical relational operator
  - Storage classes:
    - STATIC and AUTOMATIC storage classification statements
    - IMPLICIT STATIC and IMPLICIT AUTOMATIC statement support
  - Additional data type support:
    - LOGICAL*2 and LOGICAL*8
    - INTEGER*1 and INTEGER*8
    - UNSIGNED*1
    - BYTE
    - DOUBLE COMPLEX and DOUBLE COMPLEX FUNCTION type specifications
  - XREF and MAP Processing has been improved to remove storage-ordering perturbations.
  - Additional intrinsic function support:
    - INTEGER*8, INTEGER*2, and INTEGER*1 support for existing functions
    - XOR, LSHIFT, RSHIFT, ISHFTC, IBITS, and LOC have been added.
  - HALT compiler option to reduce compilation times for unsuccessful compilations
  - MVBITS and ARGSTR service routines
  - Dynamic storage support, for allocating and deallocating storage at run-time
    - Integer POINTER data type

* AIX/ESA is a trademark of the International Business Machines Corporation.

        - Including dynamically dimensioned arrays
        - ALLOCATED, DEALLOCATE, and NULLIFY statements
        - Intrinsic function ALLOCATED
        - Compile-time option DDIM
    - Dynamically loaded module support (via DYNAMIC option)
    - I/O features enhancements:
        - Support for dummy arguments in NAMELIST lists.
        - NML keyword for NAMELIST READ/WRITE statements
        - OPEN and INQUIRE statement support for the POSITION, PAD, and DELIM specifiers
        - B and O format control codes
        - Expansion of the Z format control code
        - $ format control code
- Optimization improvements:
    - Optimization for pipelined instruction scheduling
    - Improved optimization for Inter-loop register assignments
- Message improvements:
    - All messages written to SYSTERM file
    - Information messages no longer marked as errors
- Run-time options improvements:
    - Abbreviations are now allowed.
    - Default I/O units are now user-specifiable with ERRUNIT, RDRUNIT, PRTUNIT, and PUNUNIT
- Vector support enhancements:
    - The VECTOR option defaults can be set at installation.
    - VECTOR suboptions MODEL and SPRECOPT.
    - Vectorized SIGN, ANINT, DNINT, NINT and IDNINT intrinsic functions
    - Vector performance improvements
- Parallel support enhancements:
    - LOCAL statement allowing arrays for Parallel Do / Parallel Sections
    - NAMELIST processing for parallel environment allows local variables
    - Support for user-generated parallel trace (including service routines PTWRIT and PTPARM and suboption TRACE)
    - Parallel execution controls (affinity control)
    - Load module support for routines invoked via SCHEDULE and PARALLEL CALL

Release 5, September 1991

Major Changes to the Product

- Support has been added in the compiler and library for the Advanced Interactive Executive/370 (AIX/370) operating system. In addition, programs designed to be run on AIX/370 may use the following enhancements to VS FORTRAN Version 2:
  - The **fvs** command to invoke the VS FORTRAN Version 2 compiler.
  - Use of tab characters in source programs.
  - Communication between Fortran programs and C programs.
  - Coding of indirectly recursive Fortran routines.
- Support for parallel programs[1] has been added for programs running under CMS and MVS. These enhancements support:
  - Automatically generating parallel code for DO loops using a compile-time option.
  - Explicit coding of parallel loops, sections, and calls with parallel language extensions.
  - Using lock and event services to control synchronization in parallel programs.
  - Directing the compiler to generate parallel or serial code using enhanced directives.
  - Determining the number of virtual processors available and specification of the number of virtual processors to use during run time.
  - Using I/O within parallel programs.
  - Calling subroutines within parallel loops and sections.
  - Obtaining information for tuning your parallel program using a compiler report listing.
- Support for extended common blocks has been added for programs running under MVS/ESA* or VM/ESA*. Compile-time and run-time enhancements and options have been added to support the three types of common blocks that now exist.
- Virtual storage constraint relief has been added for compiling under MVS/XA*, MVS/ESA, VM/XA*, or VM/ESA. This support allows the compiler to reside above the 16MB line and to process in 31-bit addressing mode. Therefore, larger programs can now be compiled.
- The default name for a main program has been changed from MAIN to MAIN#, which allows MAIN to be used as a user name for a common block or other global entity.

[1] The VS FORTRAN Version 1 standard math routines (VSF2MATH) are not supported for parallel processing. Interactive debug can be used only with non-parallel programs, and with serial portions of parallel programs when the run-time option PARALLEL(1) is specified.

* MVS/ESA, MVS/XA, VM/ESA, and VM/XA are trademarks of the International Business Machines Corporation.

- Array declarator expressions for object-time dimensions can now be of type Integer*2.

Release 4, August 1989

Major Changes to the Product

- Enhancements to the vector and optimization features of VS FORTRAN Version 2
  - Automatic vectorization of user programs is enhanced by improvements to the dependence analysis done by the compiler. Specifically, the following constructs are eligible for vectorization:
    - Loops containing simple READ, WRITE, and PRINT statements
    - Loops containing equivalenced arrays
    - Loops containing branches out of the loop
    - Loop bound appearing in a subscript expression
    - Loop nests that process a triangular matrix
    - Simple IF loops
    - Integer sum reduction.
  - Additional advanced vector optimization.
  - In programs optimized at either OPT(2) or OPT(3), arithmetic constants will be propagated globally for scalar variables.
  - In programs optimized at either OPT(2) or OPT(3), informational messages are issued when local scalar variables may be referenced before they have been initialized.
  - Improved performance when the CHAR, ICHAR, and LEN character intrinsic functions are used.
  - Single and double precision complex divide routines can be vectorized.
- Enhancements to input/output support in VS FORTRAN Version 2
  - Improved data transfer rate for sequential DASD and tape input/output on MVS systems.
  - Ability to perform data striping (parallel I/O) on sequential data sets.
  - Ability to detect input conversion and record length errors.
- Enhancements to the intercompilation analyzer (ICA)
- Enhancements to the language capabilities of VS FORTRAN Version 2
  - Ability to use graphic relational operators as an alternative to FORTRAN 77 relational operators.
- Enhancements to the pseudo-assembler listing provided by VS FORTRAN Version 2
- Enhancements to the math routines:
  - Single, double, and extended precision MOD library routines are more precise.
  - Single and double precision scalar complex divide routines are more precise and faster, and are vectorizable.

- The old complex divide and MOD routines are in the alternate math library.

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 online HELP function of interactive debug.
  - Vectorization is allowed at OPTIMIZE(2) and OPTIMIZE(3).
- 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 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 variable names and as character data in source programs and I/O, and for interactive debug input and output.

[*] Systems Application Architecture and SAA are trademarks of the International Business Machines Corporation.

  - 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 and MVS/ESA.
  - 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.
  - Ability to determine the amount of CPU time used by a program or a portion of a program by using the CPUTIME service subroutine.
  - Ability to determine the Fortran unit numbers that are available by using the UNTANY and UNTNOFD service subroutines.
- Enhancements to the full screen functions of interactive debug

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 run-time option to control checking of file existence during the processing 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 processing 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.

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 run-time option
- Addition of support for VM/XA System Facility Release 2.0 (5664-169) operating system

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 previous versions of VS FORTRAN, 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:

- Extensive language capabilities
- High-performance capabilities
  - Vector support
  - Parallel support
- Highly optimizing compiler
- Programming aids
- Debugging aids

VS FORTRAN Version 2 supports the following operating systems:

MVS
VM

For specific information on operating systems, see "Operating System Requirements" on page 21 and "Operating System Restrictions" on page 23.

The following pages describe features and advantages of VS FORTRAN Version 2.

