Reference Summary

Release 6
Reference Summary

Release 6
Before using this information and the product it supports, be sure to read the general information under "Notices" on page iv.

Eighth Edition (November 1993)

This edition replaces and makes obsolete the previous edition, SX26-3751-06.

This edition applies to VS FORTRAN Version 2 Release 6, Program Numbers 5668-805, 5668-087, 5667-806, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

Specific changes for this edition are indicated by a vertical bar to the left of the change. A vertical bar to the left of a figure caption indicates that the figure has changed. Editorial changes that have no technical significance are not noted.

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.

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

A form for 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.

© Copyright International Business Machines Corporation 1986, 1993. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
# Contents

 Notices .............................................. iv
 Programming Interface Information .............. iv
 Trademarks and Service Marks .................... iv

 **Compiler and Library Information** .................. 1
 Arithmetic Expressions ............................. 1
 Relational Expressions ............................. 1
 Logical Expressions .................................. 2
 Hierarchy of Operations ............................ 2
 Required Order of Statements and Comments ......... 3
 Language Statement Categories ...................... 4
 Statement Syntax .................................... 6
 Parallel Statement Categories ...................... 26
 Parallel Statement Syntax ........................... 27
 Parallel Task Management Statements ............... 27
 Parallel Loop Statements ........................... 28
 Parallel Sections Statements ........................ 29
 Parallel Call Statements ........................... 30
 Compile-Time Options ................................ 31
 Conflicting Compile-Time Options ................... 36
 Compiler Directives ................................ 37
 Parallel and Vector Directives ...................... 38
 Run-Time Options ................................... 39
 Service Subroutines ................................ 41
 Parallel Library Event Service Subroutines ......... 45
 Parallel Library Lock Service Subroutines and Function 46
 Parallel Function ................................... 47
 Data-in-Virtual Subroutines ......................... 48
 Multitasking Facility (MTF) Subroutines .......... 50
 Error-Handling Subroutines ......................... 51

 **Intrinsic Functions** ............................... 52

 **Interactive Debug Commands** ...................... 58
 Interactive Debug Command Categories .............. 59
 Interactive Debug Command Syntax .................. 60
 Format and Dump Codes for the AUTOLIST and LIST Commands 74
 Valid SET Command Assignments ................... 75
Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM’s intellectual property rights or other legally protectible rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user’s responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

Programming Interface Information

This reference is intended to help you create application programs using the VS FORTRAN Version 2 licensed program. This reference documents General-Use Programming Interface and Associated Guidance Information provided by VS FORTRAN Version 2.

General-Use programming interfaces allow the customer to write programs that obtain the services of VS FORTRAN Version 2.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
SAA
Systems Application Architecture
3090
Compiler and Library Information

Arithmetic Expressions

<table>
<thead>
<tr>
<th>Arithmetic Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>+</td>
<td>Addition (or unary plus)</td>
</tr>
<tr>
<td>−</td>
<td>Subtraction (or unary minus)</td>
</tr>
</tbody>
</table>

Relational Expressions

<table>
<thead>
<tr>
<th>Standard Relational Operator</th>
<th>VS Fortran Relational Operator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>.GT.</td>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>.GE.</td>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>.LT.</td>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>.LE.</td>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>.EQ.</td>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>.NE.</td>
<td>/= &lt;&gt;</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>
Logical Expressions

<table>
<thead>
<tr>
<th>Logical Operator</th>
<th>Use</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NOT.</td>
<td>.NOT.A</td>
<td>If A is true, then .NOT.A is false; if A is false, then .NOT.A is true.</td>
</tr>
<tr>
<td>.AND.</td>
<td>A.AND.B</td>
<td>If A and B are both true, then A.AND.B is true; if either A or B or both are false, then A.AND.B is false.</td>
</tr>
<tr>
<td>.OR.</td>
<td>A.OR.B</td>
<td>If either A or B or both are true, then A.OR.B is true; if both A and B are false, then A.OR.B is false.</td>
</tr>
<tr>
<td>.XOR.</td>
<td>A.XOR.B</td>
<td>If either A or B is true, the A.OR.B is true; if both A and B are false, then A.OR.B is false.</td>
</tr>
<tr>
<td>.EQV.</td>
<td>A.EQV.B</td>
<td>If A and B are both true or both false, then A.EQV.B is true; otherwise it is false.</td>
</tr>
<tr>
<td>.NEQV.</td>
<td>A.NEQV.B</td>
<td>If A and B are both true or both false, then A.NEQV.B is false; otherwise it is true.</td>
</tr>
</tbody>
</table>

Hierarchy of Operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Hierarchy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation of functions</td>
<td>1st (highest)</td>
</tr>
<tr>
<td>Exponentiation (**)</td>
<td>2nd</td>
</tr>
<tr>
<td>Multiplication and division (× and ÷)</td>
<td>3rd</td>
</tr>
<tr>
<td>Unary addition and subtraction (+ or -)</td>
<td>4th</td>
</tr>
<tr>
<td>Binary addition and subtraction (+ and -)</td>
<td>5th</td>
</tr>
<tr>
<td>Concatenation (/)</td>
<td>6th</td>
</tr>
<tr>
<td>Relational operators (.GT.,.GE.,.LT.,.LE.,.EQ.,.NE.)</td>
<td>7th</td>
</tr>
<tr>
<td>.NOT.</td>
<td>8th</td>
</tr>
<tr>
<td>.AND.</td>
<td>9th</td>
</tr>
<tr>
<td>.OR.</td>
<td>10th</td>
</tr>
<tr>
<td>.EQV. or .NEQV. or .XOR.</td>
<td>11th</td>
</tr>
</tbody>
</table>
### Required Order of Statements and Comments

**Figure 1. Order of Statements and Comment Lines**

<table>
<thead>
<tr>
<th>Comment Lines</th>
<th>PROGRAM, FUNCTION, SUBROUTINE, or BLOCK DATA Statement</th>
<th>IMPLICIT NONE Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FORMAT and ENTRY Statements</td>
<td>IMPLICIT Statements</td>
</tr>
<tr>
<td></td>
<td>PARAMETER Statements</td>
<td>Other Specification Statements</td>
</tr>
<tr>
<td></td>
<td>DATA Statements</td>
<td>Statement Function Statements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Executable Statements and Parallel Statements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>END Statement</td>
</tr>
</tbody>
</table>

Compiler and Library Information 3
Language Statement Categories

In the following statements, extensions to the Fortran standard language are printed in color.

Assignment Statements
- Arithmetic
- Character
- Logical
- ASSIGN

Control Statements
- CALL
- CONTINUE
- DO
- DO WHILE
- END
- END DO
- GO TO
- IF
  (ELSE, ELSE IF, END IF)
- PAUSE
- RETURN
- STOP

Data Statements
- DATA

Static Debug Statements
- AT
- DEBUG
- DISPLAY
- END DEBUG
- TRACE OFF
- TRACE ON

Input/Output Statements
- BACKSPACE
- CLOSE
- DELETE
- ENDFILE

Input/Output Statements (continued)
- FORMAT
- INQUIRE
- OPEN
- PRINT
- READ
-REWIND
- REWRITE
- WAIT
- WRITE

Program Statement
- PROGRAM

Specification Statements
- AUTOMATIC
- COMMON
- DIMENSION
- EQUIVALENCE
- Explicit type:
  - CHARACTER,
  - COMPLEX,
  - DOUBLE PRECISION,
  - INTEGER,
  - LOGICAL, and REAL
  - DOUBLE COMPLEX
  - UNSIGNED
  - BYTE
- EXTERNAL
- IMPLICIT
- INTRINSIC
- NAMELIST
- PARAMETER
- POINTER
- SAVE
- STATIC
Subprogram Statements
  BLOCK DATA
  ENTRY
  FUNCTION
  SUBROUTINE

<table>
<thead>
<tr>
<th>Allocation Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALLOCATE</td>
</tr>
<tr>
<td>DEALLOCATE</td>
</tr>
<tr>
<td>NULLIFY</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parallel Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>See page 26</td>
</tr>
</tbody>
</table>
Statement Syntax

In the following statements, extensions to the Fortran standard language are printed in color.

- **ALLOCATE Statement**
  
  ALLOCATE (pointee1 [, pointee2 ...], STAT=stat)

  Obtains space for a pointee array.

- **ASSIGN Statement**
  
  ASSIGN stl TO i

  Assigns a number (statement label) to an integer variable.

- **Assignment Statement**
  
  \[ a = b \]

  Evaluates the expression to the right of the equal sign, replacing the current value of the variable, array element, character substring, or character variable to the left of the equal sign with the expression’s value.

- **AT Statement**
  
  AT stl

  Identifies the beginning of a debug packet and indicates the point in the program at which debugging statements are to be inserted.

- **AUTOMATIC Statement**
  
  AUTOMATIC name1 [, name2 ...]

  Names variables and arrays to be in the automatic storage class.

- **BACKSPACE Statement**
  
  BACKSPACE un

  BACKSPACE
Statement Syntax

```
([UNIT=] un
 [,IOSTAT=ios]
 [,ERR=ist])
```

Positions a sequentially-accessed file to the beginning of the Fortran record last written or read, or repositions the file to the beginning of the preceding record.

**BLOCK DATA Statement**

```
BLOCK DATA [name]
```

Initializes values for variables and array elements in named common blocks.

**CALL Statement**

```
CALL name ([([arg1 [,arg2]...])]
```

Evaluates actual arguments that are expressions, passes actual arguments that will be associated with dummy arguments defined in the subroutine, and transfers control to the subroutine.

**CLOSE Statement**

```
CLOSE
   ([UNIT=] un [,ERR=ist]
    [,STATUS=sta]
    [,IOSTAT=ios])
```

Disconnects a unit.

**COMMON Statement**

```
COMMON [([name1])] list1 [([|]([name2])list2...]
```

Allows two or more program units to share storage and to specify the names of variables and arrays that are to occupy the area.

**CONTINUE Statement**

```
CONTINUE
```

Labels a position in a program (can designate the end of a DO loop).
Statement Syntax

DATA Statement
DATA list1 /clist1/ [.][list2 /clist2/...]
Defines initial values of variables, array elements, arrays, and substrings.

DEALLOCATE Statement
DEALLOCATE (pointee1[,pointee2...] STAT=stat)
Releases space used for a pointee array and resets the associated pointer variable to the unassigned state.

DEBUG Statement
DEBUG option1 [,option2...]
Sets the conditions for operation of the debug facility and designates debugging operations that apply to the entire program unit.

DELETE Statement
DELETE un
DELETE (UNIT=) un
[ERR=stl]
[IOSTAT=ios]
Removes a record from a file connected for keyed access.

DIMENSION Statement
DIMENSION a1 (dim1) [,a2 (dim2)...
Specifies the name and dimensions of an array.

DISPLAY Statement
DISPLAY list
Displays data in NAMELIST output format.
Statement Syntax

**DO Statement**

DO \([stl [,]] i=1, e2 [,e3]\)

Controls the processing of the statements that follow it, up to and including the statement that denotes the end of the DO loop.

