September 1984

VAX FORTRAN
Language Summary

Order No. AV-M763B-TE

HOW TO ORDER ADDITIONAL DOCUMENTATION

In Continental USA and Puerto Rico call 800-258-1710

In New Hampshire, Alaska, and Hawaii call 603-884-6660

In Canada call 613-234-7726 (Ottawa-Hull)
800-267-6146 (all other Canadian)

DIRECT MAIL ORDERS (USA & PUERTO RICO)*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire 03061

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575)

DIRECT MAIL ORDERS (CANADA)

Digital Equipment of Canada Ltd.
940 Belfast Road
Ottawa, Ontario K1G 4C2
Attn: A&SG Business Manager

DIRECT MAIL ORDERS (INTERNATIONAL)

Digital Equipment Corporation
A&SG Business Manager
c/o Digital's local subsidiary or approved distributor

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532

digital equipment corporation · maynard, massachusetts

First Printing, October 1982
Revised, September, 1984

Contents

Tables

Figure

Preface

Manual Objectives

The *VAX FORTRAN Language Summary* is intended as a quick reference for use when you are writing FORTRAN programs, and not as a formal or complete description of the language.

Associated Documents

The following documents contain more detailed information:

- The *Programming in VAX FORTRAN* contains detailed reference information on the VAX FORTRAN language elements summarized in this booklet.
- The *VAX FORTRAN User's Guide* contains detailed VAX FORTRAN information on using the VAX/VMS operating system.

For a list of other related VAX/VMS documents, see the *VAX/VMS Master Index* and the *Introduction to the VAX/VMS Document Set*.

Symbols and Conventions

- Brackets ([ ]) enclose optional language elements. Long brackets enclose lists of elements from which one and only one element may be chosen.
- Braces ({ }) enclose lists of items from which one and only one item must be chosen.
- Horizontal ellipses (,...) indicate that additional parameters, options, or values can be entered. When a comma precedes the ellipses, it indicates that successive items must be separated by commas.
- Text in blue ink describes language features that are VAX extensions to the FORTRAN-77 standard.

Command Formats

FORTRAN Command

The FORTRAN command compiles one or more FORTRAN source programs into separate object modules, or concatenates and compiles one or more programs into a single object module. You can also specify text libraries to search for modules specified by INCLUDE statements in the source files.

By default, FORTRAN searches:

1. Libraries specified on the FORTRAN command in the order they are specified.
2. The library associated with the logical name FOR$LIBRARY, if a logical name assignment exists for FOR$LIBRARY.
3. FOR$LIBRARY:FORSYSDEF.TLB.

FORTRAN Command Format

FORTRAN[/qualifier...] file-spec[/qualifier...],...

Command Qualifiers	Default
/[NO]CHECK[=option]	/CHECK=(NOBOUNDS, OVERFLOW, NOUNDERFLOW)
/CONTINUATIONS=n	/CONTINUATIONS=19
/[NO]CROSS_REFERENCE	/NOCROSS_REFERENCE
/[NO]DEBUG[=option]	/DEBUG=(NOSYMBOLS,TRACEBACK)
/[NO]D_LINES	/NOD_LINES
/DML	
/[NO]EXTEND_SOURCE	/NOEXTEND_SOURCE
/[NO]F77	/F77

Command Qualifiers	Default
/[NO]G_FLOATING	/NOG_FLOATING
/[NO]I4	/I4
/LIBRARY	
/[NO]LIST[=file-spec]	/NOLIST (interactive default) /LIST (batch default)
/[NO]MACHINE_CODE	/NOMACHINE_CODE
/[NO]OBJECT[=file-spec]	/OBJECT
/[NO]OPTIMIZE	/OPTIMIZE
/[NO]SHOW[=(option[,...])]	/SHOW=(NOINCLUDE, NODICTIONARY, MAP, NOPREPROCESSOR, SINGLE)
/[NO]STANDARD[=(option[,...])]	/NOSTANDARD=(SYNTAX,NOSOURCE_FORM)
/[NO]WARNINGS[=(option[,...])]	/WARNINGS=GENERAL,NODECLARATION)

File Qualifier

/LIBRARY

file-spec[,...]

Specifies one or more FORTRAN source files to be compiled and, optionally, libraries to be searched for INCLUDE files that are referenced in the source file(s).

You must separate multiple input file specifications with either commas (,) or plus signs (+). The commas and plus signs have different meanings, as follows:

- Commas delimit FORTRAN source files to be compiled separately. FORTRAN compiles each file and creates an object module for each.
- Plus signs delimit files to be concatenated or libraries containing INCLUDE files. FORTRAN compiles the source files as a single file and creates one object module. Library file specifications must be preceded by a plus sign and qualified with the /LIBRARY qualifier. If no file type is specified, FORTRAN assumes the default type of TLB.

LINK Command

The LINK command binds one or more object modules into an executable image that can be executed with the RUN command. You can also specify object module libraries to be searched for modules and symbols referenced, but not defined in the object module(s) being linked.

By default, the linker searches:

1. Libraries specified in the LINK command in the order they are specified.
2. Libraries defined by the logical names LNK$LIBRARY, LNK$LIBRARY_1, LNK$LIBRARY_2, and so on, that may exist in the process, group, or system logical name tables.
3. The default system library.

LINK Command Format

```
LINK[/qualifier...] file-spec[/qualifier...],...
```

Command Qualifiers	Default
/BRIEF	
/[NO]CONTIGUOUS	/NOCONTIGUOUS
/[NO]CROSS_REFERENCE	/NOCROSS_REFERENCE
/[NO]DEBUG	/NODEBUG
/[NO]EXECUTABLE[=file-spec]	/EXECUTABLE
/FULL	
/HEADER	
/[NO]MAP[=file-spec]	/NOMAP (interactive) /MAP (batch)
/[NO]SHAREABLE[=file-spec]	/NOSHAREABLE
/[NO]SYSLIB	/SYSLIB
/[NO]SYSSHR	/SYSSHR
/[NO]TRACEBACK	/TRACEBACK
/[NO]USERLIBRARY[=(table[,...])]	/USERLIBRARY=ALL

File Qualifiers

/INCLUDE=(module-name[,...])
/LIBRARY
/OPTIONS
/SELECTIVE_SEARCH

file-spec[,...]

Specifies one or more input files. The input files can be object modules to be linked, libraries to be searched for external references or from which specific modules are to be included, shareable images to be included in the output image, or option files to be read by the linker. If you specify multiple input files, separate the file specifications with commas (,) or plus signs (+). In either case, the linker creates a single image file.

If you do not specify a file type in an input file specification, the linker supplies default file types, based on the nature of the file. All object modules are assumed to have file types of OBJ.

No wildcard characters are allowed in the file specification.

LIBRARY Command

The LIBRARY command creates and maintains libraries of text modules and object modules. Text libraries contain modules having FORTRAN source statements, and are included during compilation if specified in INCLUDE statements. Object module libraries contain object modules to be included in images during linking.

The LIBRARY command assumes, by default, that it is operating on an object module library. You must specify /TEXT to perform an operation on a text library.