Extensive Language Capabilities

VS FORTRAN Version 2 offers flexible language capabilities. It supports both FORTRAN 77 and FORTRAN 66 language standards and IBM's Systems Application Architecture (SAA) FORTRAN definition. Many extensions are available with each of these standards.

Some of the highlights of the VS FORTRAN Version 2 language are described below. (For a complete listing of the major elements of the language, see Appendix A, "Summary of the VS FORTRAN Version 2 Language" on page 29.)

Structuring Aids

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

DO WHILE Statement: which processes groups of statements based on the evaluation of a logical expression, thus providing one of 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.

The INCLUDE Directive: Lets you insert source from a separate file 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 to give the ability to selectively include source that can customize a single program for various applications, without requiring coding and maintaining a separate program for each.

End-of-Line Commentary: Lets programmers write a comment on the same line as the code to which it pertains.

Free-Form Source

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

Extended Statement Length

You can write statements that 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.

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.

Supported Data Types

VS FORTRAN Version 2 allows you to specify and process many types of variables and data.

Figure 2 (Page 1 of 2). Supported Data Types

Data Type	Length (Bytes)	Range of Values
Integer	1	-128 through 127
	2	-32 768 through 32 767
	4	-2 147 483 648 through 2 147 483 647
	8	-9 223 372 036 854 775 808 through 9 223 372 036 854 775 807
UnSigned	1	0 through 255
Byte	1	-128 through 127
Pointer	4	address-space address
	8	address- or data-space address
Logical	1	True and False
	2	True and False
	4	True and False

Figure 2 (Page 2 of 2). Supported Data Types

Data Type	Length (Bytes)	Range of Values
	8	True and False
Real	4	10**(-78) to 10**(+75) (precision: 6 hex digits; 6 decimal digits)
	8	10**(-78) to 10**(+75) (precision: 14 hex digits; 15 decimal digits)
	16	10**(-78) to 10**(+75) (precision: 28 hex digits; 32 decimal digits)
Complex	8	As Real, length 4
	16	As Real, length 8
	32	As Real, length 16
Character	1 to 32767	EBCDIC or DBCS characters

Constants: VS FORTRAN Version 2 supports constants of type integer, real, complex (including constant expressions), logical, character (with either apostrophe (') or quotation marks (") delimiters), binary, octal, and hexadecimal.

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.

Operators

Relational Operators: As an IBM extension, VS FORTRAN Version 2 recognizes the characters <, >, /, and = to form relational operators.

XOR.: The logical operator .XOR. is recognized as a logical operator, equivalent to .NEQV..

Standard Language Flaggers

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

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

IMPLICIT Statement Extensions

The IMPLICIT statement is extended with three keywords.

IMPLICIT NONE IMPLICIT NONE precludes all default implicit typing except for the intrinsic functions. This provides a check on variable usage and identifies otherwise hard-to-find programming errors such as spelling mistakes.

IMPLICIT STATIC and IMPLICIT AUTOMATIC: The IMPLICIT statement is also extended with the keywords AUTOMATIC and STATIC. These statements specify that variables beginning with the specified starting letters are to have the specified storage class unless explicitly overridden.

Dynamic Array Capabilities

VS FORTRAN Version 2 supports pointer variables with associated pointee variables, dynamically dimensioned arrays, and storage allocation statements to offer users flexibility in handling storage usage and size.

Pointers: A new type of variable, POINTER, that references dynamically addressed storage, is supported. The POINTER statement is used to specify a pointer and associate a pointee variable or array to the pointer. The pointer can be set by its use in the ALLOCATE statement (see “Allocation statements”), by assignment from another pointer, or by the use of the LOC intrinsic function.

The pointer is considered to be of type integer and is either 4 or 8 bytes in length, depending on the PTRSIZE compiler option.

Allocation statements: Three statements, ALLOCATE, DEALLOCATE, and NULLIFY can be use to dynamically allocate storage (using a pointer variable), deallocate the storage, and to nullify the value of the pointer, respectively. These enable programs to obtain space when needed within the application and to release it for future use, determining the amount needed dynamically. Thus, the use of storage can be made optimal for the problem size. The ALLOCATED intrinsic function can be use to determine if a pointer variable locates allocated space.

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). See “Operating System Restrictions” on page 23.

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

FORMAT extensions: You can use a comma to indicate the end of a data field in a formatted input record, thus relaxing the rigid rules for the format of the data.

Expanded FORMAT codes B, O, and Z are provided for editing binary, octal, and hexadecimal data, respectively. The dollar ($) FORMAT code is provided to suppress the end-of-record condition.

NAMELIST extensions: The NML keyword is support as an alternate to FMT in NAMELIST I/O statements to ease migration to and from other platforms.

With VS FORTRAN Version 2 Release 6 dummy arguments may appear in NAMELIST lists.

OPEN and INQUIRE extensions: The OPEN and INQUIRE statements in VS FORTRAN Version 2 Release 6 have three additional specifiers:

POSITION Enables you to control and inquire about file positioning for sequential, non-keyed files

PAD Enables you to control processing for short records (those which contain less data than the input list needs)

DELIM Enables you to indicate how list-directed and NAMELIST processing delimits character constants on output.

Sequential Input/Output

Through the use of both BSAM and striped input/output, VS FORTRAN Version 2 provides the means for improving sequential input/output transfer rates under MVS.

For instance, VS FORTRAN allows more than 2 buffers for DASD and tape input/output under MVS. The number of buffers to be used is specified by using JCL or by using FILEINF. Under DFSMS* 1.1, the BSAM access method allows up to 255 buffers to be used.

Striped Input/Output: Under MVS, you can use striped input/output to increase data transfer rates for unnamed files. Striped input/output allows you to split a file across multiple channels for simultaneous input/output performance on multiple disks, tapes, or a mixture of disks and tapes. To use striped input/output, you change the JCL; however, you do not need to change any of the existing application code. In addition, you can determine the number of stripes you want as well as the size of the striping unit, which corresponds to the logical block size of the file.

Under CMS, you can also use striped input/output, but it will not improve performance.

Asynchronous I/O: VS FORTRAN Version 2 includes support for unformatted data to be written and read asynchronously. This allows other program statements to be executed while data transfer is taking place, eliminating the wait for the transfer to complete.

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. VS FORTRAN Version 2 Release 4 further enhanced the MOD library routines by

* DFSMS is a trademark of the International Business Machines Corporation.

replacing the old scalar routines, with more precise single, double, and extended precision routines.

In addition to supplying increased accuracy, many of the Version 2 routines, including single and double precision complex divide library routines, can be vectorized. VS FORTRAN Version 2 Release 6 has added SIGN, ANINT, DNINT, NINT, and IDNINT to this set of vectorized intrinsic functions, removing them as inhibitors to vectorization.

All intrinsic functions that take integer arguments now accept all supported integer lengths.

Bit String Manipulation

VS FORTRAN Version 2 supplies intrinsic functions and a service subroutine that allow you to view integer and unsigned data (all lengths) as an ordered set of bits. You can perform logical operations on the bits; shift them left, right, or circularly; set them, test them, extract them, and move 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:

- Command-line parameter retrieval
- 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 and Subroutines” on page 35.

High-Performance Capabilities

Along with the vector facility, which can increase the performance of applications, parallel processing enables programmers to reduce elapsed run time when running VS FORTRAN programs and increases the high-performance capabilities of the VS FORTRAN product.

The Parallel Feature

Parallel processing support allows you to write and run parallel programs on IBM multiprocessors. By splitting a program into multiple, independent instruction streams, parallel processing simultaneously runs different portions of a single application program across multiple processors. For some applications, if you run a program in parallel, you can significantly reduce the elapsed time required to run that program.