**DO WHILE Statement**

DO \([stl [,]] WHILE (m)\)

Controls the processing of the statements that follow it, up to and including the terminating statement.

**END Statement**

END

Terminates a main program or a function, subroutine or block data subprogram.

**END DEBUG Statement**

END DEBUG

Terminates the last debug packet for the program.

**END DO Statement**

END DO

Terminates the range of a DO WHILE loop (may be used to terminate the range of a DO loop).

**ENDFILE Statement**

ENDFILE \(un\)

ENDFILE \([\text{UNIT=} un \newline \text{[,ERR=} stl \newline \text{[,IOSTAT=} ios}]\)

Writes an end-of-file record on a sequentially accessed external file.
Statement Syntax

ENTRY Statement

ENTRY name [(arg1 [arg2]...)]

Names a place in a subroutine or function subprogram that can be used in a CALL statement or as a function reference.

EQUIVALENCE Statement

EQUIVALENCE (list1) [, (list2)...]

Permits the sharing of data storage within a single program unit.

Explicit Type Statement

  type name1 [,name2...

  type*len name1*len(dim) [,name2*len(dim)...

  Specifies the type and length of variables, arrays, and user-supplied functions, specifies the dimensions of an array, and assigns initial data values for variables and arrays.

EXTERNAL Statement

EXTERNAL name1 [,name2...

Identifies a user-supplied subprogram name and permits such a name to be used as an actual argument.

FORMAT Statement

FORMAT (f1 [,f2...])

Specifies the structure of Fortran records and the form of the data fields within the records.
### Statement Syntax

#### Format Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Integer data editing</td>
</tr>
<tr>
<td>F</td>
<td>Real data editing</td>
</tr>
<tr>
<td>D</td>
<td>Real, complex, or double-precision data editing</td>
</tr>
<tr>
<td>E</td>
<td>Real, complex, double- or extended-precision data editing</td>
</tr>
<tr>
<td>Q</td>
<td>Extended-precision data editing</td>
</tr>
<tr>
<td>G</td>
<td>Real, integer, or logical data transmission</td>
</tr>
<tr>
<td>P</td>
<td>Scale factor specification</td>
</tr>
<tr>
<td>Z</td>
<td>Hexadecimal data transmission</td>
</tr>
<tr>
<td>L</td>
<td>Logical variable transmission</td>
</tr>
<tr>
<td>A</td>
<td>Character data transmission</td>
</tr>
<tr>
<td>X</td>
<td>Skipping characters</td>
</tr>
<tr>
<td>T</td>
<td>Data transmission, start</td>
</tr>
<tr>
<td>TL</td>
<td>Data transmission, starts number of characters to left</td>
</tr>
<tr>
<td>TR</td>
<td>Data transmission, starts number of characters to right</td>
</tr>
</tbody>
</table>

#### Group

Repeats a set of format codes.

- S: Plus character control restoration
- SP: Plus character control production
- SS: Plus character control cessation
- BN: Blanks ignored
- BZ: Blanks treated as zeros
- Slash: Record termination
- Colon: Format control termination
- B: Binary data transmission
- O: Octal data transmission
- $: End-of-record suppression

---

#### FUNCTION Statement

- **[type]** FUNCTION **name** ([**arg1**, **arg2**, ...])
- **[type*len]** FUNCTION **name*len** ([**arg1**, **arg2**, ...])

Identifies a function subprogram.

---

#### Assigned GO TO Statement

- GO TO **i** [,**i**,**stl1**, **stl2**, **stl3**, ...]

Transfers control to a statement label depending on the current assignment of **i**.
Statement Syntax

Computed GO TO Statement

GO TO (stl1 [,stl2 [,stl3]... [,] m)

Transfers control to a statement label depending on the current value of m.

Unconditional GO TO Statement

GO TO stl

Transfers control to the statement specified by the statement label.

Arithmetic IF Statement

IF (m) stl1, stl2, stl3

 Transfers control to a statement label depending on the value of the arithmetic expression m.

Block IF Statement

IF (m) THEN

Controls processing sequence.

ELSE Statement

ELSE

Indicates statements that are processed if the preceding block IF or ELSE IF condition is evaluated to be false.

ELSE IF Statement

ELSE IF (m) THEN

Indicates statements that are processed if the preceding block IF condition is evaluated to be false.

END IF Statement

END IF

Concludes an IF-block.
Statement Syntax

Logical IF Statement

IF (m) st

Evaluates a logical expression and processes or skips a statement, depending on whether the value of the expression is true or false.

IMPLICIT Statement

IMPLICIT type (a [,a]...) [,type(a [,a]...)...]

IMPLICIT type*len (a [,a]...) [,type*len(a [,a]...)...]

IMPLICIT NONE

Confirms or changes the default implied types, or voids implied typing altogether; type can be AUTOMATIC or STATIC.

INQUIRE by File

INQUIRE

(FILE=fn
 [,specifier[,specifier][,...]])

Allows you to determine file existence, connection status and other properties of a named file. The list of possible specifiers follows INQUIRE by Unnamed File statement on page 14

INQUIRE by Unit

INQUIRE

([UNIT=] un
 [,specifier[,specifier][,...]])

Allows you to determine the existence of a unit, whether the unit is connected to a file, and, if the unit is connected, what the properties are of the unit and file connection. The list of possible specifiers follows INQUIRE by Unnamed File statement on page 14
Statement Syntax

INQUIRE by Unnamed File

INQUIRE
  ([UNIT=] un,
   FILE=fn
   [,specifier[,specifier]...])

Allows you to determine the file existence and connection status for an
unnamed file, as well as other properties of the file. Following is the list of
possible specifiers.

ACCESS=acc    LASTKEY=lky
ACTION=act    LASTRECL=itr
BLANK=blk     NAME=nam
CHAR=chr      NAMED=nmd
| DELIM=dlm    NEXTREC=nxr
DIRECT=dir    NUMBER=num
ERR=err       OPENED=opn
| EXIST=exs    PAD=pad
FORM=frm      PASSWORD=pwd
| FORMATTED=fmt POSITION=pos
IOSTAT=ios    READ=r
KEYED=kyd     READWRITE=rw
KEYEND=ken    RECL=r
KEYID=kid     SEQUENTIAL=seq
KEYLENGTH=kle  UNFORMATTED=unf
KEYSTART=kst  WRITE=wri

INTRINSIC Statement

INTRINSIC name1 [,name2...]

Identifies a name as representing a procedure supplied by VS FORTRAN
Version 2, and permits a specific intrinsic function name to be used as an
actual argument.

NAMELIST Statement

NAMELIST /name1/ list1 [/name2/ list2...]

Specifies one or more lists or names for use in READ and WRITE state-
ments.
Statement Syntax

### NULLIFY Statement

**NULLIFY** *(pointer1 [, pointer2 ...])* 

Disassociates a pointer variable from an addressed pointee variable and sets the value of the pointer variable to the unassigned state.

### OPEN Statement

**OPEN**

```
(UNIT=) un [,ERR=err] [,STATUS=sta] 
[FILE=fn] [,ACCESS=acc] [,BLANK=blk] 
[CHAR=chr] 
[FORM=frm] [,IOSTAT=ios] 
[RECL=rcl] 
[ACTION=act] [,PASSWORD=pwd] 
[POSITION=pos] 
[PAD=pad] 
[DELIM=dlm] 
[KEYS=(start : end [,start : end]...))]
```

Connects an existing file to a unit, creates a file that is preconnected, creates a file and connects it to a unit, or changes certain specifiers of a connection between a file and a unit.

### PARAMETER Statement

**PARAMETER** *(name1=constant1 [,name2=constant2]...)*

Assigns a name to a constant.

### PAUSE Statement

**PAUSE** 

```
PAUSE [n] 
PAUSE ['message']
```

Temporarily halts the processing of the program and displays a message.

### POINTER Statement

**POINTER[*(len)[ptr1[*(len)],ptee1[dim]]][,ptr2[*(len)],ptee2[dim]]...]**

Specifies a pointer variable and associates it with a target variable, called a pointee variable. The value of the pointer variable is the storage address of the pointee variable.
Statement Syntax

PRINT Statement—Formatted with Sequential Access
PRINT fmt [, list]
Transfers data from internal storage to an external device.

PRINT Statement—List-Directed to External Devices
PRINT * [, list]
Transfers data from internal storage to an external device.

PRINT Statement—NAMELIST with External Devices
PRINT name
Transfers data from internal storage to an external device.

PROGRAM Statement
PROGRAM name
Assigns a name to a main program.

READ Statement—Asynchronous
READ
   ( [UNIT=] un,
     ID=id)
   [list]
Transmits unformatted data from a direct-access or tape device using sequential access.

READ Statement—Formatted with Direct Access
READ
   ([UNIT=] un, [FMT=] fmt,
    REC=rec [,ERR=stl]
    [,IOSTAT=ios]) [list]
Transfers data from an external direct-access device into internal storage.
Statement Syntax

**READ Statement—Formatted with Keyed Access (Direct Retrieval)**

```plaintext
READ
    ([UNIT=] un, [FMT=] fmt [,ERR=stl]
    [,IOSTAT=ios] [,KEYID=kid] [,NOTFOUND=stl]
    [,KEY=key | ,KEYGE=kgel | ,KEYGT=kgt]) [list]
```

Transfers data from an external direct-access device into internal storage.

**READ Statement—Formatted with Keyed Access (Sequential Retrieval)**

```plaintext
READ
    ([UNIT=] un, [FMT=] fmt [,ERR=stl]
    [,IOSTAT=ios] [,NOTFOUND=stl | ,END=stl])
    [list]
```

Transfers data from an external direct-access device into internal storage.

**READ Statement—Formatted with Sequential Access**

```plaintext
READ fmt [,list]
```

```plaintext
READ
    ([UNIT=] un, [FMT=] fmt
    [,ERR=stl] [,END=stl]
    [,IOSTAT=ios]) [list]
```

Transfers data from an external I/O device to storage.

**READ Statement—Formatted with Sequential Access to Internal Files**

```plaintext
READ
    ([UNIT=] un, [FMT=] fmt
    [,ERR=stl] [,END=stl]
    [,IOSTAT=ios]) [list]
```

Transfers data from one area of internal storage into another area of internal storage.
Statement Syntax

READ Statement—List-Directed from External Devices

READ * [,list]

READ
    ([UNIT=] un, [FMT=] *)
    [,ERR=stl] [,END=stl]
    [,IOSTAT=ios)] [list]

Transfers data from an external device into internal storage.

READ Statement—List-Directed with Internal Files

READ
    ([UNIT=] un, [FMT=] *)
    [,ERR=stl] [,END=stl]
    [,IOSTAT=ios)] [list]

Transfers data from one area of internal storage to one or more other areas of internal storage.

READ Statement—NAMELIST with External Devices

READ name

READ
    ([UNIT=] un
     ([,FMT=]name | [,NML=]name)
     [,ERR=stl] [,END=stl]
     [,IOSTAT=ios])

Transfers data from an external I/O device into storage.