LIBRARY Command Format

```
LIBRARY[/qualifier...] library-file-spec[/qualifier] [input-file-spec[,...]]
```

Command Qualifiers

/COMPRESS[=option[,...])]
/CREATE[=(option[,...])]
/CROSS_REFERENCE[=(option[,...])]
/DELETE=(module[,...])
/EXTRACT=(module[,...])
/FULL
/HELP
/INSERT
/MACRO
/OBJECT
/ONLY=(module[,...])
/OUTPUT=file-spec
/REMOVE=(symbol[,...])
/REPLACE
/SELECTIVE_SEARCH
/TEXT
/WIDTH=n

Command Qualifiers	Default
/[NO]GLOBALS	/GLOBALS
/[NO]LIST[=file-spec]	/NOLIST
/[NO]LOG	/NOLOG
/[NO]NAMES	/NONAMES
/[NO]SQUEEZE	/SQUEEZE

File Qualifiers

/MODULE=module-name

library-file-spec

Specifies the name of the library you want to create or modify.

If the file specification does not include a file type, the LIBRARY command assumes a default type of OLB.

input-file-spec[,...]

Specifies the names of one or more files that contain modules you want to replace or insert into the specified library.

When you use the /CREATE qualifier, the input file specification is optional. When you use the /EXTRACT qualifier, an input file specification is not permitted.

If a file specification does not include a file type, the LIBRARY command assumes a default file type of OBJ. You can control the default file type by specifying one of the following qualifiers on the LIBRARY command:

Qualifier	Default File Type
/HELP	HLP
/MACRO	MAR
/OBJECT	OBJ
/TEXT	TXT
/SHARE	EXE

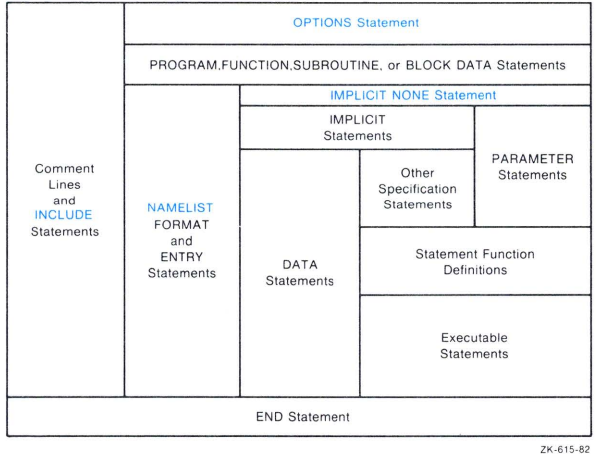
Language Summary

Order of Statements

The following figure shows the required order of statements in a FORTRAN program unit. In this figure, vertical lines separate statement types that can be interspersed. For example, DATA statements can be interspersed with executable statements. Horizontal lines indicate statement types that cannot be interspersed with executable statements.

Statements in the 'executable' category include: ACCEPT, ASSIGN, assignment statements, BACKSPACE, CALL, CLOSE, CONTINUE, DELETE, DO and END DO, ELSE, ENDFILE, FIND, GO TO, IF and END IF, INQUIRE, OPEN, PAUSE, PRINT, READ, RETURN, REWIND, REWRITE, STOP, TYPE, UNLOCK, and WRITE.

Statements in the 'specification' category include: COMMON, DICTIONARY, DIMENSION, EQUIVALENCE, EXTERNAL, INTRINSIC, RECORD, SAVE, structure declarations, type declarations, and VOLATILE.



Required Order of Statements and Lines

Statements

accept-statement:

ACCEPT See READ

assign-statement:

ASSIGN s TO v

assignment-statement:

v = e

backspace-statement:

BACKSPACE ([UNIT=]u[,ERR=s][,IOSTAT=ios])

BACKSPACE u

blockdata-statement:

BLOCK DATA [nam]

call-statement:

CALL sub[([a][,[a]]...)]

close-statement:

CLOSE ([UNIT=]u[,p][,ERR=s][,IOSTAT=ios])

p is one of the following parameters:

$$\left\{\begin{matrix}\text{STATUS}\\\text{DISPOSE}\\\text{DISP}\end{matrix}\right\} = \left\{\begin{matrix}\text{'SAVE'}\\\text{'KEEP'}\\\text{'DELETE'}\\\text{'PRINT'}\\\text{'SUBMIT'}\\\text{'PRINT/DELETE'}\\\text{'SUBMIT/DELETE'}\end{matrix}\right\}$$

common-statement:

```
COMMON [/[cb]/]nlist[[,] /[cb]/nlist]...
```

continue-statement:

```
CONTINUE
```

data-statement:

```
DATA nlist/clist/[[,] nlist/clist/]...
```

data-type-declaration (see type-declaration)

decode-statement:

```
DECODE (c,f,b[,ERR=s][,IOSTAT=ios]) [iolist]
```

definefile-statement:

```
DEFINE FILE u(m,n,U,v)[,u(m,n,U,v)]
```

delete-statement:

```
DELETE ([UNIT=]u[,REC=r][,ERR=s][,IOSTAT=ios])
DELETE (u´r[,ERR=s][,IOSTAT=ios])
```

dictionary-statement:

```
DICTIONARY ´cdd-path[/[NO]LIST]´
```

dimension-statement:

```
DIMENSION a([d1:]d2)[,a([d1:]d2)]...
```

do-statement:

```
DO [s[,]] v=e1,e2[,e3]
DO [s[,]] WHILE (e)
```

else-statement:

```
ELSE
ELSE IF (e) THEN
```

encode-statement:

```
ENCODE (c,f,b[,ERR=s][,IOSTAT=ios]) [iolist]
```

end-statement:

```
END
END DO
END IF
END MAP
END STRUCTURE
END UNION
```

endfile-statement:

```
ENDFILE ([UNIT=]u[,ERR=s][,IOSTAT=ios])
ENDFILE u
```

entry-statement:

```
ENTRY nam[([p[,p]...])]
```

equivalence-statement:

```
EQUIVALENCE (nlist)[,(nlist)]...
```

external-statement:

```
EXTERNAL v[,v]...
EXTERNAL *v[,*v]...
```

find-statement:

FIND ([UNIT=]u,REC=r[,ERR=s][,IOSTAT=ios])

FIND (u´r[,ERR=s][,IOSTAT=ios])

format-statement:

FORMAT (field-spec[,...])

function-statement:

[typ] FUNCTION nam[*m][([p[,p]...])]

goto-statement:

Go TO s

GO TO (slist)[,] e

GO TO v[[,](slist)]

if-statement:

```
IF (e) s1,s2,s3
IF (e) st
IF (e1) THEN
        block
ELSE IF (e2) THEN
        block
        ELSE
        block
END IF
```

implicit-statement:

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

IMPLICIT NONE

include-statement:

INCLUDE ´file-spec[/[NO]LIST]´

INCLUDE ´[file-spec](module-name)[/[NO]LIST]´

inquire-statement:

INQUIRE(par[,par]...)

par is a keyword specification having one of these forms:

key
key = value

key is a keyword as described below.

value depends on the keyword.

Keyword	Values
inputs	
FILE	fin
UNIT	e
DEFAULTFILE	fin
outputs	
ACCESS	cv
BLANK	cv
CARRIAGECONTROL	cv
DIRECT	cv
ERR	s
EXIST	lv
FORM	cv
FORMATTED	cv

Keyword	Values
Outputs	
IOSTAT	v
KEYED	cv
NAME	cv
NAMED	lv
NEXTREC	v
NUMBER	v
OPENED	lv
ORGANIZATION	cv
RECL	v
RECORDTYPE	cv
SEQUENTIAL	cv
UNFORMATTED	cv

instrinsic-statement:

INTRINSIC v[,v]...

map-declaration (see union-declaration)

namelist-statement:

NAMELIST /grp/ nlist[[,] /grp/ nlist]...

open-statement:

OPEN (par[,par]...)

par is a keyword specification in one of the following forms:

key
key = value

key is a keyword, as described below.

value depends on the keyword.