VS FORTRAN Version 2 allows you to specify various forms of parallelism that can occur in a Fortran program. For instance, parallel language statements dispatch units of work called *parallel threads*. These consist of subroutines, iterations of a loop, and sections of code that can be processed concurrently.

The Vector Feature

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

A simple recompilation allows most existing programs to take advantage of the 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). Vector code complements parallel code, and for some applications, parallel-vector code can be generated, resulting in faster run time than parallel-only or vector-only cases.

The Compiler Performs with Versatility

The VS FORTRAN Version 2 compiler has the ability to determine whether parallel or serial code, or vector or scalar code will give you the most efficient performance on DO loops. By using compiler directives and options, you gain finer control over the performance of your Fortran programs.

If your application requires interprogram communication, the VS FORTRAN Version 2 compiler allows you to use dynamic common blocks, extended common blocks, or static common blocks to communicate between main programs and subroutines. Using the compile-time options, you can reference or share named common blocks and run in both serial and parallel environments.

VS FORTRAN Version 2 compiles source programs to produce either traditional non-vector code (usually called scalar code), vector code, parallel-serial code, or parallel-vector code. The compiler examines up to eight levels of loops, and selects the eligible loop that will yield the fastest running time of the entire nest for conversion to vector instructions. 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.

Compiler Directives: Directives are available in VS FORTRAN Version 2 that enable you to affect the parallel and vector processing of your program. Through extensions to the PREFER compiler directive, you can choose to run DO loops in parallel or serial mode, and specify the number of loop iterations that should be assigned to run concurrently. Extensions to the IGNORE directive allow you to bypass *dependence checking* if you are running parallel DO loops that contain CALL statements or function invocations. Using these directives, you can:

- Tell the compiler what loop-count values are expected during processing. This allows the compiler to better determine whether the loop can be advantageously run with parallel or vector code.
- Indicate to the compiler that certain data relationships should not be considered in determining eligibility for vectorization or parallelization. This makes certain ambiguous loops more likely to either vectorize or run in parallel.
- Specify whether you prefer vector, scalar, parallel, or serial code.

You can include individual directives for each DO loop or choose to use certain directives globally within the source program.

Compiler Reports: In addition to producing parallel and vector code, the compiler can produce a report that tells if code was generated in parallel, serial, vector, or scalar mode. Depending on what you request, the compiler will generate a report that:

- Lists the results of the compiler's source code analysis, and shows which loops you chose to run in parallel, vector, or both
- Tells you why parallel or vector code was or was not generated
- Lists compile-time statistics for parallel statements in a table
- Lists compile-time statistics on the length and stride of each DO loop in the source program (both vectorized and nonvectorized).

Note: For vector nonparallel code, interactive debug will also generate run-time statistics.

You can use the report to *tune* the source program; that is, you can examine the source program for possible coding refinements. For example, you may recode certain sections that were not vectorized 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 you coded the source—or according to the sequence in which the compiler placed the parallelized or vectorized source code.

How Parallel Processing Works

In a VS FORTRAN Version 2 parallel environment, you manage and dispatch *parallel tasks* and *parallel threads*. A *parallel task* is a complete logical environment with its own local storage and subprograms. Within a task, you dispatch parallel threads for that same task or assign them to a different task. You can either explicitly manage and dispatch multiple parallel threads, using the VS FORTRAN Version 2 language extensions, or implicitly allow the compiler to identify threads that are eligible to run in parallel.

A collection of parallel threads scheduled to be processed is called *parallel processing*. The VS FORTRAN Version 2 library maps parallel work onto *virtual processors*, logical processors that run parallel threads. Next, the system maps these VS FORTRAN virtual processors onto the real machine processors.

The Fortran compiler and library provide these virtual processors in multiple environments. You need only refer to the VS FORTRAN Version 2 processors. To support the running of parallel programs, VS FORTRAN Version 2 provides the following:

- Extensions to the compiler for automatically generating parallel code
- Extensions to the language for explicit programming in parallel
- Extensions to the library for synchronizing parallel execution
- I/O support for parallel processing through locks and events.

The Compiler Generates Parallel Code Automatically

You can generate automatic parallel code for DO loops with VS FORTRAN Version 2 if you specify the correct compiler option. Parallel code for DO loops allows each iteration of the DO loop to be executed asynchronously and independently of the other iterations. A DO loop must meet similar eligibility requirements for parallel execution as for vector execution. If there are dependencies, you can often modify the code to eliminate these inhibitors, thereby allowing parallel code to be scheduled.

If the compiler determines that a loop is eligible to run in both parallel and vector, it can generate vector-parallel code that may result in faster running than the parallel-only or vector-only cases. Each loop that is eligible to run in parallel must contain computationally independent iterations and enough work to justify parallel processing before it can be compiled into parallel code. Coding modifications, however, can remove these inhibitors to generating parallel code.

The Language Generates Parallel Code Explicitly

Parallel support in VS FORTRAN Version 2 includes language extensions that allow you to manage the processing of parallel work through the creation of new parallel tasks and the assignment of parallel threads to different tasks. For task management, VS FORTRAN Version 2 uses the following statements:

ORIGINATE
TERMINATE
SCHEDULE
WAIT FOR TASK
WAIT FOR ANY TASK
WAIT FOR ALL TASKS

For more information on these statements, see Appendix A, "Summary of the VS FORTRAN Version 2 Language" on page 29.

In addition to scheduling the running of parallel work, you can dispatch new parallel threads within a task environment by explicitly coding parallel loops, parallel sections, and parallel calls. The following statements allow you to explicitly code parallel threads that may be run independently of and in parallel with threads within the same type of statement:

PARALLEL DO
LOCAL
DOBEFORE
DOEVERY
DOAFTER
EXIT

PARALLEL SECTIONS
LOCAL
SECTION
END SECTIONS

PARALLEL CALL
WAIT FOR ALL CALLS

For instance, iterations of a loop run in parallel with other iterations, and with sections or subroutines (see Appendix A, “Summary of the VS FORTRAN Version 2 Language” on page 29).

The LOCAL statement for either Parallel Do or Parallel Sections specifies the variables and arrays which are to have private copies for each virtual processor that participates in the execution of the loop or section.

The Language Converts Multitasking Facility to Parallel Processing

The multitasking facility of VS FORTRAN Version 2 handles the dispatching of multiple subroutines, so that the MVS user can run each subroutine simultaneously on a separate processor. You control the subroutines through normal Fortran CALL statements. (Such subroutines are not available to interactive debug.)

Although the multitasking facility and parallel processing both run independent subroutines simultaneously, there are differences between them. For instance, parallel language extensions, automatic parallelization of DO loops, and extensions to library services for controlling the running of code through the use of locks and events are available if you run programs that use parallel processing support. However, the multitasking facility (MTF) allows parallel processing to be performed only at the subroutine level. Further, if you convert your MTF programs to use parallel language constructs and facilities, you can simplify link-edit procedures, thus allowing easier maintenance of programs.

You can use either MTF or the parallel language, but not both. If you want to convert MTF applications to use parallel constructs, you should consider the following:

- MTF programs that use link mode should be converted to use load mode, because parallel programs can be run only in load mode.
- Virtual storage requirements may increase (although use of the RENT compiler option can reduce these requirements).
- CPU utilization requirements may increase.
- Assembler routines cannot use MVS supervisor services macros that are sensitive to the task in which they are run.