READ Statement—NAMELIST with Internal Files

READ
    ([UNIT=] un
     ([,FMT=]name | [,NML=]name)
     [,ERR=stl] [,END=stl]
     [,IOSTAT=ios])

Transfers data from one area of internal storage to one or more other areas of internal storage.
Statement Syntax

READ Statement—Unformatted with Direct Access

READ

\([\text{UNIT=} un, \text{REC=} rec\]
\[\text{ERR=} stl\] [\text{IOSTAT=} ios]
\[\text{NUM=} n]\) \[\text{list}\]

Transfers data without conversion from an external direct-access device into internal storage.

READ Statement—Unformatted with Keyed Access (Direct Retrieval)

READ

\([\text{UNIT=} un\[,\text{ERR=} stl\] [,\text{IOSTAT=} ios\] [,\text{KEYID=} kid\]
\[\text{KEY=} key\ | \text{KEYGE=} kge \ | \text{KEYGT=} kgt\]
\[\text{NOTFOUND=} stl\] [,\text{NUM=} n]\) \[\text{list}\]

Transfers data without conversion from an external direct-access I/O device into internal storage.

READ Statement—Unformatted with Keyed Access (Sequential Retrieval)

READ

\([\text{UNIT=} un\[,\text{ERR=} stl\] [,\text{IOSTAT=} ios\]
\[\text{NOTFOUND=} stl\] [,\text{END=} stl\]
\[\text{NUM=} n]\) \[\text{list}\]

Transfers data without conversion from an external direct-access I/O device into internal storage.

READ Statement—Unformatted with Sequential Access

READ

\([\text{UNIT=} un\[,\text{ERR=} stl\]
\[\text{END=} stl\] [,\text{IOSTAT=} ios\]
\[\text{NUM=} n]\) \[\text{list}\]

Transfers data without conversion from an external I/O device into internal storage.
Statement Syntax

RETURN Statement in Function Subprogram

RETURN

Returns control to the calling program.

RETURN Statement in Subroutine Subprogram

RETURN \[m\]

Returns control to the calling program.

REWIND Statement

REWIND \[un\]

REWIND

\([\text{UNIT=un}, \text{ERR=err}]\)

\([\text{IOSTAT=ios}]\)

Repositions a sequentially-accessed file at the beginning of the first record of the file.

REWRITE Statement—Formatted with Keyed Access

REWRITE

\((\text{[UNIT=un}, \text{FMT=fmt}]\)

\([\text{ERR=stl}], \text{[IOSTAT=ios}]\)

\([\text{DUPKEY=stl}]) \text{ list}\)

Replaces a record in a keyed file.

REWRITE Statement—Unformatted with Keyed Access

REWRITE

\((\text{[UNIT=un}, \text{ERR=stl}]\)

\([\text{IOSTAT=ios}, \text{DUPKEY=stl}]\)

\([\text{NUM=num}]\) \text{ list}\)

Replaces a record in a keyed file.
Statement Syntax

SAVE Statement

SAVE [name1 [,name2]...]

Retains the definition status of the name of a named common block, variable
or array after the processing of a RETURN or END statement in a subpro-
gram.

Statement Function Statement

name ([arg1 [,arg2]...]) = m

Specifies operations to be performed whenever that statement function name
appears as a function reference in another statement in the same program.

STATIC Statement

STATIC name1 [/clist1] [, name2[/clist2] ... ]

Identifies the variables and arrays to be assigned the static storage class.

STOP Statement

STOP [n]
STOP ['message']

Ends the processing of the object program and displays a message.

SUBROUTINE Statement

SUBROUTINE name ([([arg1 [,arg2]...]])

Identifies a subroutine subprogram.

TRACE OFF Statement

TRACE OFF

 Stops the display of program flow by statement label.
Statement Syntax

TRACE ON Statement

TRACE ON

Initiates the display of program flow by statement label.

WAIT Statement

WAIT

\[(UNIT=)un, ID=id\]
\[.COND=i1 [.NUM=i2])\]
\[list\]

Synchronizes the completion of the data transmission begun by the corresponding asynchronous READ or WRITE statement.

WRITE Statement—Asynchronous

WRITE

\[(UNIT=] un, \[ID=id\]
\[list\]

Transmits data from an array in main storage to an external file.

WRITE Statement—Formatted with Direct Access

WRITE

\[(UNIT=] un, \[FMT=] fmt, \[REC=)rec [.ERR=)stl\]
\[.JOSTAT=)ios]) [list]\]

Transfers data from internal storage onto an external device.

WRITE Statement—Formatted with Keyed Access

WRITE

\[(UNIT=] un, \[FMT=] fmt\]
\[.ERR=)stl [.JOSTAT=)ios\]
\[.DUPKEY=)stl) list\]

Transfers data from internal storage onto an external device.
Statement Syntax

WRITE Statement—Formatted with Sequential Access

WRITE
  ([UNIT=] un, [FMT=] fmt
   [,ERR=stl] [,IOSTAT=ios])
 [list]

Transfers data from internal storage to a file.

WRITE Statement—Formatted with Sequential Access to Internal Files

WRITE
  ([UNIT=] un, [FMT=] fmt
   [,ERR=stl] [,IOSTAT=ios])
 [list]

Transfers data from one or more areas of internal storage to another area in internal storage.

WRITE Statement—List-Directed to External Devices

WRITE
  ([UNIT=] un, [FMT=] *
   [,ERR=stl] [,IOSTAT=ios])
 [list]

Transfers data from internal storage to a file.

WRITE Statement—List-Directed with Internal Files

WRITE
  ([UNIT=] un, [FMT=] *
   [,ERR=stl] [,IOSTAT=ios])
 [list]

Transfers data from one or more areas of internal storage to another area of internal storage.
Statement Syntax

WRITE Statement—NAMELIST with External Devices

WRITE
  ([UNIT=] un,
   [,FMT=]name \ [,NML=]name
   [,ERR=]str
   [,IOSTAT=]ios)

Transfers data from internal storage to a file.

WRITE Statement—NAMELIST with Internal Files

WRITE
  ([UNIT=] un,
   [,FMT=]name \ [,NML=]name
   [,ERR=]str
   [,IOSTAT=]ios)

Transfers data from one or more areas of internal storage to another area of internal storage.

WRITE Statement—Unformatted with Direct Access

WRITE
  ([UNIT=] un, REC=rec
   [,ERR=]str \ [,IOSTAT=]ios
   [,NUM=n]) \ list

Transfers data without conversion from internal storage to a file.

WRITE Statement—Unformatted with Keyed Access

WRITE
  ([UNIT=] un [,ERR=]str
   [,IOSTAT=]ios \ [,NUM=n]
   [,DUPKEY=]str) \ list

Transfers data without conversion from internal storage to a file.
WRITE Statement—Unformatted with Sequential Access

WRITE
  ([UNIT=] un [,ERR=stl]
  [,IOSTAT=ios] [,NUM=n])
  [list]

Transfers data without conversion from internal storage to a file.
Parallel Statement Categories

In the following statements, extensions to the Fortran standard language are printed in color.

Parallel Task Management Statements
- ORIGINATE
- SCHEDULE
- TERMINATE
- WAIT FOR
  - ALL TASKS
  - ANY TASK
  - TASK

Parallel Loop Statements
- EXIT
- LOCAL
- PARALLEL DO
- DOAFTER
- DOBEFORE
- DOEVERY

Parallel Sections Statements
- END SECTIONS
- LOCAL
- PARALLEL SECTIONS
- SECTION

Parallel Call Statements
- PARALLEL CALL
- WAIT FOR ALL CALLS
Parallel Statement Syntax

In the following statements, extensions to the Fortran standard language are printed in color.

Parallel Task Management Statements

ORIGINATE Statement

ORIGINATE TASK \texttt{ptaskid} \| ORIGINATE ANY TASK \texttt{rtaskid}

Creates a new parallel task.

SCHEDULE Statement

\{(SCHEDULE TASK \texttt{ptaskid} \| SCHEDULE ANY TASK \texttt{rtaskid})
\[\[,\text{SHARING (shrcm [,shrcm]...)}\]
\[\[,\text{COPYING (cpcom [,cpcom]...)}\]
\[\[,\text{COPYINGI (cpcom [,cpcom]...)}\]
\[\[,\text{COPYINGO (cpcom [,cpcom]...)}\]
\[,\text{CALLING subx ([([arg],[arg]...])]}\]

Assigns a subroutine to an originated task for parallel processing. Must have a matching WAIT FOR ALL TASKS, WAIT FOR ANY TASK, or WAIT FOR TASK statement.

TERMINATE Statement

TERMINATE TASK \texttt{taskid}

Deletes a parallel task created by ORIGINATE.

WAIT FOR Statements

One of the following WAIT FOR statements is required for each SCHEDULE statement in a parallel task.

WAIT FOR ALL TASKS Statement

WAIT FOR ALL TASKS

Causes the scheduling routine to wait for all originated tasks, owned by the scheduling routine, to finish processing.
Parallel Statement Syntax

WAIT FOR ANY TASK Statement
WAIT FOR ANY TASK rtaskid
Causes the scheduling routine to wait for any originated task, owned by the scheduling routine, to finish processing.

WAIT FOR TASK Statement
WAIT FOR TASK ptaskid
Causes the scheduling routine to wait for a specified originated task, owned by the scheduling routine, to finish processing.

Parallel Loop Statements

EXIT Statement
EXIT stl
Stops processing of the DOEVERY block of a parallel loop, whether or not all the iterations have finished running.

LOCAL Statement (for parallel loops)
LOCAL var [,var]...
Specifies that an instance of each variable and array listed is provided to each virtual processor participating in execution of the parallel loop.

PARALLEL DO Statement
PARALLEL DO [stl [,]] i = e1, e2 [,e3]
Similar to the DO statement except each iteration of the loop can be processed concurrently; permits parallelism of the loops to be explicitly stated.

DOAFTER Statement
DOAFTER [LOCK]
Indicates the beginning of a block of statements that each virtual processor participating in the execution of the PARALLEL DO processes after the loop is run.
Parallel Statement Syntax

**DOBEFORE Statement**

DOBEFORE [LOCK]

Indicates the beginning of a block of statements that each virtual processor participating in the execution of the PARALLEL DO processes before the loop is run.

**DOEVERY Statement**

DOEVERY

Indicates the beginning of a block of statements with processing shared by the virtual processors assigned to the loop.

**Parallel Sections Statements**

**END SECTIONS Statement**

END SECTIONS

Terminates a group of parallel sections.

**LOCAL Statement (for parallel sections)**

LOCAL var [,var]...

Specifies that an instance of each variable and array listed is provided to each virtual processor participating in execution of the parallel loop.

**PARALLEL SECTIONS Statement**

PARALLEL SECTIONS

Indicates the beginning of a group of sections that can be run in parallel with other sections in the group.

**SECTION Statement**

SECTION [m][.WAITING (n1 [,n2]...)]