Keyword	Values
ACCESS	'SEQUENTIAL 'DIRECT' 'KEYED' 'APPEND'
ASSOCIATEVARIABLE	v
BLOCKSIZE	e
BLANK	'NULL' 'ZERO'
BUFFERCOUNT	e
CARRIAGECONTROL	'FORTRAN' 'LIST' 'NONE'
DEFAULTFILE	c
DISP	(same as DISPOSE)

DISPOSE	'KEEP' or 'SAVE' 'PRINT' 'DELETE' 'SUBMIT' 'SUBMIT/DELETE' 'PRINT/DELETE'
ERR	s
EXTENDSIZE	e
FILE	c
FORM	'FORMATTED' 'UNFORMATTED'
INITIALSIZE	e
IOSTAT	v
KEY	keyspec
MAXREC	e
NAME	(same as FILE)

Keyword	Values
NOSPANBLOCKS	—
ORGANIZATION	'SEQUENTIAL'
	'RELATIVE'
	'INDEXED'
READONLY	—
RECL	e
RECORDSIZE	(same as RECL)
RECORDTYPE	'FIXED'
	'VARIABLE'
	'SEGMENTED'
	'STREAM'
	'STREAM_CR'
	'STREAM_LF'

SHARED	—
STATUS	'OLD'
	'NEW'
	'SCRATCH'
	'UNKNOWN'
TYPE	(same as STATUS)
UNIT	e
USEROPEN	p

options-statement:

OPTIONS qualifier[,qualifier...]

/NOCHECK

```
                    ⎧ALL
                    ⎪([NO]OVERFLOW)
/CHECK=             ⎨([NO]BOUNDS)
                    ⎪([NO]UNDERFLOW)
                    ⎩NONE
/[NO]EXTEND_SOURCE
/[NO]F77
/[NO]G_FLOATING
/[NO]I4
```

parameter-statement:

PARAMETER (p=c[,p=c]...)

pause-statement:

PAUSE [disp]

print-statement:

PRINT See WRITE

program-statement:

PROGRAM nam

read-statement:

READ Statement — Formatted Sequential Access

READ (extu,fmt[,err][,iostat][,end]) [iolist]

READ f[,iolist]

ACCEPT f[,iolist]

READ Statement — List-Directed Sequential Access

READ (extu,*[,err][,iostat][,end]) [iolist]

READ *[,iolist]

ACCEPT *[,iolist]

READ Statement — Namelist-Directed Sequential Access

READ (extu,nml[,err][,iostat][,end])

READ n

ACCEPT n

READ Statement — Unformatted Sequential Access

READ (extu[,err][,iostat][,end]) [iolist]

READ Statement — Formatted Direct Access

READ (extu,fmt,rec[,err][,iostat]) [iolist]

READ (u´r,fmt[,err][,iostat]) [iolist]

READ Statement — Unformatted Direct Access

READ (extu,rec[,err][,iostat]) [iolist]

READ (u´r[,err][,iostat]) [iolist]

READ Statement — Formatted Indexed

READ (extu,fmt,keyspec[,keyid][,err][,iostat]) [iolist]

READ Statement — Unformatted Indexed

READ (extu,keyspec[,keyid][,err][,iostat]) [iolist]

READ Statement — Formatted Internal

READ (intu,fmt[,err][,iostat][,end]) [iolist]

READ Statement — List-Directed Internal

READ (intu,*[,err][,iostat][,end]) [iolist]

record-statement:

```
RECORD    /struc/rnlist
         [,/struc/rnlist]
              .
              .
              .
         [,/struc/rnlist]
```

return-statement:

RETURN [i]

rewind-statement:

```
REWIND ([UNIT=]u[,ERR=s][,IOSTAT=ios])
REWIND u
```

rewrite-statement:

REWRITE Statement — Formatted Indexed

```
REWRITE (extu,fmt[,err][,iostat]) [iolist]
```

REWRITE Statement — Unformatted Indexed

```
REWRITE (extu[,err][,iostat]) [iolist]
```

save-statement:

```
SAVE [a[,a]...]
```

Statement Function:

```
fun([p[,p]...]) = e
```

stop-statement:

```
STOP [disp]
```

structure-declaration-block:

```
STRUCTURE [/struc/] [fnlist]
          fdcl
          [fdcl]
            .
            .
            .
          [fdcl]
END STRUCTURE
```

subroutine-statement:

SUBROUTINE sub[([p[,p]...])]

type-statement See WRITE

type-declaration (character):

CHARACTER[*len[,]] v[*len][/clist/][,v[*len][/clist/]]...

type-declaration (numeric):

typ v[/clist/][,v[/clist/]]...

union-declaration:

```
UNION
        mdcl
       [mdcl]
          .
          .
          .
       [mdcl]
END UNION
```

where **mdcl** is:

```
        MAP
        fdcl
       [fdcl]
          .
          .
          .
       [fdcl]
  END MAP
```

unlock-statement:

UNLOCK ([UNIT=]u[,ERR=s][,IOSTAT=ios])

UNLOCK u

virtual-statement:

VIRTUAL a([d1:]d2)[,a([d1:]d2)]...

volatile-statement:

VOLATILE nlist

write-statement:

WRITE Statement — Formatted Sequential Access

WRITE (extu,fmt[,err][,iostat]) [iolist]

PRINT f[,iolist]

TYPE f[,iolist]

WRITE Statement — List-Directed Sequential Access

WRITE (extu,*[,err][,iostat]) [iolist]

PRINT *[,iolist]

TYPE *[,iolist]

WRITE Statement — Namelist-Directed Sequential Access

WRITE (extu,nml[,err][,iostat])

PRINT n

TYPE n

WRITE Statement — Unformatted Sequential Access

WRITE (extu[,err][,iostat]) [iolist]

WRITE Statement — Formatted Direct Access

WRITE (extu,rec,fmt[,err][,iostat]) [iolist]

WRITE (u´r,f[,err][,iostat]) [iolist]

WRITE Statement — Unformatted Direct Access

WRITE (extu,rec[,err][,iostat]) [iolist]

WRITE (u´r[,err][,iostat]) [iolist]

WRITE Statement — Formatted Internal

WRITE (intu,fmt[,err][,iostat]) [iolist]

WRITE Statement — List-Directed Internal

WRITE (intu,*[,err][,iostat]) [iolist]

VAX FORTRAN Expression Operators

The following lists the expression operators in each data type in order of descending precedence:

Precedence of Operators

Data Type	Operator	Operation	Operates Upon
Arithmetic	**	Exponentiation	Arithmetic or logical expressions
	*,/	Multiplication, division	
	+,-	Addition, subtraction, unary plus and minus	
Character	//	Concatenation	Character expressions
Relational	.GT.	Greater than	Arithmetic, logical, or character

	.GE.	Greater than or equal to	expressions (all relational operators have equal precedence)
	.LT.	Less than	
	.LE.	Less than or equal to	
	.EQ.	Equal to	
	.NE.	Not equal to	
Logical	.NOT.	.NOT.A is true if and only if A is false	Logical or integer expressions
	.AND.	A.AND.B is true if and only if A and B are both true	

Precedence of Operators (Cont.)