Converting the Main Task Program to Parallel: You can also convert specific MTF service routines to parallel language constructs in order to make use of the parallel language features, automatic generation of parallel code, or VS FORTRAN lock and event services by performing the following conversions:

MTF Calls to...	**Parallel Conversion**
DSPTCH[1]	SCHEDULE statements
SHRCOM	SHARING clauses on the SCHEDULE statement
SYNCRO	WAIT FOR ALL TASKS statements
NTASKS	NPROCS function
Add ORIGINATE statements for the number of subtasks specified on the AUTOTASK run-time option.[2]	

1 You may use the PARALLEL CALL statement to replace a call to DSPTCH only if the following are true:

- All common blocks in the main task program and subprograms can be shared, with the program processing remaining computationally independent.
- All units, including error message and print units, can be shared.

2 If your program schedules more parallel subroutines than the number of subtasks specified on the AUTOTASK run-time option, you may need to add extra statements to control the scheduling of the parallel subroutines.

The Library Synchronizes Parallel Processing

Services provided by VS FORTRAN Version 2 include subroutines for the management of locks and events. A *lock* prevents the concurrent running of certain critical operations, such as referencing or updating. Four new subroutines have been added to the library for the management of parallel locks. These subroutines create and delete locks for the program and obtain and release a lock for a parallel unit of work.

An *event* permits tasks to explicitly synchronize their processing through interthread signaling. Five subroutines are provided for the management of parallel events. These subroutines create and delete events, initialize event parameters, post events, and wait for events.

A new function, NPROCS, has been added to the library to allow the program to determine the number of virtual processors specified at run time.

| ## User Synchronization

| Service subroutines are provided to enable user control of the parallel environment.
| The current execution environment can be serialized to the main task (MVS) or
| virtual processor 0 (CMS) to allow the use of system services. Serialization
| remains in effect until terminated by the user. There is also a service subroutine to
| cause the VS FORTRAN Version 2 library to interrupt the execution of the current
| thread and attempt to execute any threads waiting to execute.

| These services remove inhibitors to parallelization by allowing code that requires
| system services to execute in the parallel environment.

The Library Provides I/O Support for Parallel Processing

VS FORTRAN Version 2 provides I/O support for parallel processing. The standard VS FORTRAN file definitions used for nonparallel execution are the default file definitions for parallel processing.

I/O support for parallel processing consists of:

- Default file definitions for the error message unit and standard print unit
- Extensions to dynamic file allocation to permit I/O to unnamed temporary files within an originated task
- Named file I/O
- INQUIRE support.

Parallel Trace

VS FORTRAN Version 2 Release 6 provides the Parallel Trace Facility to record significant actions occurring during the execution of parallel programs. By using the data obtained via the Parallel Trace Facility, users can analyze the behavior of these programs and perform necessary tuning and debugging functions.

The Parallel Trace Facility records each significant action as a record in an output file, known as the **Trace Log File**. Separate programs, collectively referred to as "trace tools," are used to analyze these records; trace tool programs must be used as post-processors against the Trace Log File after execution of the parallel program has completed.

The VS FORTRAN Version 2 Parallel Trace Facility is similar to the Trace Facility provided in the VS Fortran Parallel PRPQ (Program Number 5799-CTX); users familiar with that facility will find that the Parallel Trace Facility is an extension of that facility, although certain elements have been changed and improved.

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 3 illustrates this reduction in run time.

```
    DO  8  K = 1, 90
8   A(K)=A(K)+B(K)
```

Traditional scalar (non-vector) processing requires each element of the array A to be computed in sequence, one after the other:

```
A(1)=A(1)+B(1)   A(2)=A(2)+B(2)   A(3)=A(3)+B(3) .... A(90)=A(90)+B(90)
─────────────────────────────── run time ────────────────────────────►|
```

In comparison, vector processing allows the computation of multiple elements of array A to be overlapped, speeding up processing:

```
A(1)=A(1)+B(1)
 A(2)=A(2)+B(2)
  A(3)=A(3)+B(3)
   .
    .
     .
      A(90)=A(90)+B(90)
───  run time  ─────►|
```

Figure 3. How Vector Processing Speeds Run Time

Vector 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 (with some restrictions)
COMPLEX*16 (with some restrictions)
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
| LOGICAL*2
| LOGICAL*8
| INTEGER*8
| UNSIGNED*1

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

Certain types of branches
| Allocation 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

DO loops containing READ, WRITE, and PRINT statements may be eligible for partial vectorization. I/O statements cannot be vectorized.

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 cannot be safely vectorized, or would not run faster as vector instructions, are compiled into standard nonvector instructions.

VS FORTRAN Version 2 has an enhancement that reduces the overhead required for setting up loops and initializing them. If you are using short vectors, this enhancement will improve performance in certain cases.

Vector Facility Exploitation

VS FORTRAN Version 2 fully exploits the ES/9000* vector facility through the use of the VECTOR(MODEL) option, which allows the user to identify the level of the vector facility on which the program will be executed. This allows the compiler to generate code to use the enhanced features of that facility whenever possible, allowing the user to get full benefit from the facility.

The SPRECOPT suboption is provided to allow single-precision Multiply and Add/Subtract instruction sequences to be generated, further exploiting the facility.

Interactive Debug Tunes Source Programs

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, you 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, you can modify the vector directives in the program to more accurately reflect the vector lengths and strides.

Programming Aids

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

Optimized Object Code

You 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.)

   With VS FORTRAN Version 2 Release 6, full optimization includes *instruction scheduling* and inter-loop register assignments. Instruction scheduling removes conflicts in register use that cause serialization within the instruction pipeline. Inter-loop register assignment reduces the number of instructions generated by considering prior assignments made for inter-loop variables appearing in inter loops at the same level. Vector loops in particular benefit from this 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.)

* ES/9000 is a trademark of the International Business Machines Corporation.

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 Compiler and Library Routines

Users can choose to have the VS FORTRAN Version 2 compiler loaded above the 16-megabyte line and 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 ESA or XA environment—allows many library routines to reside above the 16-megabyte line.

Dynamic Loading of User Routines

Modules containing user subroutines and functions can be dynamically loaded during execution as needed. The DYNAMIC compiler option is used to identify a referenced subprogram name that is the name of a separately linked module.

This can be particularly advantageous when used in the parallel environment where separate copies of routines are required for different tasks.

Multiple System Support

VS FORTRAN Version 2 runs on MVS and CMS systems, including the ESA and XA environments with their large amounts of virtual storage. (For a complete list, see "Operating System Requirements" on page 21.)

A VS FORTRAN Version 2 program can be compiled on one supported operating system and then link-edited or loaded and run on the other 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 ESA or XA environment. This also reduces storage requirements by allocating only those common blocks that are in subprograms actually referenced during processing.

Extended Common

Extended common allows the user to define multiple extended common blocks at run time, each as large as two gigabytes in size. The number of extended common blocks and their combined size is not limited by VS FORTRAN. Where dynamic common blocks are allocated storage from within program space, extended common blocks are allocated storage from within an ESA/390* data space.

Static Common

Common blocks that are compiled in a VS FORTRAN program and reside within the running program are known as static common blocks. Static common blocks are limited in size because the combined sizes of all the blocks and the program code cannot exceed 16 megabytes of storage.

Upward Compatibility

In general, VS FORTRAN Version 2 is compatible with Version 1 and with earlier IBM large-system versions of Fortran. For details, see “Compatibility between Versions” on page 24.

Data-in-Virtual Support

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

Data Window Services

MVS/ESA allows FORTRAN programmers to make use of data window services. Data window services are a set of system services that can be accessed by calls from high-level languages. Using data window services, VS FORTRAN Version 2 can define large temporary and permanent collections of data that application programs can access.