Indicates the beginning of a block of statements to be processed as a parallel thread.
Parallel Statement Syntax

Parallel Call Statements

PARALLEL CALL Statement

PARALLEL CALL name [[arg1[,arg2]...]]

Assigns a subroutine to run as a parallel thread.

WAIT FOR ALL CALLS Statement

WAIT FOR ALL CALLS

Causes the calling routine to wait until all subroutines invoked with PARALLEL CALL within the same parallel thread have completed.
Compile-Time Options

**Note:** To specify compile-time options on a program-by-program basis, use the `@PROCESS` compiler directive; for example: `@PROCESS LIST TEST`.

**AUTODBL**  
(None | `DBL` | `DBL4` | `DBL8` | `DBLPAD` | `DBLPAD4` | `DBLPAD8` | value)  
Provides an automatic means of converting single-precision floating-point calculations to double-precision, and double-precision calculations to extended-precision.

**CHARLEN**  
(number | 500)  
Specifies the maximum length permitted for any character variable, character array element, or character function.

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

**DBCS | NODBCS**  
Specifies whether the source file may contain double-byte characters.

**DC**  
(* | name1, name2,...)  
Defines the names of common blocks to be allocated at run time.

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

**DECK | NODECK**  
Specifies whether the compiler is to write the object module to the data set defined by the ddname SYSPUNCH.

**DIRECTIVE**  
(trigger-constant | NODIRECTIVE [(trigger-constant ) ] )  
Specifies whether selected comments containing compiler directive statements are to be processed.

**DYNAMIC**  
(name1,name2...)  
Provides dynamic loading of user subroutines or functions during program execution.

**EC**  
(* | name1, name2,...)  
Defines the names of common blocks to be dynamically allocated as extended common blocks.

**EMODE | NOEMODE**  
Specifies that the code compiled for a subroutine or function can receive parameters that reside in an extended common block.
Compile-Time Options

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

Items not defined in the current American National Standard are flagged.

FLAG (I | W | E | S)
Specifies the level of diagnostic messages to be written: I (information) or higher, W (warning) or higher, E (error) or higher, or S (severe) or higher.

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

GOSTMT | NOGOSTMT
Specifies whether internal statement numbers (for run-time error debugging information) are to be generated for a calling sequence to a subprogram or to the run-time library from the compiler-generated code.

HALT (I | W | E | S)
Causes termination of the compile after any phase if the compiler return code is at or above the specified level.

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

NOICA
Specifies whether intercompilation analysis is to be performed, specifies the files containing intercompilation analysis information to be used or updated, and controls output from intercompilation analysis.

IL (DIM | NODIM)
Specifies whether the code for adjustably-dimensioned arrays is to be placed inline, IL(DIM) or called from the library, IL(NODIM).

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

LINECOUNT (number | 60)
Specifies the maximum number of lines on each page of the printed source listing.
Compile-Time Options

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

MAP | NOMAP
Specifies whether a table of source program variable names, named constants, and statement labels and their displacements is to be produced.

NAME (name | MAIN#)
Specifies the name of the control section (CSECT) generated in the object module of the main program (valid only when LNGLVL(66) is specified).

OBJECT | NOOBJECT
Under CMS, specifies whether the compiler is to write the object module to the file associated with the ddname TEXT.

Under MVS, specifies whether the compiler is to write the object module to the data set associated with the ddname SYSLIN.

OPTIMIZE (0 | 1 | 2 | 3) | NOOPTIMIZE
Specifies the optimizing level to be used during compilation:
  - OPTIMIZE (0) or NOOPTIMIZE specifies no optimization.
  - OPTIMIZE (1) specifies partial optimization.
  - OPTIMIZE (2) specifies full optimization with interruption localizing.
  - OPTIMIZE (3) specifies full optimization without interruption localizing.

PARALLEL [ (]
[ REPORT [ (optionlist) ] | NOREPORT ]
[ LANGUAGE | NOLANGUAGE ]
[ AUTOMATIC | NOAUTOMATIC ]
[ REDUCTION | NOREDUCTION ]
[ TRACE | NOTRACE ]
[ ANZCALL | NOANZCALL ]
[ ) ]

NOPARALLEL
Specifies suboptions to the compiler for generating code for DO loops, the PARALLEL DO, PARALLEL SECTIONS, and PARALLEL CALL constructs, and the task management statements.

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

RENT | NORENT
Specifies whether the object module generated is suitable for use in a shareable area.

Compiler and Library Information  33
Compile-Time Options

**SAA | NOSAA**
Specifies whether flagging of language elements that are not part of the Systems Application Architecture (SAA) is to be performed.

**SC (** 1 name1, name2,...**)**
Defines the names of common blocks to be compiled as static common blocks.

**SDUMP [(ISN | SEQ )] | NOSDUMP**
Specifies whether symbolic dump information is to be generated, and if so, whether internal statement numbers or sequence numbers will be used. The -g flag on the fvs command is equivalent to this option.

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

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

**SXM | NOSXM**
Formats XREF or MAP listing output to a 72-character width.

**SYM | NOSYM**
Invokes the production of SYM cards in the object text file. The SYM 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 SYSTERM data set and whether a summary of messages for all compilations is to be written at the end of the listing.

**TEST | NOTEST**
TEST overrides any optimization level about OPTIMIZE(0).

**TRMFLG | NOTRMFLG**
Controls the display of error messages on the terminal.

**VECTOR [ (**

- **REPORT [ (optionlist) ] | NOREPORT ]**
- **INTRINSIC | NOINTRINSIC ]**
- **IVA | NOIVA ]**
- **REDUCTION | NOREDUCTION ]**
- **SIZE ((ANY | LOCAL | n ) ) ]**
- **MODEL (ANY/VF2/LOCAL) ]**
- **SPRECOPT | NOSPRECOPT ]**
- **ANZCALL | NOANZCALL ]**

---

* SAA and Systems Application Architecture are trademarks of the International Business Machines Corporation.
Compile-Time Options

[ CMPLXOPT | NOCMPLXOPT ]

NOVECTOR
Specifies whether to invoke the vectorization process, which produces programs that can utilize the speed of the IBM' 3090' vector facility.

XREF | NOXREF
Specifies whether a cross-reference listing is to be produced.

* IBM and 3090 are trademarks of the International Business Machines Corporation.
### Conflicting Compile-Time Options

<table>
<thead>
<tr>
<th>Conflicting Compile-Time Options</th>
<th>Options Assumed</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCS FIPS</td>
<td>DBCS NOFIPS</td>
</tr>
<tr>
<td>DBCS SAA</td>
<td>DBCS NOSAA</td>
</tr>
<tr>
<td>DC EC</td>
<td>Note 1 Note 1</td>
</tr>
<tr>
<td>DC SC</td>
<td>Note 1 Note 1</td>
</tr>
<tr>
<td>EC SC</td>
<td>Note 1 Note 1</td>
</tr>
<tr>
<td>EC NOEMODE</td>
<td>EC EMODE</td>
</tr>
<tr>
<td>FIPS FLAG=¬I</td>
<td>FIPS FLAG=I</td>
</tr>
<tr>
<td>FIPS SAA</td>
<td>Installation default</td>
</tr>
<tr>
<td>FREE FREE</td>
<td>NOFIPS</td>
</tr>
<tr>
<td>FREE SAA</td>
<td>NOSAA</td>
</tr>
<tr>
<td>FREE SDUMP(SEQ)</td>
<td>FREE SDUMP(ISN)</td>
</tr>
<tr>
<td>LANGlvl(66) DBCS</td>
<td>NODBCS</td>
</tr>
<tr>
<td>LANGlvl(66) FIPS</td>
<td>NOFIPS</td>
</tr>
<tr>
<td>LANGlvl(66) SAA</td>
<td>NOSAA</td>
</tr>
<tr>
<td>LANGlvl(77) NAME</td>
<td>Ignore NAME</td>
</tr>
<tr>
<td>NOOECK SYM</td>
<td>NOSYM</td>
</tr>
<tr>
<td>NOTRMFLG VEC(REP(TERM...)</td>
<td>VEC(REP(NOTERM...))</td>
</tr>
<tr>
<td>NOTRMFLG PAR(REP(TERM...)</td>
<td>PAR(REP(NOTERM...))</td>
</tr>
<tr>
<td>PAR OPT(0) or OPT(1)</td>
<td>PAR OPT(3)</td>
</tr>
<tr>
<td>PAR TEST</td>
<td>NOPAR TEST</td>
</tr>
<tr>
<td>PAR(NOREP) VEC(REP(LIST))</td>
<td>VEC(REP(LIST))</td>
</tr>
<tr>
<td>PAR(ANZCALL) VEC(NOANZCALL)</td>
<td>VEC(NOANZCALL)</td>
</tr>
<tr>
<td>PAR(NOANZCALL) VEC(ANZCALL)</td>
<td>VEC(ANZCALL)</td>
</tr>
<tr>
<td>PAR(NOREP) VEC(REP(XLIST))</td>
<td>VEC(REP(XLIST))</td>
</tr>
<tr>
<td>PAR(REP(LIST)) PAR(REP(STAT))</td>
<td>VEC(REP(STAT))</td>
</tr>
<tr>
<td>PAR(REP(XLIST)) VEC(REP(XLIST))</td>
<td>VEC(REP(XLIST))</td>
</tr>
<tr>
<td>PAR(REP(STAT)) VEC(REP(STAT))</td>
<td>VEC(REP(STAT))</td>
</tr>
<tr>
<td>PAR(IV) VEC(REP(SLIST XLIST))</td>
<td>VEC(REP(SLIST XLIST))</td>
</tr>
<tr>
<td>TEST OPT(1), OPT(2), OPT(3)</td>
<td>TEST OPT(0)</td>
</tr>
<tr>
<td>TEST NOSDUMP TEST</td>
<td>SDUMP(ISN)</td>
</tr>
<tr>
<td>VEC OPT(0) or OPT(1)</td>
<td>VEC OPT(3)</td>
</tr>
<tr>
<td>VEC(IV) NOSDUMP VEC(IV)</td>
<td>SDUMP(ISN)</td>
</tr>
<tr>
<td>VEC(IV) PAR VEC(NOIVA) PAR</td>
<td></td>
</tr>
</tbody>
</table>

**Note:**

1. The SC, DC, and EC compile-time options use the last option indicated to resolve conflicts between them.
Compiler Directives

@PROCESS

Provides compile-time options that override the corresponding default options or those specified at compiler invocation.

EJECT

Starts a new full page of the source listing.

INCLUDE

INCLUDE (member) [n]

INCLUDE 'filename [filetype [filemode]] [(member)]'

INCLUDE 'dsn [(member)]'

INCLUDE 'filename'

Inserts a specified statement or group of statements into a program unit.
Parallel and Vector Directives

ASSUME COUNT

Local Directive:
ASSUME COUNT (val)

Global Directive:
ASSUME COUNT ((val=var=val[,var=val]...)) ON
ASSUME COUNT OFF

Specifies the value to be used for vector or parallel cost analysis when a loop iteration count cannot be determined at compile time.