Data Type	Operator	Operation	Operates Upon
	.OR.	A.OR.B is true if either A or B or both are true	
	.EQV.	A.EQV.B is true if and only if A and B are both true or A and B are both false	.EQV., .NEQV., and .XOR. have equal priority
	.NEQV.	A.NEQV.B is true if and only if A is true and B is false or B is true and A is false	
	.XOR.	Same as .NEQV.	

Parentheses may be used to group operands so that they are evaluated irrespective of the precedence of operators.

VAX Symbolic Debugger Command Summary

Debugger Command Qualifiers

@filespec
ALLOCATE n-bytes
ATTACH process-name
CALL routine [(arg [,arg ...])]
CANCEL ALL

CANCEL BREAK [/BRANCH | /CALL | /EXCEPTION | /INSTRUCTION [=opcode] | /LINE | /MODIFY] {/ALL | breakpt [,breakpt ...]}

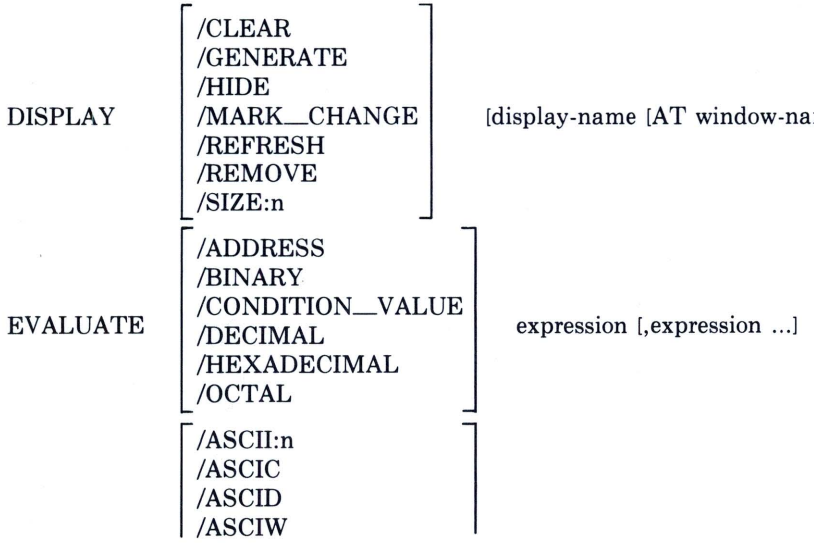
CANCEL DISPLAY {/ALL | display-name}

```
CANCEL EXCEPTION BREAK
CANCEL MODE
CANCEL MODULE      { /ALL        }
                   { module-name }
CANCEL RADIX   [ /OVERRIDE ]
CANCEL SCOPE
CANCEL SOURCE  [ /MODULE=module-name ]
                   [ /BRANCH                  ]
                   [ /CALL                    ]
CANCEL TRACE       [ /EXCEPTION               ]   { /ALL                    }
                   [ /INSTRUCTION [=opcode]   ]   { tracept [,tracept ...] }
                   [ /LINE                    ]
                   [ /MODIFY                  ]
```

```
CANCEL TYPE/OVERRIDE

CANCEL WATCH    { /ALL                          }
                { watchpt [,watchpt ...]        }

CANCEL WINDOW   { /ALL                          }
                { window-name [,window-name ...] }

DECLARE name [:kind] [,name [:kind]]

DEFINE          [ /ADDRESS  ]
                [ /VALUE    ]
                [ /COMMAND  ]   symbol = expression [,symbol = expression ...]
                [ /GLOBAL   ]

DEFINE/KEY      [ /[NO]ECHO          ]
                [ /[NO]IF_STATE      ]
                [ /[NO]LOCK_STATE    ]
                [ /[NO]LOG           ]   keyname expression
                [ /[NO]SET_STATE     ]
                [ /[NO]TERMINATE     ]
```

DELETE/KEY [/ALL | /[NO]LOG | /[NO]STATE] keyname

DEPOSIT [/ASCII:n | /ASCIC | /ASCIW | /ASCIZ | /BYTE | /D_FLOAT | /FLOAT | /G_FLOAT | /H_FLOAT | /INSTRUCTION | /LONG | /OCTAWORD | /QUADWORD | /WORD] addr-expresn = expression

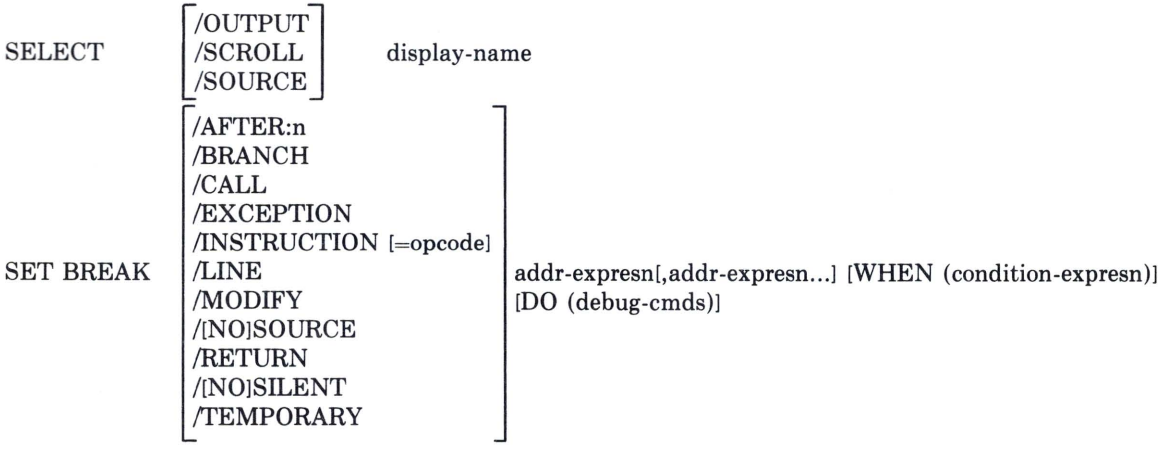

DISPLAY
/CLEAR
/GENERATE
/HIDE
/MARK_CHANGE
/REFRESH
/REMOVE
/SIZE:n
[display-name [AT window-name] [kind]] ,...
EVALUATE
/ADDRESS
/BINARY
/CONDITION_VALUE
/DECIMAL
/HEXADECIMAL
/OCTAL
expression [,expression ...]
/ASCII:n
/ASCIC
/ASCID
/ASCIW

EXAMINE	/ASCIZ /BINARY /BYTE /CONDITION_VALUE /D_FLOAT /DECIMAL /FLOAT /G_FLOAT /H_FLOAT /HEXADECIMAL /INSTRUCTION /LONG /OCTAL /OCTAWORD /QUADWORD /SOURCE /[NO]SYMBOL /WORD	addr-expresn [,addr-expresn ...]