Intercompilation Analysis

Intercompilation analysis(ICA) significantly reduces development time by detecting incompatibilities between program units before you begin to debug your applications. It detects conflicting external names, provides usage information for common blocks, and checks for disagreement between actual arguments and dummy arguments, and for undefined arguments. ICA will do a complete search down the CALL chains for modified constant arguments.

When the ICA(UPDATE) compiler option is in effect, VS FORTRAN Version 2 builds a file of information obtained from compilations. You 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. Using a suboption of MXREF, you can eliminate lines caused by the inclusion of ICA files that contain unreferenced subprograms.

* ESA/390 is a trademark of the International Business Machines Corporation.

Using the MSGOFF suboption, you can suppress error messages that you both expect and allow to occur. VS FORTRAN Version 2 lists the numbers of the suppressed messages immediately after the heading.

By using the DEF suboption, to specify names of ICA files that are to be searched for specific definitions, you can save storage during ICA processing.

| By using the suboption RCHECK, you can detect recursive subroutine calls or function invocations.

At installation, you can specify ICA and a limited set of suboptions.

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 file identifier (a ddname) or data set name.

A 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 that 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.

Debugging Aids

VS FORTRAN Version 2 offers a variety of functions which help with the task of debugging programs.

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” on page 46.

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 21.

Source Listing and Display Animation

VS FORTRAN Version 2 interactive debug offers full screen support. You 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 you can halt processing where desired, a stop-action or freeze-frame ability is also provided.

Because your 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 portions of program units, are 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 processor 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: You can measure the processor 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(DBCS), 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.

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 include an explanation, and the identify of 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 utility routines supplied with VS FORTRAN Version 2 (for example, OVERFL and DVCHK). The product supplies additional routines 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.

Static debug statements cannot be used in a parallel program. Static debug statements cannot be used with extended common blocks.

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 utility routine provided by VS FORTRAN Version 2.

Chapter 2. Programming Requirements and Support

Operating System Requirements

Figure 4. Compilation and Execution Support for Vector and Parallel Features

Product	Vector	Parallel	Scalar/ Serial
MVS/SP[*] Version 1 (5740-XYN or 5740-XYS) all releases, with or without TSO/E (5665-285)	◊	◊	♦
MVS/XA:			
• MVS/SP Version 2 (5665-291 or 5740-XC6) all releases, and MVS/XA DFP Version 2 (5665-XA2) Release 3 or later, with or without TSO/E (5665-285)	◊	♦	♦
• MVS/SP Version 2 (5665-291 or 5740-XC6) Release 1.3 Vector Facility Enhancement or Release 1.7 or later, and MVS/XA DFP Version 2 (5665-XA2) Release 3 or later, with or without TSO/E (5665-285) Release 3 or later	♦	♦	♦
MVS/ESA: MVS/SP Version 3 (5685-001 or 5685-002), MVS/ESA Version 4 (5695-047 or 5695-048) and MVS/ESA DFP Version 3 (5665-XA3), with or without TSO/E (5665-285) Release 3 or later or TSO/E Version 2 (5685-025)	♦	♦	♦
VM/SP (5664-167) Release 5 or later	◊	◊	♦
VM/SP HPO (5664-173) Release 5 or later	♦	◊	♦
VM/XA SP (5664-308):			
• Release 1.0 or later with bimodal CMS	♦	◊	♦
• Release 2.0 or later with bimodal CMS.	♦	♦	♦
VM/ESA (5684-112):			
• Release 1.0 or later	♦	◊	♦
• with the CMS PRPQ to Support Parallel Processing in VS FORTRAN (5799-DGW)	♦	♦	♦

Legend:

◊	indicates that compilation is supported for the specified feature.
♦	indicates that compilation and execution are supported for the specified feature.

VM/ESA requires VSE/VSAM (5746-AM2) Release 4 (or later) to process VSAM files.

VM/SP requires VSE/VSAM (5746-AM2) Release 1, 2, or 3 to process VSAM files. VM/XA requires VSE/VSAM (5746-AM2) Release 3 to process VSAM files. VSAM support is included with the operating system environment as specified above.

[*] MVS/SP is a trademark of the International Business Machines Corporation.

| Assembler H Version 2 Release 1.0 or IBM High Level Assembler/MVS & VM &
| VSE is required for customization of VS FORTRAN Version 2 in the VM/XA, MVS/XA, and MVS/ESA environments.

Data-in-virtual processing requires MVS/SP Version 2 Release 2 (MVS/XA), MVS/XA DFP Version 2 Release 3 or MVS/SP Version 3 (MVS/ESA), and MVS/ESA DFP Version 3.

Using the interactive debug component requires TSO/E (on MVS) or CMS (on VM). Interactive debugging in full-screen mode requires:

- Under MVS, ISPF Version 2 for MVS (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 Versions 2 for VM (5664-285). For enhanced full-screen functions, 5664-282 is required.
- Under VM/XA, ISPF Version 2 Release 2 for VM (5684-015) with or without ISPF/PDF Version 2 Release 2, for VM (5684-014). For enhanced full-screen functions, 5684-015 is required.
- Under VM/SP Release 6, ISPF Version 2 Release 2.1 for VM (288-5521) is required.
- Under VM/ESA, ISPF Version 3 Release 2 for VM (5684-043) with or without ISPF/PDF Version 3 Release 2 for VM (5684-123). For enhanced full-screen functions, 5684-123 is required.

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

- PDF browse and edit facilities in split-screen mode
- Automatic browse of the interactive debug print file and log at session end
- Start of debugging by means of the interactive debug foreground invocation panel.

Hardware Requirements

Programs compiled with the vectorization option specified are designed to run on the vector facility. All VS FORTRAN Version 2 Release 6 programs can be compiled, and all scalar programs will run on, any IBM System/370*, System/390*, 43xx, 30xx, or 9370 Processor supported by the operating systems listed under the "Software Requirements" section above.

The processor must have sufficient real storage to meet the combined storage requirements of the host operating system and access methods used. There is no impact by VS FORTRAN Version 2 Release 6 on 24-bit virtual constraint.

* System/370 and System/390 are trademarks of the International Business Machines Corporation.

Extended common blocks require program execution:

- Under MVS/ESA on an IBM ES/3090* model S or J/JH, or IBM ES/9000 processor.
- Under VM/ESA on an IBM ES/9000 processor.

Virtual Storage

On MVS, the compiler can be installed either above or below the 16-megabyte line. If the compiler is installed above the 16-megabyte line, approximately 1.8 megabytes of storage within the 16-megabyte addressable area are available for other MVS use. This storage estimate is for the compiler and its work areas only, and does not include space for operating system overhead or other programs. On CMS, shared segments can be used above the 16-megabyte line, however, AIX/370 is not supported above the 16-megabyte line. Because VS FORTRAN Version 2 does no page fixing, the minimum real storage for initiating a job is the real storage requirement. No auxiliary storage is required because 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 to 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, sample installation jobs, and the customization jobs needed to install and customize VS FORTRAN Version 2.

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

For customizing VS FORTRAN Version 2 on a VM/XA, VM/ESA, MVS/XA, or MVS/ESA system, Assembler H Version 2 Release 1.0 or IBM High Level Assembler/MVS & VM & VSE is required.

| With VS FORTRAN Version 2 Release 6, defaults for the VECTOR compiler option
| and its associated suboptions can be set at install time.

Operating System Restrictions

The following are supported only under MVS:

- Multitasking facility
- Data-in-virtual
- Asynchronous input/output.

* ES/3090 is a trademark of the International Business Machines Corporation.

Programs with functions unique to an operating system can be run only on that system. Load modules that have been link-edited on one operating system cannot be run on another operating system.

The CMSBATCH service is *not* supported for a parallel program with the parallel Fortran interface on VM/XA or VM/ESA.

