0 0 3 0 5 1 0 4 4 3 5

LBL-7978
UC-32 c. 1

A COMPARISON OF BKY-FTN4 AND VAX-FORTRAN IV-PLUS

Christopher Horne and Barrie Pardoe

RECEIVED
LAWRENCE
BERKELEY LABORATORY

August 1978

SEP 12 1978

LIBRARY AND
DOCUMENTS SECTION

Prepared for the U. S. Department of Energy
under Contract W-7405-ENG-48

For Reference

Not to be taken from this room

LBL-7978
c. 1

LEGAL NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the Department of Energy, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

Printed in the United States of America

Available from

National Technical Information Service

U. S. Department of Commerce

5285 Port Royal Road

Springfield, VA 22161

Price: Printed Copy, \$ 5.25 Domestic; \$10.50 Foreign

Microfiche, \$ 3.00 Domestic; \$ 4.50 Foreign

0 0 4 0 5 1 0 4 4 3 6

LBL-7978

A COMPARISON OF BKY-FTN4 AND VAX-FORTRAN IV-PLUS

Christopher Horne
Summer Student
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

University of Arizona
Tucson, Arizona

Barrie Pardoe
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

August 1978

Work supported by the Department of Energy

CONTENTS

Section 1	Introduction	1
Section 2	Notation for Description of FTN4	3
Section 3	Description of FTN4	5
Section 4	Notation for Description of FORTRAN IV-Plus	12
Section 5	Description of FORTRAN IV-Plus	14
Section 6	Comparison of Basic FORTRAN Features	21
Section 7	Semantic Differences	26
Section 8	Library Routines	39
Section 9	Summary of VAX/VMS Operating System Commands	45
Section 10	Transferring Programs via Magnetic Tape	49

1.0 INTRODUCTION

This note serves as a guide for those who intend to transfer (port) FORTRAN IV programs between the CDC machines at LBL and the VAX. For those porting existing code, this guide should prove helpful by pointing out language constructs which will cause trouble. It should also aid in the development of new software intended to be run on either computer by alerting the programmer to incompatible language constructs, and presenting him with a "minimal language" which requires only minor alteration to run on either machine.

In particular, this note compares the FORTRAN IV dialect FTN4 as implemented on the BKY system at LBL, and FORTRAN IV-PLUS on the VAX. FTN4 and FORTRAN IV-PLUS are dialects of ANSI FORTRAN (X3.9-66), both of which offer features far in excess of the standard -- making FORTRAN a more pleasant language to program in. Although some of these extended features are compatible, others cause considerable problems in porting programs from one machine to the other. A trade-off exists: using incompatible language constructs reduces the initial programming time (and generally the execution time) but increases the time and effort required to port programs.

Selecting a language construct from one dialect and comparing it with its counterpart in the other dialect yields three possible results.

- o The construct is available in one dialect but not the other.
- o The construct is present in both dialects but has different interpretations.
- o The construct is available in both dialects with the same interpretation.

In the latter case the construct is said to be compatible and is thus a member of the minimal language.

Sections 3 and 5 describe, in the style of a foreign language dictionary, the dialects FTN4 and FORTRAN IV-Plus respectively, with sections 2 and 4 serving as an introduction to notation. The non-shaded areas in these sections describe the minimal language; sections 2 and 3 are for those familiar with FTN4 and sections 4 and 5 are for the FORTRAN IV-Plus user. In section 3 and 5 FORTRAN statements are categorized. Examples and references are given. The reference number is the page in the appropriate FORTRAN dialect reference manual:

- o BKY-FORTRAN Extended Version 4 Reference manual
(LBL-2 11/76).

- o VAX-11 FORTRAN IV-Plus Language Reference Manual
(Base Level 5 Release Note).

Section 6 is a table comparing some basic features of the languages. Section 7 concentrates on more subtle semantic differences between the dialects while again pointing out some syntactic differences. Section 8 is a comparison of the libraries provided by the two dialects. A summary of the VAX-VMS operating system control language is presented in section 9. Section 10 explains a procedure for transferring programs via magnetic tape between the BKY system and the VAX.

The numbering system used in this note allows quick access to related information. For example, section 3.2.3 describes the syntax of FTN4's DO loop, section 5.2.3 describes the syntax of FORTRAN IV-Plus's DO loop, and section 7.2.3 contains a discussion of the different interpretations of DO loop constructs within the two dialects.

The information presented in this note was extracted from the manuals cited above with short test programs being constructed for clarification. The papers mentioned in the bibliography provided direction by indicating where to look in the dialects for trouble spots.