IGNORE

IGNORE [RECRDEPS [(array-list)]] [CALLDEPS [(name,name...)]]

Instructs the compiler to ignore specified dependences in a loop.

Warning: Use with extra caution. Incorrectly specifying IGNORE can produce erroneous program results.

PREFER

Local PREFER:
PREFER [SCALAR | VECTOR]
    [SERIAL | PARALLEL]
    [CHUNK([(n:n::m:n::m)])]

Global PREFER:
PREFER [SCALAR | SERIAL] ON | OFF

Requests that particular loops be run in vector or scalar and/or parallel or serial modes.
Run-Time Options

**ABSDUMP | NOABSDUMP**
Specifies whether the post-ABEND symbolic dump information is to be printed in the event of an abnormal termination.

**AUTOTASK (loadmod,ntasks) | NOAUTOTASK**
Specifies whether the multitasking facility (MTF) is enabled for your program. This option available on MVS only.

**CNVIOERR | NOCNVIOERR**
Specifies whether to treat input conversion errors as I/O errors.

**DEBUG | NODEBUG**
Specifies whether to call VS FORTRAN Version 2 interactive debug.

**DEBUNIT(s1 [, s2...]) | NODEBUNIT (MVS format)**
Identifies Fortran units considered to be connected to a terminal, so that interactive debug can handle their I/O in batch mode.

**DEBUNIT(s1 [ s2 s3]) | NODEBUNIT (CMS format)**
Identifies Fortran units considered to be connected to a terminal, so that interactive debug can handle their I/O in batch mode.

**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.

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

**FAIL (ABEND | RC | ABENDRC)**
Indicates how to terminate unsuccessful programs. This option is not supported for parallel processing.

**FILEHIST | NOFILEHIST**
Specifies whether to allow the file definition referred to by a ddname to be changed at run time.

**INQPCOPN | NOINQPCOPN**
Controls using the OPENED specifier on an INQUIRE by unit to determine whether a preconnected unit had any I/O statements directed to it.

**IOINIT | NOIOINIT**
Specifies whether the normal initialization for I/O processing occurs during initialization of the run-time environment.

**OCSTATUS | NOOCSTATUS**
Specifies whether to verify the OPEN and CLOSE status specifiers.

**PARALLEL [(numprocs)] | NOPARALLEL**
Specifies whether the program runs in the parallel processing environment.
Run-Time Options

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

- **PTRACE[(options)]**
  - Enables the Parallel Trace Facility and causes the Trace 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.

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

- **RECPAD | 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.

- **SPIE | NOSPIE**
  - Specifies whether the run-time environment takes control when a program interrupt occurs.

- **STAE | NOSTAE**
  - Specifies whether the run-time environment takes control in the event of an abnormal termination.

- **XUFLOW | NOXUFLOW**
  - Specifies whether an exponent underflow causes a program interrupt.
Service Subroutines

ARGSTR Subroutine
CALL ARGSTR(string, rc)
Retrieves the user-supplied parameters from the command line.

ASSIGNM Subroutine
CALL ASSIGNM(input, output, rcode, rsncode)
Moves a character string containing double-byte data to a character variable, substring, or array element, preserving balanced shift codes.

CLOCK Subroutine
CALL CLOCK(cpuclk, [count [,max]])
Returns the value of the processor clock as a positive integer.

CLOCKX Subroutine
CALL CLOCKX(cpuclk [,xcount [,xmax]])
Returns an abbreviated version of the processor clock in a REAL*8 variable.

CDUMP/CPDUMP Subroutines
CALL (CDUMP | CPDUMP) (a1,b1,a2,b2 ...)
Provides a symbolic dump of a specified area of storage containing character data.

CPUTIME Subroutine
CALL CPUTIME(accumcpu, rcode)
Lets you determine the amount of processor time used by a program or portion of a program. This subroutine is not allowed in a parallel program.

DATIM Subroutine
CALL DATIM(now)
Provides information about the date, time of day, and processor clock.
Service Subroutines

DATIMX Subroutine
CALL DATIMX (now)
Provides the date and time in a form that can be used to produce printable or formatted data.

DUMP/PDUMP Subroutines
CALL (DUMP | PDUMP) (a1,b1,k1,a2,b2,k2 ...)
Provides a symbolic dump of a specified area of storage.

DVCHK Subroutine
CALL DVCHK (k)
Tests for divide-check exception.

EXIT Subroutine
CALL EXIT
Ends processing of the program. This subroutine can be called only in serial parts of the root task.

FILEINF Subroutine
CALL FILEINF [{rcode [,param1, value1, param2, value2 ...]}]
Sets up file characteristics to be used by an OPEN or an INQUIRE statement.

MVBITS Subroutine
CALL MVBITS(arg1,arg2,arg3,arg4,arg5)
Allows a bit subfield of one integer value to be assigned to a bit subfield of another integer value.
Service Subroutines

**OVERFL Subroutine**

CALL OVERFL (k)

Tests for exponent overflow or underflow.

---

**PFAFFS Routine**

CALL PFAFFS

Switches the virtual processor the parallel thread is running on to the same one that the parallel program first started running on; allows use of system services.

---

**PFAFFC Routine**

CALL PFAFFC

Releases any processor affinity previously set by calling PFAFFS.

---

**PYIELD Routine**

CALL PYIELD

Causes the VS FORTRAN Version 2 library to interrupt the execution of the current thread and to attempt to execute any threads now waiting to execute.

---

**PTPARM Routine**

CALL PTPARM (argstring)

Provides a means for your program to dynamically control the tracing activity of the Parallel Trace Facility during program execution.

---

**PTWRIT Routine**

CALL PTWRIT(category,type,user_data [,user_data_len])

Allows you to generate your own trace records for events you determine are of significance in the execution of your program.
Service Subroutines

SDUMP Subroutine
CALL SDUMP [(rtn1 [,rtn2]...)]
Provides a symbolic dump of all variables in a program unit.

SYSABD Subroutine
CALL SYSABD (compl-code)
Causes abnormal termination of your job with a dump.

SYSABN Subroutine
CALL SYSABN (compl-code)
Causes abnormal termination of your job without a dump.

SYSRCS Subroutine
CALL SYSRCS (n)
Saves a return code value for future termination.

SYSRCT
CALL SYSRCT (m)
Obtains the value of the currently saved return code.

SYSRCX Subroutine
CALL SYSRCX [(k)]
Ends program processing using either the saved return code or a supplied return code.

UNTANY Subroutine
CALL UNTANY (rcode, startnum, endnum, unitnum)
Identifies the lowest Fortran unit number that is available, within a range of unit numbers, regardless of the file definitions in effect.
Parallel Service Subroutines

**UNTNOFD Subroutine**

CALL UNTNOFD (rcode, startnum, endnum, unitnum)

Identifies the lowest Fortran unit number that is available, within a range of unit numbers, that does not have a user-specified file definition associated with it.

---

**XUFLOW Subroutine**

CALL XUFLOW (k)

Allows or suppresses a program interrupt caused by exponent underflow.

---

**Parallel Library Event Service Subroutines**

**PEORIG Subroutine**

CALL PEORIG (eventid [,postcount [,waitcount [,unique ] ] ] )

Creates and initializes an event and returns an identifier for the event.

---

**PEPOST Subroutine**

CALL PEPOST (eventid)

Posts the specified event.

---

**PETERM Subroutine**

CALL PETERM (eventid)

Deletes the specified event.

---

**PEWAIT Subroutine**

CALL PEWAIT (eventid)

Causes the calling parallel thread to wait until the event’s post-count (or wait-count if the current post-count equal to the initial value) is reached.
Parallel Service Subroutines

Parallel Library Lock Service Subroutines and Function

PLCOND Function

PLCOND (lockid [,mode [,var [,var]... ] ] )
Conditionally obtains the specified lock.

PLFREE Subroutine

CALL PLFREE (lockid [,var [,var]... ] )
Releases the specified lock.

PLLOCK Subroutine

CALL PLLOCK (lockid [,mode [,var [,var]... ] ] )
Obtains the specified lock. If the lock is currently owned by another parallel thread, waits until lock is available.

PLORIG Subroutine

CALL PLORIG (lockid)
Creates and initializes a lock and returns an identifier for the lock.

PLTERM Subroutine

CALL PLTERM (lockid)
Deletes the specified lock.
Parallel Service Subroutines

Parallel Function

NPROCS Function

NPROCS ([n])

Allows the program to determine the number of virtual processors specified at run time.
Data-in-Virtual Subroutines

The data-in-virtual subroutines are not available for parallel processing.

DIVCML Subroutine
CALL DIVCML (rcode, dyncom, length)

Obtains the length of a dynamic or extended common.

DIVINF Subroutine
CALL DIVINF (rcode, dyncom, objsize_commons, divobj, type, access)

Allows you to associate a data object with a dynamic or extended common for reading or for reading and writing (fixed-view).

DIVINV Subroutine
CALL DIVINV (rcode, obj-id, objsize_pages, divobj, type, access)

Allows you to associate a data object with a data object ID for reading or for reading and writing (varying-view).

DIVRES Subroutine
CALL DIVRES (rcode, dyncom)

Resets the data in the dynamic or extended common to the values in the mapped part of the data object, eliminating any changes that have been made in the dynamic or extended common, either initially or since the last DIVSAV.

DIVSAV Subroutine
CALL DIVSAV (rcode, dyncom)

Saves changes made in the dynamic or extended common to the data object that has been accessed for READWRITE.
Data-in-Virtual Subroutines

**DIVTRF Subroutine**

CALL DIVTRF (rcode, dyncom)

Terminates the association of the data object to the dynamic or extended common (fixed-view).

**DIVTRV Subroutine**

CALL DIVTRV (rcode, obj-id)

Terminates the association between the data object ID and the data object (varying-view).

**DIVVWF Subroutine**

CALL DIVVWF (rcode, dyncom, mapnum)

Establishes the part of the data object the dynamic or extended common maps (fixed-view).

**DIVVWV Subroutine**

CALL DIVVWV (rcode, dyncom, offset, obj-id)

Establishes the part of the data object the dynamic or extended common maps (varying-view).
Multitasking Facility (MTF) Subroutines

**DSPTCH Subroutine**

CALL DSPTCH (subrname [, (arg1) [, (arg2) ]...])

Schedules a parallel subroutine for processing in a subtask.

**NTASKS Subroutine**

CALL NTASKS (n)

Returns the number of subtasks specified with the AUTOTASK keyword in the PARM parameter of the EXEC statement for the job step. Returns a value of zero when the AUTOTASK keyword is not in effect.

**SHRCOM Subroutine**

CALL SHRCOM (dyncom)

Designates a dynamic common as shareable among the main task program and the parallel subroutines.

**SYNCRO Subroutine**

CALL SYNCRO

Causes the main task program to wait until all scheduled parallel subroutines finish processing.
Error-Handling Subroutines

Error-Handling Subroutines

Note that each parallel task has its own error option table. A parallel thread uses the error option table associated with the parallel task in which it runs.