However, with VS FORTRAN Version 2 Release 6, calls to assembler routines that rely on system services are supported with the use of service subroutines to control the parallel execution environment.

The VS FORTRAN Version 2 Release 6 Library must be installed on the system that the parallel program will execute on, because a parallel program can run only in load mode. Because you cannot run a parallel program in link mode, it is impossible to make a complete, self-contained module for a parallel program.

Compatibility between Versions

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. You do not need to recompile main programs and subroutines if your main programs and subroutines specify only dynamic common blocks. If extended common blocks are specified, the user must recompile with the EMODE compile-time options.

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 25.

Error Handling Compatibility

With VS FORTRAN Version 2 Release 6, error message numbers 0 through 499 are reserved. Users can use error message numbers 500 through 899. Programs written for prior releases that used error message numbers 302 through 599 as user-defined error message numbers must be modified.

Object Module Compatibility

Existing CMS and MVS 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. However, the modules must be recompiled by the VS FORTRAN Version 2 compiler or Version 1 Release 3 or later compiler to be compatible, if *both* of the following are true:

- The object modules were produced by the VS FORTRAN Version 1 compiler prior to Release 3.
- The object modules are programs that pass character arguments to subprograms or are subprograms that receive character arguments.

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 6 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 6 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/ NOOCSTATUS and FILEHIST/NOFILEHIST 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 determine properties of connected files.

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, because 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 value of the BLANK or PAD 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 that 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'.
- If run-time option FILEHIST is in effect:
  - The history of a file is used in determining its existence
  - The file definition of a file cannot be changed during run time and have the same results produced as in previous VS FORTRAN Version 2 releases.
- If run-time option NOFILEHIST is in effect:
  - The history of a file is disregarded in determining its existence
  - A file whose definition is changed during run time is treated as if it were being opened for the first time.

Load Module Compatibility

Existing VS FORTRAN Version 1 and VS FORTRAN Version 2 load modules are compatible with the VS FORTRAN Version 2 Release 6 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 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 Release 6 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.
  - Load modules created under MVS cannot be run under CMS using the OSRUN command.
- 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 25.

Parallel and Vector Considerations

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) and VECTOR(NOSPRECOPT) (and, if the Version 1 library is used, the VECTOR(NOINTRINSIC)) compiler options; when these sub-options are used, however, some vectorization may be inhibited.

The instructions used by the VS FORTRAN Version 2 Release 6 compiler to generate more efficient vectorized code may produce different results from the same program compiled with prior releases of the Version 2 compiler.

For parallel processing, results may differ when you process data in parallel and in serial order, because with parallel processing, the sequence of data will change from one run to another of a program. In addition, different units are associated with different types of tasks. For instance, data striping I/O is limited to the units associated with the *root task*, the parallel task where program processing begins, of a parallel program. Any units accessed within an *originated task*, a parallel task that is created using an ORIGINATE statement, are separate from units accessed within another originated task.

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[*], System/370, and System/390 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 on MVS and CMS systems, 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.

Interactive debug cannot be used in any program that specifies extended common blocks, Integer*8 data, Integer*1 data, Logical*8 data, Logical*2 data, Pointer data, or UnSigned*1 data and can be used only with serial portions of a parallel program run with the PARALLEL(1) run-time option.

[*] System/360 is a trademark of the International Business Machines Corporation.

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 VS FORTRAN Version 2 Release 3 or later 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 later releases.

License

Separate licenses for the Compiler and Library and Interactive Debug (5668-806), the Compiler and Library (5688-087), 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 12 months' written notice.

Appendix A. Summary of the VS FORTRAN Version 2 Language

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.

Beginning with Release 3, VS FORTRAN Version 2 supports the SAA 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.

Source Language Flaggers

The VS FORTRAN Version 2 compiler can flag Fortran statements that do not conform to the syntax of the full or subset ANS FORTRAN Standard (FORTRAN 77). The FIPS compile-time option specifies whether this flagging is to be performed.

The VS FORTRAN Version 2 compiler can also flag Fortran statements that are not a part of the Systems Application Architecture (SAA) Common Programming Interface (CPI) FORTRAN language definition. The SAA compile-time option specifies whether this flagging is to be performed.

For more information about the compile-time options, see the *VS FORTRAN Version 2 Programming Guide*.

Major Elements of the VS FORTRAN Version 2 Language

Figure 5 on page 30 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
- | An element is flagged for not conforming to the FORTRAN 77 standard (FIPS), if "No" or "Yes - Ext" appears in the corresponding column
- | An element is flagged for not conforming to the Systems Application Architecture FORTRAN definition (SAA), if "No" appears in the corresponding column.

Only high-level aspects of the language are shown. (For complete information, refer to the *VS FORTRAN Version 2 Language and Library Reference*).

Figure 5 (Page 1 of 5). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In LANGLVL (77)?	In LANGLVL (66)?	In SAA?
Ampersand (&)	Allowed as special character	No	Yes - Ext	No
ALLOCATE	Allocate storage	Yes - Ext	No	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
	Assign values to unsigned data items	Yes - Ext		
Asynchronous I/O	Read/write in asynchronous mode	Yes - Ext	Yes - Ext	No
AT	Specifies beginning of debugging packet	Yes - Ext	Yes - Ext	No
AUTOMATIC	Controls storage class	Yes - Ext	No	No
BACKSPACE	Repositions file at previous record	Yes - Std	Yes - Std	Yes
Binary constants	For binary data values	Yes - Ext	No	No
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 nonblank 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
DEALLOCATE	Deallocate storage	Yes - Ext	No	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

Figure 5 (Page 2 of 5). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In LANGLVL (77)?	In LANGLVL (66)?	In SAA?
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
DOAFTER	Identifies a block of statements that each virtual processor must process once after running the iterations of the loop.	Yes - Ext	Yes - Ext	No
DOBEFORE	Identifies a block of statements that each virtual processor must process once before running the iterations of the loop.	Yes - Ext	Yes - Ext	No
DOEVERY	Identifies a block of statements that make up the body of the loop.	Yes - Ext	Yes - Ext	No
DO WHILE	Initiates processing of program loops based on evaluation of a logical expression	Yes - Ext	No	No
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 , PARALLEL 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
END SECTIONS	Indicates the end of a group of sections	Yes - Ext	Yes - Ext	No
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
EXIT	Stops the running of a parallel loop	Yes - Ext	Yes - Ext	No
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 hex data values	Yes - Ext	Yes - Ext	No

Figure 5 (Page 3 of 5). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In LANGLVL (77)?	In LANGLVL (66)?	In SAA?
Hollerith constants	For initializing variables	Yes - Ext	Yes - Std	No
	As arguments	Yes - Ext	Yes - Ext	No
	As character constants 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
	Classifies groups of variables for storage class	Yes - Ext	Yes - Ext	No
INCLUDE	Copies prewritten source statements into program	Yes - Ext	Yes - Ext	Yes
INQUIRE	Retrieves information about a file	Yes - Std	No	Yes
Integer data type	For integer numbers	Yes - Std	Yes - Std	Yes
	Integer*1 (1 byte)	Yes - Ext	Yes - Ext	No
	Integer*2 (2 byte)	Yes - Ext	Yes - Ext	Yes
	Integer or Integer*4 (4 byte)	Yes - Std	Yes - Std	Yes
	Integer*8 (8 byte)	Yes - Ext	Yes - Ext	No
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
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
LOCAL	Specifies variables, instances of which are provided to each virtual processor	Yes - Ext	Yes - Ext	No
Logical data type	For true/false value	Yes - Std	Yes - Std	Yes
	Logical*1 (1 byte)	Yes - Ext	Yes - Ext	No
	Logical*2 (2 byte)	Yes - Ext	Yes - Ext	No
	Logical or Logical*4 (4 byte)	Yes - Std	Yes - Std	Yes
	Logical*8 (8 byte)	Yes - Ext	Yes - Ext	No
Mixed-mode expressions	Allow mixing of data types	Yes - Std	Yes - Ext	Yes
NAMELIST	Read/write referencing named list	Yes - Ext	Yes - Ext	No
NULLIFY	Nullifies a pointer variable	Yes - Ext	No	No
Octal constants	For octal data values	Yes - Ext	No	No
OPEN	Connects files to a program; error routines can be specified	Yes - Std	No	Yes