2.0 Description of FTN4

2.1 Notation

A Meta-symbol identifies a syntatic category without having to enumerate all possible members of that category. The following Meta-symbols are used in the description of FTN4 statements.

<u>Symbol</u>	<u>Interpretation</u>
v	a variable, array element, or array name.
vn	a variable or array name.
iv	an integer variable.
lv	a logical variable.
ipvc	a positive integer variable or constant
exp	an expression. NOTE -- a constant, variable, array name, function reference or an array element constitute the trivial cases of of an expression.
expi	an expression which evaluates to an integer. If it does not evaluate to an integer, its value is truncated and converted to an integer.
ariexp	arithmetic expression. Operators +, -, *, /, **.
mskexp	masking expression. Operators .AND., .OR., .NOT. (abbreviated .A., .O., .N.).
relexp	relational expression. Operators .EQ., .NE., .GT., .GE., .LT., and .LE. .
logexp	logical expression. Operators .AND., .OR., .NOT., .TRUE., and .FALSE. (abbreviated .A., .O., .N., .T., and .F.) NOTE -- a logical expression differs from a masking expression in that its operands must be declared logical.
m	an integer variable, or an integer or octal constant.
name	a symbolic name of length 7 or less, beginning with a letter, containing only alphanumeric characters.

sn	a statement label.
sne	the label of an executable statement. This excludes FORMAT statement labels.
estat	any executable statement except a DO loop header or a logical IF statement.
fn	statement label of a format statement.
type	INTEGER, REAL, LOGICAL, COMPLEX, DOUBLE PRECISION , DOUBLE .
u	input/output unit: 1 or 2 digit integer constant, integer variable with value in range [1-99], or Hollerith value which is the display code filename left justified zero filled.
list	A series or variable names, array names, array elements, or implied do lists, separated by commas.

Upper case words, as well as punctuation marks, are written as shown. Meta-symbols are written in lower case. An ellipsis (...) indicates that the preceding item may be repeated.

Reference numbers given in the right margin are to the
BKY-FORTRAN Extended Version 4 Reference Manual (LBL-2 11/76).

3.1 ASSIGNMENT STATEMENTS

v = ariexp	i-4-1
lv = logexp or relexp	i-4-1
v = mskexp	i-4-1
ASSIGN sne TO iv	i-5-4

3.2 multiple assignments

v = v = ... v = exp	i-4-6
v = v \$ v = v ... \$ v = v	i-1-2

3.2 CONTROL STATEMENTS3.2.1 GOTO statements

GOTO sne	i-5-2
GOTO (sne , ... , sne) , expi	i-5-2
GOTO (sne , ... , sne) expi	i-5-2
GOTO iv , (sne , ... , sne)	i-5-5
GOTO iv (sne , ... , sne)	i-5-5

3.2.2 IF statements

IF (ariexp or mskexp) sne ,sne ,sne	i-5-6
IF (ariexp, mskexp, logexp, or relexp) sne ,sne	i-5-7 .
IF (logexp or relexp) estat	i-5-7

3.2.3 DO loops

DO sne iv = ipvc , ipvc , ipvc	i-5-8
DO sne iv = ipvc , ipvc	i-5-8

3.2.4 CONTINUE statements

CONTINUE	i-5-14
----------	--------

3.2.5 Subroutine CALL statements

CALL name	i-7-14
CALL name (exp , ... , exp)	i-7-14

CALL name , RETURNS (b , ... , b)	i-7-14
-------------------------------------	--------

CALL name (exp , ... , exp), RETURNS (b,...b)	i-7-14
--	--------

3.2.6 RETURN statements

RETURN	i-5-15
--------	--------

RETURN i { through RETURNS list}	i-5-16
----------------------------------	--------

3.2.7 PAUSE statements

PAUSE	i-5-14
-------	--------

PAUSE i	
---------	--

PAUSE *text*	i-5-14
--------------	--------

3.2.8 STOP statements

STOP	i-5-15
------	--------

STOP i	i-5-15
--------	--------

STOP *text*	i-5-15
-------------	--------

3.3 TYPE DECLARATIONS

INTEGER name , ... , name	i-6-2
---------------------------	-------

TYPE INTEGER name , ... , name	i-6-2
--------------------------------	-------

REAL name , ... , name	i-6-2
------------------------	-------

TYPE REAL name , ... , name	i-6-2
-----------------------------	-------

COMPLEX name , ... , name	i-6-2
---------------------------	-------

TYPE COMPLEX name , ... , name	i-6-2
--------------------------------	-------

DOUBLE PRECISION name , ... , name	i-6-3
------------------------------------	-------