EXIT
EXITLOOP [n-level]
FOR name = expression TO expression [BY expression] DO (debug-cmds) GO
HELP [topic]
IF language-expression THEN (debug-cmds) [ELSE (debug-cmds)]
REPEAT language-expression DO (debug-cmds)
SAVE old-display AS new-display

SCROLL [/BOTTOM /DOWN /LEFT /RIGHT /TOP /UP] [display-name]

SEARCH [/ALL /NEXT /IDENTIFIER /STRING] [range] string


SELECT
/OUTPUT
/SCROLL
/SOURCE
display-name
SET BREAK
/AFTER:n
/BRANCH
/CALL
/EXCEPTION
/INSTRUCTION [=opcode]
/LINE
/MODIFY
/[NO]SOURCE
/RETURN
/[NO]SILENT
/TEMPORARY
addr-expresn[,addr-expresn...] [WHEN (condition-expresn)]
[DO (debug-cmds)]

```
SET DISPLAY [/MARK_CHANGE]                [display-name [AT window] [kind]] ,...
            [/REMOVE     ]
            [/SIZE:n     ]

SET EXCEPTION BREAK
SET LANGUAGE language
SET LOG filespec

SET MARGIN { right-margin                 }
           { left-margin:right-margin     }
           { left-margin:                 }
           {            :right-margin     }

SET MAX_SOURCE_FILES n-files
SET MODE mode [,mode ...]
SET MODULE [ /ALLOCATE ] { /ALL                          }
                         { module-name [,module-name ...] }
```

SET OUTPUT [NO]LOG
[NO]SCREEN_LOG
[NO]TERMINAL
[NO]VERIFY

SET RADIX /INPUT, /OUTPUT, /OVERRIDE
BINARY
DECIMAL
DEFAULT
HEXADECIMAL
OCTAL

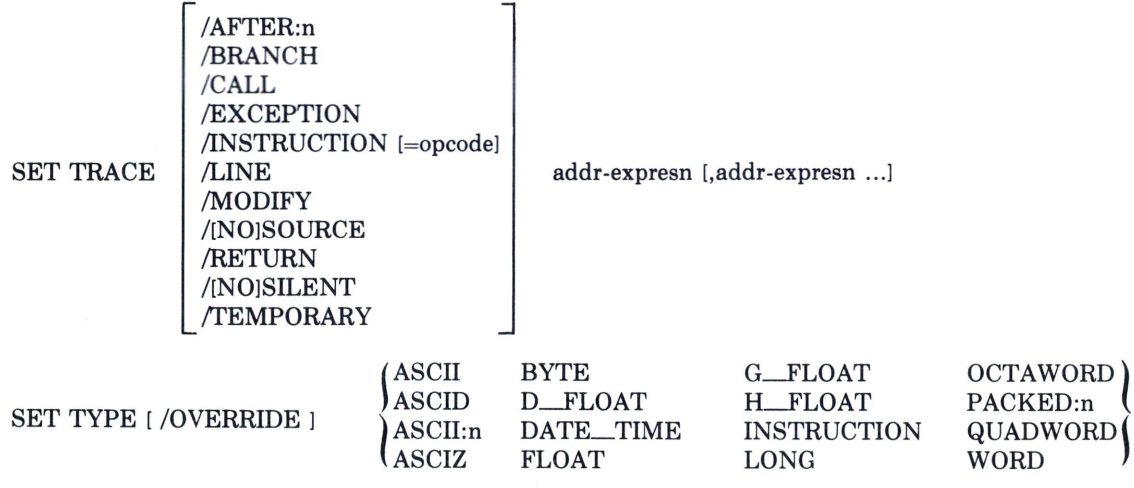
SET SCOPE [/MODULE] location [,location ...]

SET SEARCH ALL
NEXT
IDENTIFIER
STRING

SET SOURCE [/MODULE=module-name] filespec

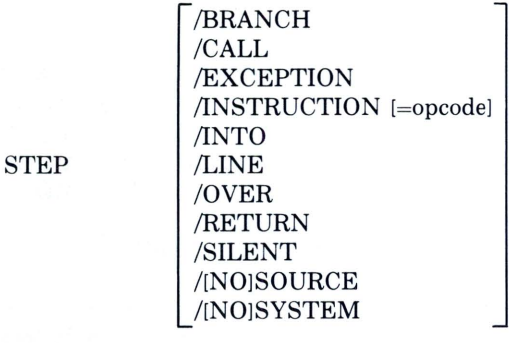
SET STEP

- BRANCH
- CALL
- EXCEPTION
- INSTRUCTION[=opcode]
- INTO
- LINE
- OVER
- RETURN
- [NO]SILENT
- [NO]SOURCE
- [NO]SYSTEM


/AFTER:n
/BRANCH
/CALL
/EXCEPTION
/INSTRUCTION [=opcode]
SET TRACE
/LINE
/MODIFY
/[NO]SOURCE
/RETURN
/[NO]SILENT
/TEMPORARY
addr-expresn [,addr-expresn ...]
SET TYPE [/OVERRIDE]
ASCII
ASCID
ASCII:n
ASCIZ
BYTE
D_FLOAT
DATE_TIME
FLOAT
G_FLOAT
H_FLOAT
INSTRUCTION
LONG
OCTAWORD
PACKED:n
QUADWORD
WORD

```
SET TERMINAL/WIDTH:n

SET WATCH [/AFTER:n
           /SILENT
           /SOURCE
           /TEMPORARY]   addr-expresn ,... [WHEN (condition)]

SET WINDOW name AT (line, line)
SHOW BREAK
SHOW CALLS [n-calls]
SHOW DISPLAY

SHOW KEY [/BRIEF
          /DIRECTORY
          /[NO]STATE]  {      /ALL      }
                       {key-name [,key-name ...]}
SHOW LANGUAGE
SHOW LOG
SHOW MARGINS
```

```
SHOW MAX_SOURCE_FILES
SHOW MODE
SHOW MODULE
SHOW OUTPUT
SHOW SCOPE
SHOW SEARCH
SHOW SOURCE
SHOW STEP
                 [ /ADDRESS ]
SHOW SYMBOL      [ /DIRECT  ]      symbol ,... [IN scope ,...]
                 [ /TYPE    ]
SHOW TRACE
SHOW TYPE [ /OVERRIDE ]
SHOW WATCH
SHOW WINDOW
SPAWN      [/NOWAIT]          dcl-command
```



SYMBOLIZE addr-expression
TYPE [module\]line[:line] ,...

WHILE language-expression DO (debug-cmds)

VAX FORTRAN Generic and Intrinsic Functions

Generic and Intrinsic Functions

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Square Root[1] $a^{1/2}$	1	SQRT	SQRT	REAL*4	REAL*4
			DSQRT	REAL*8	REAL*8
			QSQRT	REAL*16	REAL*16
			CSQRT	COMPLEX*8	COMPLEX*8
			CDSQRT	COMPLEX*16	COMPLEX*16
Natural Logarithm[2] $\log_e a$	1	LOG	ALOG	REAL*4	REAL*4
			DLOG	REAL*8	REAL*8
			QLOG	REAL*16	REAL*16
			CLOG	COMPLEX*8	COMPLEX*8
			CDLOG	COMPLEX*16	COMPLEX*16

Common Logarithm[2] $\log_{10}a$	1	LOG10	ALOG10 DLOG10 QLOG10	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Exponential e^a	1	EXP	EXP DEXP QEXP CEXP CDEXP	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16
Sine[3] Sin a	1	SIN	SIN DSIN QSIN CSIN CDSIN	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Sine[3] (degree) Sin a	1	SIND	SIND DSIND QSIND	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Cosine[3] Cos a	1	COS	COS DCOS QCOS CCOS CDCOS	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16	REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16
Cosine[3] (degree) Cos a	1	COSD	COSD DCOSD QCOSD	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16