Figure 5 (Page 4 of 5). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In LANGLVL (77)?	In LANGLVL (66)?	In SAA?
ORIGINATE	Creates a new parallel task	Yes - Ext	Yes - Ext	No
PARALLEL CALL	Specifies a subroutine to run in parallel	Yes - Ext	Yes - Ext	No
PARALLEL DO	Begins a parallel loop	Yes - Ext	Yes - Ext	No
PARALLEL SECTIONS	Begins a group of parallel sections	Yes - Ext	Yes - Ext	No
PARAMETER	Establishes names for constants	Yes - Std	No	Yes
PAUSE	Suspends program processing temporarily	Yes - Std	Yes - Std	Yes
POINTER	Defines a pointer variable	Yes - Ext	No	No
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	Yes - Ext	No
Relational Operators	Compare two arithmetic expressions	Yes - Std	Yes - Std	Yes - Std
Relational Operator Extensions	Compare two arithmetic expressions	Yes - Ext	Yes - Ext	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
SCHEDULE	Assigns a subroutine to an originated task	Yes - Ext	Yes - Ext	No
SECTION	Indicates the beginning of a group of statements to be processed as a parallel thread	Yes - Ext	Yes - Ext	No
Sequential I/O	Read/write sequential files	Yes - Std	Yes - Std	Yes
Statement functions	Allow convenient programming of expressions	Yes - Std	Yes - Std	Yes
STATIC	Controls storage class	Yes - Ext	No	No
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
TERMINATE	Deletes an originated task	Yes - Ext	Yes - Ext	No
TRACE ON/OFF	Traces specific portions of a program	Yes - Ext	Yes - Ext	No
Underscore character (_)	Can be used in names	Yes - Ext	Yes - Ext	Yes
Unsigned data type	Allows natural numbers	Yes - Ext	No	No
VSAM I/O	Supports ESDS, RRDS, and KSDS files	Yes - Ext	No	No
WAIT FOR ALL CALLS	Causes the program to wait for all subroutines to finish processing before continuing	Yes - Ext	Yes - Ext	No
WAIT FOR ALL TASKS	Waits for all parallel tasks to complete assigned work	Yes - Ext	Yes - Ext	No

Figure 5 (Page 5 of 5). Major Elements of the VS FORTRAN Version 2 Language

Statement or Feature	Use	In LANGLVL (77)?	In LANGLVL (66)?	In SAA?
WAIT FOR ANY TASK	Waits for any parallel task to complete assigned work	Yes - Ext	Yes - Ext	No
WAIT FOR TASK	Waits for completion of assigned work in a specific parallel task	Yes - Ext	Yes - Ext	No
WRITE	Writes a record into a file	Yes - Std	Yes - Std	Yes

Appendix B. Supplied Functions and Subroutines

The VS FORTRAN Version 2 program product supplies many predefined, built-in functions and subroutines. These can help perform a wide variety of programming tasks, such as mathematical calculation, character and bit manipulations, and error-handling operations.

This appendix lists the necessary 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.

Figure 6. Mathematical Functions Supplied

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
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

Additional Functions

Figure 7. Additional Functions Supplied

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 or XOR
	Logical complement	NOT
	Bit test	BTEST
	Bit set to 1	IBSET
	Bit set to 0	IBCLR
	Bit extraction	IBITS
	Left or right shift	ISHFT
	Left shift	LSHIFT
	Right shift	RSHIFT
	Circular shift	ISHFTC
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
Parallel Processing	Calculate number of virtual processors	NPROCS
	Obtain the specified lock	PLCOND
Storage Allocation	Obtain address of variable	LOC
	Test if pointer has allocated storage	ALLOCATED

Utility Routines

Utility routines 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, and allow the programmer to manipulate double-byte character strings and to specify certain attributes for dynamically allocated files or data sets. All of these prewritten subroutines are accessed through a CALL statement.

The following routines are provided:

Mathematical Exception Subroutines

DVCHK	Tests for a divide-check exception and returns a value indicating the condition that exists.
OVERFL	Tests for an exponent overflow or underflow exception and returns a value indicating the condition that exists.
XUFLOW	The XUFLOW subroutine changes the exponent underflow mask in the program mask to allow or suppress program interrupts that could result from an exponent underflow exception.

Storage Dump Subroutines

DUMP|PDUMP Dumps a specified area of storage and either terminates (DUMP) or continues processing (PDUMP).

CDUMP|CPDUMP Dynamically dumps a specified area of storage.

SDUMP Provides a symbolic dump of variables.

Return Code Subroutines

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.

Termination Subroutines

EXIT Terminates processing by returning control to the operating system.

SYSABD|SYSABN Causes abnormal termination of the job, either with or without a dump.

File Service Subroutines

FILEINF Sets up file characteristics before processing an OPEN or INQUIRE statement.

UNTANY Within a specified range, UNTANY will return the lowest unit number of an available unit regardless of the file definitions in effect.

UNTNOFD Within a specifed range, UNTNOFD will return the lowest unit number that does not have a user-specified file definition associated with it (with a ddname of FT*nn*F001 or FT*nn*K01, where *nn* is the unit number) and that is available.

Date/Time Subroutines

CLOCK|CLOCKX Provides the current time.

CPUTIME Allows you to determine how much CPU time a program or a portion of a program has used. It cannot be used in a parallel program.

DATIM|DATIMX Provides the current day and time.

Other Service Subroutines

ASSIGNM	Moves a character string that contains both single- and double-byte characters to a character variable, substring, or array element.
ARGSTR	Retrieves command-line user parameters.
MVBITS	Assigns a bit subfield of an integer value to a bit subfield of another integer value.

Data In Virtual Subroutines

DIVCML	Obtains the length of a dynamic/extended common (varying-view).
DIVINF	Associates a data object with a dynamic/extended common for reading, or for reading and writing (fixed-view).
DIVINV	Associates a data object with a data object ID for reading, or for reading and writing (varying-view).
DIVRES	Resets the data in the dynamic/extended common to the values in the mapped part of the data object, eliminating any changes that have been made in the dynamic/extended common.
DIVSAV	Saves changes made to the data object in the dynamic/extended common that has been accessed for READWRITE.
DIVTRF	Terminates the association of the data object to the dynamic/extended common (fixed-view).
DIVTRV	Terminates the association between the data object ID and the data object (varying-view).
DIVVWF	Establishes the part of the data object that the dynamic/extended common will map (fixed-view).
DIVVWV	Establishes the part of the data object that the dynamic/extended common will map (varying-view).

Multitasking Facility (MTF) Subroutines

DSPTCH	The DSPTCH subroutine schedules a parallel subroutine for execution in a subtask. You may call DSPTCH as many times as necessary. Eventually, you must call SYNCRO to wait for all the parallel subroutines to finish executing.
NTASKS	Returns the number of subtasks specified with the AUTOTASK keyword in the PARM parameter of the EXEC statement for the job step.
SHRCOM	The SHRCOM subroutine allows you to designate a dynamic common block as shareable among the main task program and the parallel subroutines.
SYNCRO	The SYNCRO subroutine causes the main task program to wait until all scheduled parallel subroutines have completed execution.