DOUBLE name , ... , name	i-6-3
--------------------------	-------

TYPE DOUBLE PRECISION name , ... , name	i-6-3
---	-------

TYPE DOUBLE name , ... , name	i-6-3
-------------------------------	-------

LOGICAL name , ... , name	i-6-3
---------------------------	-------

TYPE LOGICAL name , ... , name	i-6-3
--------------------------------	-------

IMPLICIT type (ac) , ... , type (ac)	i-6-3
--------------------------------------	-------

ac is a list of single alphabetic characters
or a (range which is represented by the
first and last characters separated by a
minus sign) .

3.4 EXTERNAL DECLARATIONS

EXTERNAL name , ... , name i-6-16

3.5 STORAGE ALLOCATION

3.5.1 ARRAY storage declaration

DIMENSION name (d) , ... , name (d) i-6-5

type name (d) , ... , name (d) i-6-1

TYPE type name (d) , ... , name(d) i-6-1

d array declarator. 1-3 integer constants
separated by commas; or if name is a
dummy argument in a subprogram, 1-3
integer variables or integer constants.

3.5.2 COMMON statements

COMMON vn , ... , vn i-6-8

COMMON /bname/ vn ,...,vn /bname/ vn ,..., vn i-6-8

COMMON // vn , ... , vn i-6-8

bname symbolic name or a 1-7 digit number
// blank common

3.5.3 DATA statements

DATA vlist /dlist / , ... , vlist /dlist / i-6-10

DATA (vlist = dlist) ,..., (vlist = dlist) i-6-19

vlist list of array names, array elements,
variables, or implied DO lists, separated
by commas.

dlist one or more of the following forms
separated by commas :

constant
(constant list)
f*constant
f*(constant list)
f(constant list)

where f is a repetition factor.

3.5.4 EQUIVALANCE statements

EQUIVALANCE (v , ... , v) , ... , (v , v) i-6-10

3.5.5 LEVEL statement

LEVEL n , v , ... , v i-6-15

where n = 1, 2, or 3 .

3.6 PROGRAM MODULES

3.6.1 MAIN programs

PROGRAM name (file , ... , file) i-7-1

PROGRAM name i-7-1

3.6.2 FUNCTION subprograms

FUNCTION name (vn , ... , vn) i-7-6

type FUNCTION name (vn , ... , vn) i-7-6

3.6.3 SUBROUTINE subprograms

SUBROUTINE name (vn , ... , vn) i-7-12

SUBROUTINE name i-7-12

SUBROUTINE name (vn , ..., vn), RETURNS(b , ..., b) i-7-12

SUBROUTINE name, RETURNS(b , ..., b) i-7-12

3.6.4 BLOCK DATA subprograms

BLOCK DATA i-6-25

BLOCK DATA name i-6-25

3.6.5 ENTRY point

ENTRY name i-7-20

3.7 STATEMENT FUNCTIONS

name(vn , ... , vn) = exp i-7-9

3.8 INPUT/OUTPUT STATEMENTS

3.8.1 PRINT statements

i-9-3

```
PRINT fn, list
```

```
PRINT fn
```

```
PRINT ( u, fn) list
```

```
PRINT ( u, fn)
```

```
PRINT * , list
```

```
PRINT ( u, * ) list
```

3.8.2 PUNCH statements

i-9-4

```
PUNCH fn, list
```

```
PUNCH fn
```

```
PUNCH ( u, fn) list
```

```
PUNCH ( u, fn)
```

```
PUNCH * , list
```

```
PUNCH ( u, * ) list
```

3.8.3 WRITE statements

i-9-5

```
WRITE ( u, fn) list
```

```
WRITE ( u, fn)
```

```
WRITE fn, list
```

```
WRITE fn
```

```
WRITE (u) list
```

```
WRITE (u)
```

```
WRITE ( u, * ) list
```

```
WRITE * , list
```

3.8.4 READ statements

i-9-7

```
READ ( u, fn)
```

```
READ ( u, fn) list
```

```

READ  fn, list
READ  (u) list
READ  (u)
READ  ( u, * ) list
READ  * , list

```

3.8.5 BUFFERING

```

BUFFER IN ( u, p) ( a, b)           i-9-13
BUFFER OUT ( u, p) ( a, b)         i-9-15

```

3.8.6 NAMELIST statements

```

NAMELIST / name / v , ... , v      i-9-16
READ  ( u, name )                  i-9-17
WRITE ( u, name )