Tangent[3] Tan a	1	TAN	TAN DTAN QTAN	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Tangent[3] (degree) Tan a	1	TAND	TAND DTAND QTAND	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Sine[4,5] Arc Sin a	1	ASIN	ASIN DASIN QASIN	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Sine (degree) Arc Sin a	1	ASIND	ASIND DASIND QASIND	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Cosine[4,5] Arc Cos a	1	ACOS	ACOS DACOS QACOS	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Arc Cosine (degree) Arc Cos a	1	ACOSD	ACOSD DACOSD QACOSD	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Tangent[5] Arc Tan a	1	ATAN	ATAN DATAN QATAN	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Tangent[5,7] (degree) Arc Tan a	1	ATAND	ATAND DATAND QATAND	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Arc Tangent[5,6] Arc Tan a_1/a_2	2	ATAN2	ATAN2 DATAN2 QATAN2	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16

Arc Tangent[5,7] (degree) Arc Tan a_1/a_2	2	ATAN2D	ATAN2D DATAN2D QATAN2D	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Hyperbolic Sine Sinh a	1	SINH	SINH DSINH QSINH	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Hyperbolic Cosine Cosh a	1	COSH	COSH DCOSH QCOSH	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16
Hyperbolic Tangent Tanh a	1	TANH	TANH DTANH QTANH	REAL*4 REAL*8 REAL*16	REAL*4 REAL*8 REAL*16

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Absolute Value[8]	1	ABS	IIABS	INTEGER*2	INTEGER*2
|a|			JIABS	INTEGER*4	INTEGER*4
			ABS	REAL*4	REAL*4
			DABS	REAL*8	REAL*8
			QABS	REAL*16	REAL*16
			CABS	COMPLEX*8	REAL*4
			CDABS	COMPLEX*16	REAL*8
		IABS	IIABS	INTEGER*2	INTEGER*2
			JIABS	INTEGER*4	INTEGER*4
Truncation[9,12]	1	INT	IINT	REAL*4	INTEGER*2
|a|			JINT	REAL*4	INTEGER*4
			IIDINT	REAL*8	INTEGER*2
			JIDINT	REAL*8	INTEGER*4

	IIQINT	REAL*16	INTEGER*2
	JIQINT	REAL*16	INTEGER*4
	—	COMPLEX*8	INTEGER*2
	—	COMPLEX*8	INTEGER*4
	—	COMPLEX*16	INTEGER*2
	—	COMPLEX*16	INTEGER*4
IDINT	IIDINT	REAL*8	INTEGER*2
	JIDINT	REAL*8	INTEGER*4
IQINT	IIQINT	REAL*16	INTEGER*2
	JIQINT	REAL*16	INTEGER*4
AINT	AINT	REAL*4	REAL*4
	DINT	REAL*8	REAL*8
	QINT	REAL*16	REAL*16

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Nearest Integer[9,12] [a + .5*sign(a)]	1	NINT	ININT	REAL*4	INTEGER*2
			JNINT	REAL*4	INTEGER*4
			IIDNNT	REAL*8	INTEGER*2
			JIDNNT	REAL*8	INTEGER*4
			IIQNNT	REAL*16	INTEGER*2
			JIQNNT	REAL*16	INTEGER*4
		IDNINT	IIDNNT	REAL*8	INTEGER*2
			JIDNNT	REAL*8	INTEGER*4
		IQNINT	IIQNNT	REAL*16	INTEGER*2
			JIQNNT	REAL*16	INTEGER*4
		ANINT	ANINT	REAL*4	REAL*4
			DNINT	REAL*8	REAL*8
			QNINT	REAL*16	REAL*16

Zero-Extend Functions	1	ZEXT	IZEXT	LOGICAL*1	INTEGER*2
				LOGICAL*2	
				INTEGER*2	
			JZEXT	LOGICAL*1	INTEGER*4
				LOGICAL*2	
				LOGICAL*4	
				INTEGER*2	
				INTEGER*4	
Conversion to[10] REAL*4	1	REAL	FLOATI	INTEGER*2	REAL*4
			FLOATJ	INTEGER*4	REAL*4
			—	REAL*4	REAL*4
			SNGL	REAL*8	REAL*4
			SNGLQ	REAL*16	REAL*4
			—	COMPLEX*8	REAL*4
			—	COMPLEX*16	REAL*4

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Conversion to[10] REAL*8	1	DBLE	—	INTEGER*2	REAL*8
			—	INTEGER*4	REAL*8
			DBLE	REAL*4	REAL*8
			—	REAL*8	REAL*8
			DBLEQ	REAL*16	REAL*8
			—	COMPLEX*8	REAL*8
			—	COMPLEX*16	REAL*8
Conversion to REAL*16	1	QEXT	—	INTEGER*2	REAL*16
			—	INTEGER*4	REAL*16
			QEXT	REAL*4	REAL*16
			QEXTD	REAL*8	REAL*16

			—	REAL*16	REAL*16
			—	COMPLEX*8	REAL*16
			—	COMPLEX*16	REAL*16
Fix[10,12]	1	IFIX	IIFIX	REAL*4	INTEGER*2
(REAL*4-to-integer conversion)			JIFIX	REAL*4	INTEGER*4
Float[10]	1	FLOAT	FLOATI	INTEGER*2	REAL*4
(Integer-to-REAL*4 conversion)			FLOATJ	INTEGER*4	REAL*4
REAL*8 Float[10]	1	DFLOAT	DFLOTI	INTEGER*2	REAL*8
(Integer-to-REAL*8 conversion)			DFLOTJ	INTEGER*4	REAL*8
REAL*16 Float	1	QFLOAT	—	INTEGER*2	REAL*16
(Integer-to-REAL*16 conversion)				INTEGER*4	REAL*16
Conversion to COMPLEX*8,	1,2[13]	CMPLX	—	INTEGER*2	COMPLEX*8
or	1,2		—	INTEGER*4	COMPLEX*8

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
COMPLEX*8 from Two Arguments	1,2		—	REAL*4	COMPLEX*8
	1,2		—	REAL*8	COMPLEX*8
	1,2		—	REAL*16	COMPLEX*8
	1		—	COMPLEX*8	COMPLEX*8
	1		—	COMPLEX*16	COMPLEX*8
Conversion to COMPLEX*16,	1,2[13]	DCMPLX	—	INTEGER*2	COMPLEX*16
or	1,2		—	INTEGER*4	COMPLEX*16
COMPLEX*16 from Two Arguments	1,2		—	REAL*4	COMPLEX*16
	1,2		—	REAL*8	COMPLEX*16
	1,2		—	REAL*16	COMPLEX*16
	1		—	COMPLEX*8	COMPLEX*16
	1		—	COMPLEX*16	COMPLEX*16

Real Part of Complex	1	—	REAL DREAL	COMPLEX*8 COMPLEX*16	REAL*4 REAL*8
Imaginary Part of Complex	1	—	AIMAG DIMAG	COMPLEX*8 COMPLEX*16	REAL*4 REAL*8
Complex from Two Arguments	(See Conversion to COMPLEX*8 and Conversion to COMPLEX*16)				
Complex Conjugate (if a = (X,Y) CONJG (a) = (X,–Y))	1	CONJG	CONJG DCONJG	COMPLEX*8 COMPLEX*16	COMPLEX*8 COMPLEX*16
REAL*8 product of REAL*4's $a_1 * a_2$	2	—	DPROD	REAL*4	REAL*8