**ERRMON Subroutine**

CALL ERRMON (imes, iretdc, ierno [, data1] [, data2, ... ])

Calls the error monitor routine.

**ERRSAV Subroutine**

CALL ERRSAV (ierno, tabent)

Copies an option table entry into an 8-byte storage area accessible to the Fortran programmer.

**ERRSET Subroutine**

CALL ERRSET (ierno, inoal [, inomes] [, itrace]

[, iusadr] [, irangel])

Permits the user to control processing when error conditions occur.

**ERRSTR Subroutine**

CALL ERRSTR (ierno, tabent)

Stores an entry in the option table.

**ERRTRA Subroutine**

CALL ERRTRA

Dynamically requests a traceback and continued processing.
Intrinsic Functions

Intrinsic Functions

Intrinsic functions are procedures supplied in VS FORTRAN Version 2 for standard mathematical computations, character manipulations, and bit manipulations.

The intrinsic functions provided by VS FORTRAN Version 2 are described in the following figure.

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exponential and Logarithmic Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exponential</td>
<td>EXP</td>
<td>EXP</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td>DEXP</td>
<td>DEXP</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QEXP</td>
<td>QEXP</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CEXP</td>
<td>CEXP</td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDEXP</td>
<td>CDEXP</td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CQEXP</td>
<td>CQEXP</td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
<td></td>
</tr>
<tr>
<td>Natural logarithm</td>
<td>LOG</td>
<td>ALOG</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td>DLOG</td>
<td>DLOG</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QLOG</td>
<td>QLOG</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDLOG</td>
<td>CDLOG</td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CQLOG</td>
<td>CQLOG</td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
<td></td>
</tr>
<tr>
<td>Common logarithm</td>
<td>LOG10</td>
<td>ALOG10</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td>DLOG10</td>
<td>DLOG10</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QLOG10</td>
<td>QLOG10</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td><strong>Trigonometric Functions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sine</td>
<td>SIN</td>
<td>SIN</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td>DSIN</td>
<td>DSIN</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QSIN</td>
<td>QSIN</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CSIN</td>
<td>CSIN</td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDSIN</td>
<td>CDSIN</td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCOSIN</td>
<td>CCOSIN</td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
<td></td>
</tr>
<tr>
<td>Cosine</td>
<td>COS</td>
<td>COS</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td>DCOS</td>
<td>DCOS</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>QCOS</td>
<td>QCOS</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CCOS</td>
<td>CCOS</td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDCOS</td>
<td>CDCOS</td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CQOS</td>
<td>CQOS</td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
<td></td>
</tr>
</tbody>
</table>
## Intrinsic Functions

### Intrinsic Functions

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tangent</td>
<td>TAN</td>
<td>TAN(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DTAN(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QTAN</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Cotangent</td>
<td>COTAN</td>
<td>COTAN(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCOTAN(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QCOTAN</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Arcsine</td>
<td>ASIN</td>
<td>ASIN(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DASIN(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QARSIN</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Arccosine</td>
<td>ACOS</td>
<td>ACOS(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DACOS(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QARCOS</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Arctangent</td>
<td>ATAN</td>
<td>ATAN(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QATAN(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td>ATAN2</td>
<td>ATAN2(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATAN2(^2)</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QTAN2</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
</tbody>
</table>

### Hyperbolic Functions

<table>
<thead>
<tr>
<th>Hyperbolic function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperbolic sine</td>
<td>SINH</td>
<td>SINH(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSINH</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QSINH</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Hyperbolic cosine</td>
<td>COSH</td>
<td>COSH(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCOSH</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QCOSH</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
<tr>
<td>Hyperbolic tangent</td>
<td>TANH</td>
<td>TANH(^2)</td>
<td>1</td>
<td>REAL(^*)4</td>
<td>REAL(^*)4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DTANH</td>
<td></td>
<td>REAL(^*)8</td>
<td>REAL(^*)8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QTANH</td>
<td></td>
<td>REAL(^*)16</td>
<td>REAL(^*)16</td>
</tr>
</tbody>
</table>

### Miscellaneous Mathematical Functions

| Truncation          | AINT         | AINT\(^2\)    | 1                | REAL\(^*\)4      | REAL\(^*\)4               |
|                     | DINT         | DINT\(^2\)    |                  | REAL\(^*\)8      | REAL\(^*\)8               |
|                     | QINT         | QINT\(^2\)    |                  | REAL\(^*\)16     | REAL\(^*\)16              |
| Nearest whole number| ANINT        | ANINT\(^2\)   | 1                | REAL\(^*\)4      | REAL\(^*\)4               |
|                     | DNINT        | DNINT\(^2\)   |                  | REAL\(^*\)8      | REAL\(^*\)8               |
| Nearest integer     | NINT         | NINT\(^2\)    | 1                | REAL\(^*\)4      | INTEGER\(^*\)4           |
|                     | IDNINT       | IDNINT\(^2\)  |                  | REAL\(^*\)8      | INTEGER\(^*\)4           |
### Intrinsic Functions

#### Figure 2 (Page 3 of 6). Intrinsic Functions

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute value</td>
<td>ABS</td>
<td>IABS(^a)</td>
<td>1</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABS</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DABS</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QABS</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CABS</td>
<td></td>
<td>COMPLEX*8</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDABS</td>
<td></td>
<td>COMPLEX*16</td>
<td>REAL*8</td>
</tr>
<tr>
<td>Error function</td>
<td>ERF</td>
<td>ERF</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DERF</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QERF</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td></td>
<td>ERFC</td>
<td>ERFC</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DERFC</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QERFC</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td>Gamma and log gamma</td>
<td>GAMMA</td>
<td>GAMMA</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DGAMMA</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALGAMA</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DLGAMA</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td>Remaindering</td>
<td>MOD</td>
<td>MOD(^3)</td>
<td>2</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMOD</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMOD</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMOD</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td>Transfer of sign</td>
<td>SIGN</td>
<td>ISIGN(^5)</td>
<td>2</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SIGN</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSIGN</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QSIGN</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td>Positive difference</td>
<td>DIM</td>
<td>IDIM(^5)</td>
<td>2</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIM</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDIM</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QDIM</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td>Double precision product</td>
<td>DPROD</td>
<td></td>
<td>2</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>Imaginary part of a complex argument</td>
<td>IMAG</td>
<td>AIMAG</td>
<td>1</td>
<td>COMPLEX*8</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIMAG</td>
<td></td>
<td>COMPLEX*16</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QIMAG</td>
<td></td>
<td>COMPLEX*32</td>
<td>REAL*16</td>
</tr>
<tr>
<td>Complex conjugate</td>
<td>CONJG</td>
<td>CONJG</td>
<td>1</td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCNJG</td>
<td></td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QCNJG</td>
<td></td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
</tr>
<tr>
<td>Square root</td>
<td>SQRT</td>
<td>SQR(^7)</td>
<td>1</td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DSGRT(^2)</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QSGRT</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CSGRT</td>
<td></td>
<td>COMPLEX*8</td>
<td>COMPLEX*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CDSGR(^7)</td>
<td></td>
<td>COMPLEX*16</td>
<td>COMPLEX*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CQSGRT</td>
<td></td>
<td>COMPLEX*32</td>
<td>COMPLEX*32</td>
</tr>
</tbody>
</table>
### Intrinsic Functions

#### Conversion and Maximum/Minimum Functions

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion to type integer</td>
<td>INT</td>
<td>—</td>
<td>1</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td>IFIX</td>
<td>REAL*4</td>
<td>1</td>
<td>INTEGER*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDINT</td>
<td>REAL*8</td>
<td>1</td>
<td>INTEGER*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IQINT</td>
<td>REAL*16</td>
<td>1</td>
<td>INTEGER*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HIIFX</td>
<td>COMPLEX*8</td>
<td>1</td>
<td>INTEGER*4</td>
<td></td>
</tr>
<tr>
<td>Conversion to type real</td>
<td>REAL</td>
<td>FLOAT*3</td>
<td>1</td>
<td>any integer</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNGL</td>
<td>REAL*4</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>SNGLQ</td>
<td>REAL*8</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
<td>REAL*16</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DREAL</td>
<td>COMPLEX*8</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QREAL</td>
<td>COMPLEX*16</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td>Conversion to type double precision</td>
<td>DBLE</td>
<td>DFLOAT*3</td>
<td>1</td>
<td>any integer</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBLE</td>
<td>REAL*4</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DBLEQ</td>
<td>REAL*16</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
<td>COMPLEX*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td>Conversion to type extended precision</td>
<td>QEXT</td>
<td>QFLOAT*3</td>
<td>1</td>
<td>any integer</td>
<td>REAL*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QEXT</td>
<td>REAL*4</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QEXTD</td>
<td>REAL*8</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td>Conversion to type complex</td>
<td>CMPLX</td>
<td>—</td>
<td>1 or 2</td>
<td>any integer</td>
<td>COMPLEX*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CMPLX</td>
<td>REAL*4</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
<td>REAL*8</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QCPLX</td>
<td>REAL*16</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>—</td>
<td>COMPLEX*32</td>
<td>COMPLEX*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCPLX*3</td>
<td>REAL*8</td>
<td>COMPLEX*16</td>
<td></td>
</tr>
<tr>
<td>Choosing largest value</td>
<td>MAX</td>
<td>MAXO*</td>
<td>≥ 2</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMAX1</td>
<td>REAL*4</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMAX1</td>
<td>REAL*8</td>
<td>REAL*8</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMAX1</td>
<td>REAL*16</td>
<td>REAL*16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMAXO*3,5</td>
<td>any integer</td>
<td>REAL*4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MAXI*3</td>
<td>REAL*4</td>
<td>INTEGER*4</td>
<td></td>
</tr>
</tbody>
</table>
### Intrinsic Functions

**Figure 2 (Page 5 of 6). Intrinsic Functions**

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choosing smallest value</td>
<td>MIN</td>
<td>MIN0&lt;sup&gt;9&lt;/sup&gt;</td>
<td>2</td>
<td>any integer</td>
<td>same as arg</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMIN1</td>
<td></td>
<td>REAL*4</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DMIN1</td>
<td></td>
<td>REAL*8</td>
<td>REAL*8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QMIN1</td>
<td></td>
<td>REAL*16</td>
<td>REAL*16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>AMIN0&lt;sup&gt;5&lt;/sup&gt;,&lt;sup&gt;5&lt;/sup&gt;</td>
<td></td>
<td>any integer</td>
<td>REAL*4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MIN1&lt;sup&gt;5&lt;/sup&gt;</td>
<td></td>
<td>REAL*4</td>
<td>INTEGER*4</td>
</tr>
</tbody>
</table>

**Storage Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>ALLOCATED</th>
<th>1</th>
<th>any</th>
<th>LOGICAL*4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location of variable</td>
<td>LOC</td>
<td>1</td>
<td>any</td>
<td>POINTER</td>
</tr>
</tbody>
</table>