Parallel Subroutines

PEORIG	Creates an event and returns an identifier for the event.
PEPOST	Posts the specified event.
PETERM	Deletes a specified event.

PEWAIT	Causes the calling parallel thread to wait until the specified event's *post-count* (or *wait-count* if the post-count is equal to zero) is reached.
PLFREE	Releases the lock that you specify.
PLLOCK	Obtains the lock that you specify
PLORIG	Creates and initializes a lock and returns an identifier for the lock.
PLTERM	Deletes the specified lock.

Error-Handling Subroutines

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

- Operation
- Fixed-point 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. You can set the default options at installation for most of these options.

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.

| **DDIM | NODDIM**
| Indicates that the pointee arrays that specify object-time dimensions are to
| have those dimensions evaluated dynamically at each element reference; these
| are known as dynamically dimensioned arrays.

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

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

| **DYNAMIC({ * | name [,...] })**
| Provides dynamic loading of user subroutines or functions during program exe-
| cution. Dynamic loading allows these routines to be contained in separate load
| modules and to load these modules when they are needed during program
| execution.

EC (* | *name1***,***name2***,...)**
Defines the names of common blocks to be dynamically allocated as extended common blocks that reside in the storage space available through the ESA environment.

EMODE | NOEMODE
Specifies whether the code that is compiled for a subroutine or function can receive arguments that reside in an extended common block.

FIPS (S | F) | NOFIPS
Specifies whether standard language flagging is to be performed, and, if performed, specifies 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.

| **HALT(level)**
| Causes termination of a compilation after any compiler phase, if diagnostics at
| or above a specified level have been generated. Further compiler processing
| is prevented.

ICA [(
[USE (*name1*,*name2*,**...)]**
[UPDATE (*name***)]**
[DEF (*nameA*,*nameB*,**...)]**
[MXREF | NOMXREF]
[CLEN | NOCLEN]
[CVAR | NOCVAR]
[MSG ({ NEW | NONE | ALL })]
[MSGON (*number1*,*number2*,**...) | MSGOFF (***number1*,*number2*,**...)]**
[RCHECK | NORCHECK]
)]
| 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 inline 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.

```
PARALLEL [ (
      [ REPORT (LIST | XLIST | TERM | SLIST | STAT) | NOREPORT ]
      [ LANGUAGE | NOLANGUAGE ]
      [ AUTOMATIC | NOAUTOMATIC ]
      [ REDUCTION | NOREDUCTION ]
      [ TRACE | NOTRACE ]
      [ ANZCALL | NOANZCALL ]
    ) ]
  | NOPARALLEL
```
Controls the parallel suboptions used by the compiler when generating code for DO loops and PARALLEL DO, PARALLEL SECTIONS, and PARALLEL CALL statements. It is required if parallel language constructs are present.

PTRSIZE(4|8)
Sets the default length for pointer variables.

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 that are not part of Systems Application Architecture.

SC (* | *name1*,*name2*,...)
Defines the names of common blocks to be compiled as static common blocks.

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 | SLIST | STAT) | NOREPORT]
[INTRINSIC | NOINTRINSIC]
[IVA | NOIVA]
[REDUCTION | NOREDUCTION]
[SIZE ({ ANY | LOCAL | *number* **})]**
| **[MODEL (level)]**
| **[SPRECOPT | NOSPRECOPT]**
| **[ANZCALL | NOANZCALL]**
| **[CMPLXOPT | NOCMPLXOPT]**
)]
| NOVECTOR
Invokes the vectorization process, which produces programs that can exploit the speed of the 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. You can set the default values for most of these options at installation time.

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).

CNVIOERR | NOCNVIOERR
Specifies whether input conversion errors will be treated as input/output errors.

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.

ERRUNIT(*number*)
Identifies the unit number to which run-time error information is directed.

FAIL (ABEND | RC | ABENDRC)
Indicates how to terminate unsuccessfully executed programs.

ECPACK | NOECPACK
Specifies whether a data space should be filled with as many extended common blocks as possible before a new data space is allocated.

INQPCOPN | NOINQPCOPN
Determines whether a unit is connected to a file when executing an INQUIRE by unit.

IOINIT | NOIOINIT
Specifies whether the error message unit is to be opened 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'`.

PRTUNIT(*number*)
Identifies the unit number that is to be used for output statements that do not specify a unit number.

PTRACE[(options)] | NOPTRACE
Enables the Parallel Trace Facility and causes the Trace Log File to be initialized.

PUNUNIT(*number*)
Identifies the unit number that is to be used for PUNCH statements that do not specify a unit number.

RECPAD] (ALL) [| NORECPAD
Specifies whether a formatted input record is padded with blanks when an input list and format specification require more data from the record than the record contains.

RDRUNIT(*number*)
Identifies the unit number that is to be used for READ statements that do not specify a unit number.

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 on MVS and CMS systems. 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 processing 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 that 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 online 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 nonvectorized.
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 that 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 programs are:

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

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 for CMS and MVS (SC26-4222) gives guidance information on designing, coding, compiling, executing, and debugging VS FORTRAN programs under CMS or MVS operating systems.

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) summarizes source language and compile-time options in a convenient pocket-sized reference booklet.

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) briefly describes the VS FORTRAN Version 2 Release 6 Program Product, and serves as the warranty.

VS FORTRAN Version 2 Master Index and Glossary (SC26-4603) contains both combined subject indexes from the VS FORTRAN library, including references to specific books and pages, and combined glossaries formerly contained in the VS FORTRAN library.

VS FORTRAN Version 2 Migration from the Parallel FORTRAN PRPQ (SC26-4686) describes the language differences between the PRPQ program offering and VS FORTRAN Version 2 Release 6. It also contains some considerations for recompiling and running programs that were written with PRPQ, on VS FORTRAN Version 2 Release 6.

VS FORTRAN Version 2 Release 6 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

We'd Like to Hear from You

VS FORTRAN Version 2
General Information
Release 6

Publication No. GC26-4219-10

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Fax—Use the Readers' Comments form on the next page and fax it to this U.S. number: 800-426-7773.
- Electronic mail—Use one of the following network IDs:
  - IBMLink: HLASMPUB at STLVM27
  - Internet: COMMENTS@VNET.IBM.COM

  Be sure to include the following with your comments:

  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

VS FORTRAN Version 2
General Information
Release 6

Publication No. GC26-4219-10

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments
GC26-4219-10

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Department J58
International Business Machines Corporation
PO BOX 49023
SAN JOSE CA 95161-9945

Fold and Tape **Please do not staple** Fold and Tape

GC26-4219-10

Cut or Fold
Along Line

File Number: S370-20
Program Number: 5668-805
5668-806
5688-087

Printed in the U.S.A.

The VS FORTRAN Version 2 Library

LY27-9516	Diagnosis Guide
GC26-4219	General Information
SC26-4340	Installation and Customization for MVS
SC26-4339	Installation and Customization for CMS
SC26-4420	Installation and Customization for AIX/370
SC26-4223	Interactive Debug Guide and Reference
SC26-4221	Language and Library Reference
GC26-4225	Licensed Program Specifications
SC26-4603	Master Index and Glossary
SC26-4686	Migration from the Parallel FORTRAN PRPQ
SC26-4741	Programming Guide for AIX/370
SC26-4222	Programming Guide for CMS and MVS
SX26-3751	Reference Summary

GC26-4219-10

Spine information:

IBM VS FORTRAN Version 2 General Information *Release 6*