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Maximum[12]	n	MAX	IMAX0	INTEGER*2	INTEGER*2
$\max(a_1,a_2,...a_n)$			JMAX0	INTEGER*4	INTEGER*4
			AMAX1	REAL*4	REAL*4
(returns the maximum			DMAX1	REAL*8	REAL*8
value from among the			QMAX1	REAL*16	REAL*16
argument list; there must		MAX0	IMAX0	INTEGER*2	INTEGER*2
be at least two arguments)			JMAX0	INTEGER*4	INTEGER*4
		MAX1	IMAX1	REAL*4	INTEGER*2
			JMAX1	REAL*4	INTEGER*4
		AMAX0	AIMAX0	INTEGER*2	REAL*4
			AJMAX0	INTEGER*4	REAL*4

Minimum[12] $\min(a_1,a_2,...a_n)$ (returns the minimum value among the argument list; there must be at least two arguments)	n	MIN	IMIN0	INTEGER*2	INTEGER*2
			JMIN0	INTEGER*4	INTEGER*4
			AMIN1	REAL*4	REAL*4
			DMIN1	REAL*8	REAL*8
			QMIN1	REAL*16	REAL*16
		MIN0	IMIN0	INTEGER*2	INTEGER*2
			JMIN0	INTEGER*4	INTEGER*4
		MIN1	IMIN1	REAL*4	INTEGER*2
			JMIN1	REAL*4	INTEGER*4
		AMIN0	AIMIN0	INTEGER*2	REAL*4
			AJMIN0	INTEGER*4	REAL*4

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Positive Difference $a_1-(\min(a_1,a_2))$ (returns the first argument minus the minimum of the two arguments)	2	DIM	IIDIM	INTEGER*2	INTEGER*2
			JIDIM	INTEGER*4	INTEGER*4
			DIM	REAL*4	REAL*4
			DDIM	REAL*8	REAL*8
			QDIM	REAL*16	REAL*16
		IDIM	IIDIM	INTEGER*2	INTEGER*2
			JIDIM	INTEGER*4	INTEGER*4
Remainder $a_1-a_2*[a_1/a_2]$ (returns the remainder when the first argument is divided by the second)	2	MOD	IMOD	INTEGER*2	INTEGER*2
			JMOD	INTEGER*4	INTEGER*4
			AMOD	REAL*4	REAL*4
			DMOD	REAL*8	REAL*8
			QMOD	REAL*16	REAL*16

Transfer of Sign $\|a_1\|$ Sign a_2	2	SIGN	IISIGN	INTEGER*2	INTEGER*2
			JISIGN	INTEGER*4	INTEGER*4
			SIGN	REAL*4	REAL*4
			DSIGN	REAL*8	REAL*8
			QSIGN	REAL*16	REAL*16
		ISIGN	IISIGN	INTEGER*2	INTEGER*2
			JISIGN	INTEGER*4	INTEGER*4
Bitwise AND (performs a logical AND on corresponding bits)	2	IAND	IIAND	INTEGER*2	INTEGER*2
			JIAND	INTEGER*4	INTEGER*4
Bitwise OR (performs an inclusive OR on corresponding bits)	2	IOR	IIOR	INTEGER*2	INTEGER*2
			JIOR	INTEGER*4	INTEGER*4

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Bitwise Exclusive OR (performs an exclusive OR on corresponding bits)	2	IEOR	IIEOR JIEOR	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bitwise Complement (complements each bit)	1	NOT	INOT JNOT	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bitwise Shift (a_1 logically shifted left a_2 bits)	2	ISHFT	IISHFT JISHFT	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bit Extraction (extracts bits a_2 through $a_2 + a_3 - 1$ from a_1); see also MVBITS system subroutine	3	IBITS	IIBITS JIBITS	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4

Bit Set (returns the value of a_1 with bit a_2 of a_1 set to 1)	2	IBSET	IIBSET JIBSET	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bit Test (returns .TRUE. if bit a_2 of argument a_1 equals 1)	2	BTEST	BITEST BJTEST	INTEGER*2 INTEGER*4	LOGICAL*2 LOGICAL*4
Bit Clear (returns the value of a_1 with bit a_2 of a_1 set to 0)	2	IBCLR	IIBCLR JIBCLR	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bitwise Circular Shift [14] (circularly shifts rightmost a_3 bits of argument a_1 by a_2 places)	3	ISHFTC	IISHFTC JISHFTC	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4

Generic and Intrinsic Functions (Cont.)

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Length[12] (returns length of the character expression)	1	—	LEN	CHARACTER	INTEGER*4
Index (C_1,C_2)[12] (returns the position of the substring c_2 in the character expression c_1)	2	—	INDEX	CHARACTER	INTEGER*4
Character[12] (returns a character that has the ASCII value specified by the argument)	1	—	CHAR	LOGICAL*1 INTEGER*2 INTEGER*4	CHARACTER

ASCII Value[11] (returns the ASCII value of the argument; the argument must be a character expression that has a length of 1)	1	—	ICHAR	CHARACTER	INTEGER*4
Character relationals (ASCII collating sequence)	2	—	LLT	CHARACTER	LOGICAL*4
	2	—	LLE	CHARACTER	LOGICAL*4
	2	—	LGT	CHARACTER	LOGICAL*4
	2	—	LGE	CHARACTER	LOGICAL*4

NOTES

1. The argument of SQRT, DSQRT, or QSQRT must be greater than or equal to zero. The result of CSQRT or CDSQRT is the principal value, with the real part greater than or equal to zero. When the real part is zero, the result is the principal value, with the imaginary part greater than or equal to zero.

2. The argument of ALOG, DLOG, QSQRT, ALOG10, DLOG10, QLOG10, ATAND, ATAN2D, ASIND, DASIND, ACOSD, DACOSD, or QACOSD must be greater than zero. The argument of CLOG or CDLOG must not be (0.,0.).

3. The argument of SIN, DSIN, QSIN, COS, DCOS, QCOS, TAN, DTAN, or QTAN must be in radians. The argument is treated modulo $2*\pi$. The argument of SIND, COSD, or TAND must be in degrees. The argument is treated modulo 360.

4. The absolute value of the argument of ASIN, DASIN, QASIN, ACOS, DACOS, QACOS, ASIND, DASIND, QASIND, ACOSD, DACOSD, or QACOSD must be less than or equal to 1.

5. The result of ASIN, DASIN, QASIN, ACOS, DACOS, QACOS, ATAN, DATAN, QATAN, ATAN2, DATAN2, or QATAN2 is in radians. The result of ASIND, DASIND, QASIND, ACOSD, DACOSD, QACOSD, ATAND, DATAND, QATAND, ATAN2D, DATAN2D, or QATAN2D is in degrees.

6. If the value of the first argument of ATAN2, DATAN2, or QATAN2 is positive, the result is positive. When the value of the first argument is zero, the result is zero if the second argument is positive and π if the second argument is negative. If the value of the first argument is negative, the result is negative. If the value of the second argument is zero, the absolute value of the result is $\pi/2$. Both arguments must not have the value zero. The range of the result for ATAN2, DATAN2, and QATAN2 is: $-\pi < \text{result} < \pi$.

7. If the value of the first argument of ATAN2D, DATAN2D, or QATAN2D is positive, the result is positive. When the value of the first argument is zero, the result will be zero if the second argument is positive and 180 degrees if the second argument is negative. If the value of the first argument is negative, the result is negative. If the value of the second argument is zero, the absolute value of the result is 90 degrees. Both arguments must not have the value zero. The range of the result for ATAN2, DTAN2D, QATAN2D is: -180 degrees < result ≤ 180 degrees.