**Character Manipulation Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>ICHAR</th>
<th>1</th>
<th>CHARACTER*1</th>
<th>INTEGER*4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conversion to type character</td>
<td>CHAR&lt;sup&gt;5&lt;/sup&gt;</td>
<td>1</td>
<td>any integer</td>
<td>CHARACTER*1</td>
</tr>
<tr>
<td>Length</td>
<td>LEN</td>
<td>1</td>
<td>CHARACTER</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>Index of a substring</td>
<td>INDEX</td>
<td>2</td>
<td>CHARACTER</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>Lexically greater than or equal</td>
<td>LGE</td>
<td>2</td>
<td>CHARACTER</td>
<td>LOGICAL*4</td>
</tr>
<tr>
<td>Lexically greater than</td>
<td>LGT</td>
<td>2</td>
<td>CHARACTER</td>
<td>LOGICAL*4</td>
</tr>
<tr>
<td>Lexically less than or equal</td>
<td>LLE</td>
<td>2</td>
<td>CHARACTER</td>
<td>LOGICAL*4</td>
</tr>
<tr>
<td>Lexically less than</td>
<td>LLT</td>
<td>2</td>
<td>CHARACTER</td>
<td>LOGICAL*4</td>
</tr>
</tbody>
</table>

**Bit Manipulation Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>IOR&lt;sup&gt;1&lt;/sup&gt;</th>
<th>2</th>
<th>any integer or unsigned</th>
<th>same as arg</th>
</tr>
</thead>
</table>
### Intrinsic Functions

#### Notes:

1. The generic name must be used for conversion functions when no specific name is supplied.
2. Also available in the alternate mathematical library, which provides alternative functions that provide results compatible with VS FORTRAN Version 1.
3. The specific name must be used to obtain a function value of this type.
4. The bits in bit-manipulation functions are numbered from right to left, beginning at zero.
5. This specific name is also a generic name when used with integer argument(s); it is specific only with INTEGER*4 argument(s). When there is more than one argument, all arguments must agree in length.

#### Figure 2 (Page 6 of 6). Intrinsic Functions

<table>
<thead>
<tr>
<th>Intrinsic Function</th>
<th>Generic Name</th>
<th>Specific Name</th>
<th>No. of Arguments</th>
<th>Type of Argument</th>
<th>Type and Range of Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical and</td>
<td>IAND</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Logical complement</td>
<td>NOT</td>
<td></td>
<td>1</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Exclusive or</td>
<td>IÆOR</td>
<td>Corresponding</td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Shift bits</td>
<td>ISHFT</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Shift left</td>
<td>LSHIFT</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Shift right</td>
<td>RSHIFT</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Shift circularly</td>
<td>ISHFTC</td>
<td></td>
<td>2 or 3</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Bit test</td>
<td>BTTEST</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>LOGICAL*4</td>
</tr>
<tr>
<td>Bit set</td>
<td>IBSET</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Bit clear</td>
<td>IBCLR</td>
<td></td>
<td>2</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
<tr>
<td>Bit extraction</td>
<td>IBITS</td>
<td></td>
<td>3</td>
<td>any integer or unsigned</td>
<td>same as arg</td>
</tr>
</tbody>
</table>
Interactive Debug

Interactive Debug Commands

Note that for parallel programs, you can only use the interactive debug to debug the root task and Fortran code that is not parallel. New types and other features introduced in VS FORTRAN Version 2 Release 6 are not supported by the Interactive Debug. See the summary of changes in the front of the VS FORTRAN Version 2 Language and Library Reference for a list of these features.

When using interactive debug commands:

- A statement number is either an ISN or a sequence number (columns 73-80), depending on how the program was compiled.
- Precede a statement label by a slash (/) when it is used in place of a statement number.
- Separate list items with commas or blanks (except for command lists).
## Interactive Debug Command Categories

### Controlling Program Processing
- AT
- ENDDEBUG
- GO
- HALT
- LISTBRKS
- NEXT
- OFF
- OFFWN
- RESTART
- STEP
- WHEN

### Monitoring and Modifying Variables
- AUTOLIST
- DESCRIBE
- IF
- LIST
- QUALIFY
- SET

### Processing Sequential Files
- BACKSPACE
- CLOSE
- ENDFILE
- RECONNECT
- REWIND

### Controlling Full Screen Display
- COLOR
- DOWN
- LEFT
- LISTINGS
- MOVECURS
- POSITION
- PREVDISP
- PROFILE
- REFRESH

### Controlling Full Screen Display (continued)
- RESTORE
- RETRIEVE
- RIGHT
- SEARCH
- SIZE
- UP
- WINDOW
- ZOOM

### Handling Run Time Library Errors
- ERROR
- FIXUP

### Gathering Vector Tuning Information
- LISTVEC
- VECSTAT

### Tracing and Timing
- ANNOTATE
- LISTFREQ
- LISTSAMP
- LISTSUBS
- LISTTIME
- TIMER
- TRACE
- WHERE

### General Commands
- * or " (Comments)
- DBCS
- HELP
- PURGE
- QUIT
- SYSCMD
- TERMIO
Interactive Debug Command Syntax

* or ' (Comments)

{* | *}

[comment]

Inserts comments in the debugging log.

ANNOTATE Command—Copying Source Listings to a Print File

ANNOTATE

{unit | (unit-list) | *}

[SAMPLING [DIRECT | CALLED | ALL] | FREQUENCY]

Shows sampling or frequency data as a source listing to the AFFPRINT file.

ANNOTATE Command—Providing a Bar Chart in the Source Window

ANNOTATE

[ON | OFF | TOGGLE]

[SAMPLING [DIRECT | CALLED | ALL]

| FREQUENCY | MESSAGE]

Shows sampling or frequency data as a bar chart overlay on the source listing window.

Note: Valid in full screen mode only.

ANNOTATE Command—Querying the Settings

ANNOTATE

Queries the ANNOTATE settings.
Interactive Debug Command Syntax

AT Command

AT
[qual.]
{number[:qual. number] | ENTRY | EXIT}
| (number/ENTRY/EXIT list)
[(command-list)]
[COUNT(n)]
[NOTIFY | NONOTIFY]

Sets breakpoints.

Note: Separate individual commands in the command list with percent signs (%).

AUTOLIST Command

AUTOLIST
{[[qual. ] name :[[qual. ] name]
| * | 'string' | number | (list)}
[FORMAT [(code)] | DUMP [(code)]]]

Automatically displays values of variables in the monitor window.

Notes:
1. Valid in full screen mode only.
2. Check the format and dump codes table on page 74.

BACKSPACE Command

BACKSPACE
{number | [qual. ] integer-variable |
[qual.]integer-array-element}

Positions a sequentially accessed external file at the beginning of the previous record.

CLOSE Command

CLOSE
{number | [qual. ] integer-variable |
[qual.]integer-array-element}

Disconnects a sequential external file from a unit.
Interactive Debug Command Syntax

COLOR Command

COLOR

Customizes color, highlighting, and intensity on the main debugging panel.

Note: Valid in full screen mode only.

DBCS Command

DBCS

[YES | NO]

Specifies whether X'0F' and X'0E' are interpreted as the double-byte character set shift characters in input and output.

DESCRIBE Command

DESCRIBE

[[qual.] name | * | (name-list)]

[PRINT]

Displays data types of scalar variables and arrays, and dimension information for arrays.

DOWN Command

DOWN

[number | PAGE | HALF | CSR | DATA | MAX]

Scrolls the contents of a window so that lines below those currently displayed in the window can be seen.

Note: Valid in full screen mode only.

ENDDEBUG Command

ENDDEBUG

[SAMPLE[(msecs)]

[MAXSAMP(n[,STOP])]

[CALLED]]

Discontinues debugging and continues program processing. Also initiates program sampling.
Interactive Debug Command Syntax

ENDFILE Command

ENDFILE
{number | [qual.] integer-variable |
[qual.]integer-array-element}

Writes an end-of-file record on a sequentially accessed external file.

ERROR Command

ERROR
{error | error.error | (error-list)}
[MSG | NOMSG]
(EXIT | NOEXIT]

Selects diagnostic options for run-time errors.

FIXUP Command

FIXUP
[ARG1(value)]
[ARG2(value)]

Specifies corrective action.

GO Command

GO
[[qual.] {number | EXIT}]

Resumes processing.

HALT Command

HALT
[OFF | STMT | GOTO | ENTRY | IMMED]

Causes processing to be suspended for every statement of a given class, or at a specific point in a command list.
Interactive Debug Command Syntax

HELP Command—CMS or TSO Full Screen Mode

HELP
   [command | vecmsg-id]

Requests online information about interactive debug commands, common tasks, and vector messages contained in the vector report listing, as well as a task-oriented tutorial.

HELP Command—CMS Line Mode

HELP
   [command [ALL | (DESC | (PARM | (FORM | vecmsg-id]

Requests online information about interactive debug commands, common tasks, and vector messages contained in the vector report listing, as well as a task-oriented tutorial.

HELP Command—TSO Line Mode

HELP
   [command [ALL | FUNCTION | SYNTAX
   | OPERANDS [(keyword-list)] | vecmsg-id]

Requests online information about interactive debug commands, common tasks, and vector messages contained in the vector report listing, as well as a task-oriented tutorial.

IF Command

IF
   (condition) command

Tests a condition.

LEFT Command

LEFT
   [number | PAGE | HALF | CSR | DATA | MAX]

Scrolls the contents of a window so that columns to the left of those currently displayed in the window can be seen.

Note: Valid in full screen mode only.
Interactive Debug Command Syntax

LIST Command

LIST

\{qual. \ name \ [\ qual. \ name]\ 
| \* | \string | number \ (list)\ 
[PRINT]
[FORMAT \ ((code)) | DUMP \ ((code))]\}

Displays values of variables.

Note:  Check the format and dump codes table on page 74.

LISTBRKS Command

LISTBRKS [PRINT]

Lists all breakpoints and WHEN conditions currently set, and the current
HALT status.

LISTFREQ Command

LISTFREQ

\{\ qual. \ 
\{number \ [\ qual. \ number] | ENTRY | EXIT\ 
| (number/ENTRY/EXIT list)\ 
[ZEROFREQ] [PRINT]\}

Lists the number of times statements processed.

LISTINGS Command

LISTINGS

Displays the interactive debug listings panel.

Note:  Valid in full screen mode only.
Interactive Debug Command Syntax

LISTSAMP Command—Statement

LISTSAMP
   {[[qual.] number:[[qual.] number]
      | [qual.]ENTRY | [qual.]* | (list) | **}
      [DIRECT][CALLED][ALL]
      [TOP([n])][PRINT]

Lists sampling counts by statement.

LISTSAMP Command—Program Unit

LISTSAMP
   {unit-name | (unit-name-list) | * } SUMMARY
      [DIRECT | CALLED | ALL]
      [TOP([n])][PRINT]

Lists sampling counts by program unit.

LISTSAMP Command—DO Loop

LISTSAMP
   {[[qual.] number:[[qual.] number]
      | [qual.]* | (list) | **}
      DOILOOP [DIRECT | CALLED | ALL ]
      [TOP([n])][PRINT]

Lists sampling counts by DO loop.

LISTSUBS Command

LISTSUBS
   [PRINT]

Lists information about all debuggable program units in the running load module.

LISTTIME Command—Program Unit

LISTTIME
   [PRINT]

Displays timing information for program units.
Interactive Debug Command Syntax

LISTTIME Command—DO Loop

LISTTIME

\[
\{ [qual.] \text{number} [: [qual.] \text{number}] | \\
\text{[qual.]} \ast | (\text{list}) | ** \}
\]

\text{DOLOOP [PRINT]}

Displays timing information for analyzable DO loops.

LISTVEC Command

LISTVEC

\[
\{ [qual.] \text{number} [: [qual.] \text{number}] | \\
\text{[qual.]} \ast | (\text{list}) | ** \}
\]

\text{[TOP [**(n)] [PRINT]}

Displays DO loop length and stride information.

MOVECURS Command

MOVECURS

Toggles the cursor between the command line and its most recent position in the main debugging panel.

\text{Note: Valid in full screen mode only.}

NEXT Command

NEXT

Suspends program execution at the next statement, entry, or exit with a debugging hook.

OFF Command

OFF

\[
\{ [qual.] \text{number} [: [qual.] \text{number}] | \\
\text{ENTRY | EXIT} | \ast | (\text{number|ENTRY/EXIT list})
\]

Removes breakpoints in the currently qualified or specified program unit.
Interactive Debug Command Syntax

**OFFWN Command**

OFFWN

\[
\text{condition name} | \ast | (\text{condition-name-list})
\]

Turns off WHEN condition monitoring.

**POSITION Command**

POSITION

\[
\text{number}
\]

Positions the cursor in the log window at a specified log line, in the source window at a given ISN or sequence number, or in the monitor window at a specified monitor line.

**Note:** Valid in full screen mode only.

**PREVDISP Command**

PREVDISP

Redisplays the previous panel displayed by the application program.

**Note:** Valid in full screen mode only.

**PROFILE Command**

PROFILE

Displays a profile panel to change current conditions or profile settings.

**Note:** Valid in full screen mode only.

**PURGE Command**

PURGE

Purges output.

**QUALIFY Command**

QUALIFY

\[
\text{[program]}
\]

Changes or displays the current qualification.
Interactive Debug Command Syntax

QUIT Command
QUIT
Ends the debugging session.

RECONNECT Command
RECONNECT
{number | [qual. ]integer-variable | [qual.]integer-array-element}
Resets a file to its original (preconnected) condition.

REFRESH Command
REFRESH
[ON | OFF]
Controls whether the IAD panel is refreshed.
Note: Valid in full screen mode only.

RESTART Command
RESTART
Restarts the debugging session while maintaining the log file.
Note: Valid in full screen mode only.

RESTORE Command
RESTORE
Restores the source window to the last point of execution.
Note: Valid in full screen mode only.

RETRIEVE Command
RETRIEVE
Redisplays the last command specified on the command line.
Note: Valid in full screen mode only.
Interactive Debug Command Syntax

REWIND Command

REWIND
  \{number \mid [qual.] \text{integer-variable}
     \mid [qual.] \text{integer-array-element}\}

Positions a sequentially accessed external file at the beginning of its first record.

RIGHT Command

RIGHT
  \{number \mid \text{PAGE} \mid \text{HALF} \mid \text{CSR} \mid \text{DATA} \mid \text{MAX}\}

Scrolls the contents of a window so that columns to the right of those currently displayed in the window can be seen.

SEARCH Command

SEARCH
  \[/string[/]]

Searches either the source, monitor, or log window for a given character string.

\textbf{Note:} Valid in full screen mode only.

SET Command

SET
  \{[qual.] \text{name}=\text{value} [,\text{value}...]\}

Assigns values to variables.

SIZE Command

SIZE
  \{\text{SOURCE} \mid \text{MONITOR} \mid \text{LOG}\}

Resizes the windows on the main debugging panel.

\textbf{Note:} Valid in full screen mode only.
Interactive Debug Command Syntax

**STEP Command**

STEP

    [number]

Processes one or more statements, then suspends processing. In full screen mode, processing is animated.

**SYSCMD Command**

SYSCMD

    [system-command]

Issues system commands during debugging.

*Note:* The abbreviations CMS and TSO are recognized and executed, but the command entered does not appear in the session log.

**TERMIO Command**

TERMIO

    [IAD | LIBRARY]
    [MSG [(userid)] | NOMSG]

Selects I/O routines for terminal I/O from the VS FORTRAN program.

**TIMER Command—Program Unit**

TIMER

    { * | program-unit-name | (program-unit-name-list) }
    [ON | OFF | RESET]

Controls timing by program unit.

**TIMER Command—DO Loop**

TIMER

    {qual.| number [:qual.| number
    | qual.| * | (list) | **}
    DOLOOP [ON | OFF | RESET]

Controls timing by DO loop.
Interactive Debug Command Syntax

TRACE Command

TRACE
  [GOTO | ENTRY | OFF]
  [PRINT]
Traces statement branches and subprogram calls.

UP Command

UP
  [number | PAGE | HALF | CSR | DATA | MAX]
Scrolls the contents of a window so that lines above those currently displayed in the window can be seen.

Note: Valid in full screen mode only.

VECSTAT Command

VECSTAT
  {[(qual.) number [: (qual.)] number
   | [qual.] * | (list) | *:*}
  [ON | OFF | RESET]
Activates, deactivates, or resets DO loop length and stride recording.

WHEN Command

WHEN
  condition-name [((condition) | variable]
Sets up monitoring of a condition.

WHERE Command

WHERE
  [TRBACK] [FLOW] [PRINT]
Displays statement at which processing is suspended.
Interactive Debug Command Syntax

**WINDOW Command for Changing Configuration**

WINDOW

Changes the window configuration of the debugging session.

**Note:** Valid in full screen mode only.

**WINDOW Command for Saving Configuration**

WINDOW
   SAVE

Saves the window configuration of the debugging session.

**Note:** Valid in full screen mode only.

**WINDOW Command for Opening or Closing**

WINDOW
   {OPEN | CLOSE}
   {SOURCE | MONITOR | LOG}

Opens or closes a specified window of the debugging session.

**Note:** Valid in full screen mode only.

**ZOOM Command**

ZOOM
   [SOURCE | MONITOR | LOG]

Toggles between displaying one window and a configuration of windows.
### Interactive Debug

#### Format and Dump Codes for the AUTOLIST and LIST Commands

<table>
<thead>
<tr>
<th>Code</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>LOGICAL*1</td>
</tr>
<tr>
<td>L4</td>
<td>LOGICAL*4</td>
</tr>
<tr>
<td>I2</td>
<td>INTEGER*2</td>
</tr>
<tr>
<td>I4</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>R4</td>
<td>REAL*4</td>
</tr>
<tr>
<td>R8</td>
<td>REAL*8</td>
</tr>
<tr>
<td>R16</td>
<td>REAL*16</td>
</tr>
<tr>
<td>C8</td>
<td>COMPLEX*8</td>
</tr>
<tr>
<td>C16</td>
<td>COMPLEX*16</td>
</tr>
<tr>
<td>C32</td>
<td>COMPLEX*32</td>
</tr>
<tr>
<td>L</td>
<td>LOGICAL with size closest to internal data size</td>
</tr>
<tr>
<td>I</td>
<td>INTEGER with size closest to internal data size</td>
</tr>
<tr>
<td>R</td>
<td>REAL with size closest to internal data size</td>
</tr>
<tr>
<td>C</td>
<td>COMPLEX with size closest to internal data size</td>
</tr>
<tr>
<td>X[nnn]</td>
<td>Hexadecimal with nnn bytes per data item (default to internal data size)</td>
</tr>
<tr>
<td>Z[nnn]</td>
<td>Hexadecimal with nnn bytes per data item (default to Z4)</td>
</tr>
<tr>
<td>A[nnn]</td>
<td>CHARACTER with nnn bytes per data item (default to internal data size)</td>
</tr>
<tr>
<td>H[nnn]</td>
<td>CHARACTER with nnn bytes per data item (default to continuous full line output)</td>
</tr>
</tbody>
</table>
Valid SET Command Assignments

Note that values must be those allowable for the type of variable being set.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type of Value Set to</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>Another scalar variable</td>
<td>ALPHA=BETA, ALPHA=−BETA</td>
</tr>
<tr>
<td>(REAL, INTEGER, or COMPLEX)</td>
<td>An array element</td>
<td>ALPHA=A(3) ALPHA=−A(3)</td>
</tr>
<tr>
<td></td>
<td>A numeric constant</td>
<td>NUM=7</td>
</tr>
<tr>
<td>LOGICAL</td>
<td>Another logical variable</td>
<td>LOG1=LOG2</td>
</tr>
<tr>
<td></td>
<td>An array element</td>
<td>LOG1=LOG(2)</td>
</tr>
<tr>
<td></td>
<td>A logical constant</td>
<td>LOG=.TRUE.</td>
</tr>
<tr>
<td>CHARACTER</td>
<td>Another character variable</td>
<td>CHAR1=CHAR2</td>
</tr>
<tr>
<td></td>
<td>An array element</td>
<td>CHAR1=CHAR(2)</td>
</tr>
<tr>
<td></td>
<td>A character constant</td>
<td>MSG='HELLO'</td>
</tr>
<tr>
<td></td>
<td>A substring</td>
<td>A(1:3)= 'ABC'</td>
</tr>
<tr>
<td>Array element</td>
<td>Another array element</td>
<td>A(4)=B(1) AR(2,2)=−AR(5,5)</td>
</tr>
<tr>
<td></td>
<td>A scalar variable</td>
<td>C(7)=RATE C(8)=−TIME</td>
</tr>
<tr>
<td></td>
<td>A constant</td>
<td>D(I,J)=0.0</td>
</tr>
<tr>
<td>Contiguous array elements</td>
<td>Value, value,... (Values can be variables, array elements, or constants; multiple assignments of a value can be entered as n*value.)</td>
<td>A=3<em>1.0,4</em>0.0,7.2, 5.,ACCL,8.5E9, B(J,K)= 'C', 3* 'Q', 'X'</td>
</tr>
</tbody>
</table>
We'd Like to Hear from You

VS FORTRAN Version 2
Reference Summary
Release 6
Publication No. SX26-3751-07

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
Reference Summary
Release 6
Publication No. SX26-3751-07

How satisfied are you with the information in this book?

<table>
<thead>
<tr>
<th></th>
<th>Very Satisfied</th>
<th>Satisfied</th>
<th>Neutral</th>
<th>Dissatisfied</th>
<th>Very Dissatisfied</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technically accurate</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Complete</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to find</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Easy to understand</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Well organized</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Applicable to your tasks</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Grammatically correct and consistent</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Graphically well designed</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Overall satisfaction</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

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. ____________________________ ____________________________