8. The absolute value of a complex number, (X,Y), is the real value:

$$(X^2+Y^2)^{\frac{1}{2}}$$

9. [x] is defined as the largest integer whose magnitude does not exceed the magnitude of x and whose sign is the same as that of x. For example [5.7] equals 5. and [-5.7] equals -5.

10. Functions that cause conversion of one data type to another type provide the same effect as the implied conversion in assignment statements. The following functions return the value of the argument without conversion: the function REAL with a real argument, the function DBLE with a double precision argument, the function INT with an integer argument, and the function QEXT with a REAL*16 argument.

11. See *Programming in VAX FORTRAN* for additional information on character functions.

12. The functions INT, IDINT, IQINT, NINT, IDNINT, IQNINT, IFIX, MAX1, MINI, and ZEXT return INTEGER*4 values if the /I4 command qualifier is in effect, INTEGER*2 values if the /NOI4 qualifier is in effect.

13. When CMPLX and DCMPLX have only one argument, this argument is converted into the real part of a complex value, and zero is assigned to the imaginary part. (When there are two arguments (not complex), a complex value is produced by converting the first argument into the real part of the value and converting the second argument into the imaginary part.)

14. Bits in a1 beyond the value specified by a3 are unaffected.

VAX FORTRAN Run-Time Error Summary

Summary of FORTRAN Run-Time Errors

FORTRAN Condition Symbol	Error Number	Severity	Message Text
FOR$_NOTFORSPE	1	F	not a FORTRAN-specific error
FOR$_SYNERRNAM	17	F	syntax error in NAMELIST input
FOR$_TOOMANVAL	18	F	too many values for NAMELIST variable
FOR$_INVREFVAR	19	F	invalid reference to variable in NAMELIST input
FOR$_REWERR	20	F	REWIND error
FOR$_DUPFILSPE	21	F	duplicate file specifications
FOR$_INPRECTOO	22	F	input record too long
FOR$_BACERR	23	F	BACKSPACE error
FOR$_ENDDURREA	24	F	end-of-file during read
FOR$_RECNUMOUT	25	F	record number outside range

FOR$_OPEDEFREQ	26	F	OPEN or DEFINE FILE required
FOR$_TOOMANREC	27	F	too many records in I/O statement
FOR$_CLOERR	28	F	CLOSE error
FOR$_FILNOTFOU	29	F	file not found
FOR$_OPEFAI	30	F	open failure
FOR$_MIXFILACC	31	F	mixed file access modes
FOR$_INVLOGUNI	32	F	invalid logical unit number
FOR$_ENDFILERR	33	F	ENDFILE error
FOR$_UNIALROPE	34	F	unit already open
FOR$_SEGRECFOR	35	F	segmented record format error
FOR$_ATTACCNON	36	F	attempt to access non-existent record
FOR$_INCRECLEN	37	F	inconsistent record length
FOR$_ERRDURWRI	38	F	error during write
FOR$_ERRDURREA	39	F	error during read
FOR$_RECIO_OPE	40	F	recursive I/O operation
FOR$_INSVIRMEM	41	F	insufficient virtual memory

Summary of FORTRAN Run-Time Errors (Cont.)

FORTRAN Condition Symbol	Error Number	Severity	Message Text
FOR$__NO__SUCDEV	42	F	no such device
FOR$__FILNAMSPE	43	F	file name specification error
FOR$__INCRECTYP	44	F	inconsistent record type
FOR$__KEYVALERR	45	F	keyword value error in OPEN statement
FOR$__INCOPECLO	46	F	inconsistent OPEN/CLOSE parameters
FOR$__WRIREAFIL	47	F	write to READONLY file
FOR$__INVARGFOR	48	F	invalid argument to FORTRAN Run-Time Library
FOR$__INVKEYSPE	49	F	invalid key specification
FOR$__INCKEYCHG	50	F	inconsistent key change or duplicate key
FOR$__INCFILORG	51	F	inconsistent file organization
FOR$__SPERECLOC	52	F	specified record locked
FOR$__NO__CURREC	53	F	no current record

FOR$_REWRITERR	54	F	REWRITE error
FOR$_DELERR	55	F	DELETE error
FOR$_UNLERR	56	F	UNLOCK error
FOR$_FINERR	57	F	FIND error
FOR$_LISIO_SYN	59	F,C	list-directed I/O syntax error
FOR$_INFFORLOO	60	F	infinite format loop
FOR$_FORVARMIS	61	F,C	format/variable-type mismatch
FOR$_SYNERRFOR	62	F	syntax error in format
FOR$_OUTCONERR	63	E,C	output conversion error
FOR$_INPCONERR	64	F,C	input conversion error
FOR$_OUTSTAOVE	66	F	output statement overflows record
FOR$_INPSTAREQ	67	F	input statement requires too much data
FOR$_VFEVALERR	68	F,C	variable format expression value error
SS$_INTOVF	70	F,C	arithmetic trap, integer overflow
SS$_INTDIV	71	F,C	arithmetic trap, integer zero divide
SS$_FLTOVF	72	F,C	arithmetic trap, floating overflow

Summary of FORTRAN Run-Time Errors (Cont.)

FORTRAN Condition Symbol	Error Number	Severity	Message Text
SS$_FLTOVF_F	72	F,C	arithmetic fault, floating overflow
SS$_FLTDIV	73	F,C	arithmetic trap, zero divide
SS$_FLTDIV_F	73	F,C	arithmetic fault, zero divide
SS$_FLTUND	74	F,C	arithmetic trap, floating underflow
SS$_FLTUND_F	74	F,C	arithmetic fault, floating overflow
SS$_SUBRNG	77	F,C	subscript out of range
MTH$_WRONUMARG	80	F	wrong number of arguments
MTH$_INVARGMAT	81	F	invalid argument to math library
MTH$_UNDEXP	82	F,C	undefined exponentiation
MTH$_LOGZERNEG	83	F,C	logarithm of zero or negative value
MTH$_SQUROONEG	84	F,C	square root of negative value

MTH$_SIGLOSMAT	87	F,C	significance lost in math library
MTH$_FLOOVEMAT	88	F,C	floating overflow in math library
MTH$_FLOUNDMAT	89	F,C	floating underflow in math library
FOR$_ADJARRDIM	93	F,C	adjustable array dimension error

Character Sets

Standard FORTRAN Character Set

The character set specified by the FORTRAN 77 Standard consists of the uppercase letters A through Z, the digits 0 through 9, and the following special characters:

HT (tab)	+ (plus sign)
SP (space)	, (comma)
$ (dollar sign)	- (minus sign)
' (apostrophe)	. (period)
((left parenthesis)	/ (slash)
) (right parenthesis)	: (colon)
* (asterisk)	= (equal sign)

VAX FORTRAN Character Set

The VAX FORTRAN character set includes the entire FORTRAN 77 Standard set plus the lower case letters a through z (upper- and lowercase letters are equivalent) and the following special characters:

! (exclamation mark)	< (left angle bracket)
" (quotation mark)	> (right angle bracket)
% (percent sign)	_ (underscore)
& (ampersand)	

All printable characters (those with ASCII values 20 through 7D, inclusive) can appear in character constants, Hollerith constants, and comments.

ASCII Character Set

ASCII Character Set

		Column							
		0	1	2	3	4	5	6	7
	0	NUL	DLE	SP	0	@	P		p
	1	SOH	DC1	!	1	A	Q	a	q
	2	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
Row	7	BEL	ETB	´	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y

A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell	ETB	End of Transmission Block

ASCII Character Set (Cont.)

BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tab	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
SP	Space	DEL	Delete