```

3.8.7 ENCODE/DECODE statements

```

ENCODE ( m, fn, v) list            i-9-22
DECODE ( m, fn, v) list            i-9-23

```

3.8.8 FILE MANIPULATION

```

REWIND u                            i-9-12
BACKSPACE u                          i-9-12
ENDFILE u                           i-9-12

```

3.9 FORMAT STATEMENTS

```

sn    FORMAT ( fs , ... , fs )

```

Field	Translation
Iw	Integer
Iw.z	Integer, minimum digits
Ow	Octal
Ow.z	Octal, minimum digits

0 0 4 0 5 1 0 4 4 4 3

BKY-FTN4

11

BKY-FTN4

Zw	Hexadecimal
Fw.d	Single precision, no exponer
Ew.d	Single precision, Exponent
Ew.dEz	Single precision, exponent
Ew.dDz	Single precision, exponent
Dw.d	Double precision, with expor
Gw.d	Single precision general
Lw	Logical
Aw	Alphanumeric (left justif.)
Rw	Alphanumeric (right justif.)
nH	Hollerith
nX	Space over n columns
Tn	Tab over to column n
nP	Scale factor (for F,G,D,E)
V	Variable (right justified)
=	Variable (integer)
/	Separator, skip a line
,	Separator
* ... *	Character string
# ... #	Character string

4.0 Description of FORTRAN IV-Plus

4.1 Notation

A Meta-symbol identifies a syntactic category without having to enumerate all possible members of that category. The following Meta-symbols are used in the description of FORTRAN IV-Plus statements.

<u>Symbol</u>	<u>Interpretation</u>
v	a variable, array element, or array name.
vn	a variable or array name.
iv	an integer variable.
lv	a logical variable.
cv	character variable
exp	an expression. NOTE -- a constant, variable, array name, function reference or an array element constitute the trivial cases of an expression.
expi	an expression which evaluates to an integer. If it does not evaluate to an integer, its value is truncated and converted to an integer.
ariexp	arithmetic expression. Operators +, -, *, /, **.
relexp	relational expression. Operators .EQ., .NE., .GT., .GE., .LT., and .LE. .
logexp	logical expression. Operators .AND., .OR., .NOT., .EQV., .XOR., .TRUE., and .FALSE.
chrexp	a character expression. Operators //, and substring reference (::#).
m	an integer variable, or an integer or octal constant.
name	a symbolic name of length 15 or less, beginning with a letter, containing only alphanumeric characters, \$, and _ (underscore).
block	a self contained group of executable statements.
sn	a statement label.

sne	the label of an executable statement. This excludes FORMAT statement labels.
estat	any executable statement except a DO loop header or a logical IF statement.
fn	statement label of a format statement.
type	INTEGER, REAL, COMPLEX, DOUBLE PRECISION, LOGICAL, INTEGER*2, INTEGER*4, REAL*4, REAL*8, COMPLEX*8, LOGICAL*1, LOGICAL*2, LOGICAL*4, BYTE or, CHARACTER * m ; where m is the number of characters .
u	input/output unit: 1 or 2 digit integer constant, integer variable with value in range [1-99].
list	A series or variable names, array names, array elements, or implied do lists, separated by commas.

Upper case words, as well as punctuation marks, are written as shown. Meta-symbols are written in lower case. An ellipsis (...) indicates that the preceding item may be repeated.

Reference numbers in the right margin refer to the VAX-11 FORTRAN IV-Plus Language Reference Manual, Base level 5 release note.

5.1 ASSIGNMENT STATEMENTS

v = ariexp

lv = logexp or relexp

cv = chrexp

ASSIGN sne TO iv

PARAMETER v = c , ... , v = c

5.2 CONTROL STATEMENTS5.2.1 GOTO statements

GOTO sne 4-1-1

GOTO (sne , ... , sne) , expi 4-1-1

GOTO (sne , ... , sne) expi 4-1-2

GOTO iv , (sne , ... , sne) 4-1-3

GOTO iv (sne , ... , sne) 4-1-3

GOTO iv 4-1-3

5.2.2 IF statements

IF (ariexp or mskepx) sne , sne , sne 4-2-1

IF (logexp or relexp) estat 4-2-2

```

IF ( logexp or relexp ) THEN 4-2-3
    !
    block
    !
    ELSE IF ( logexp or relexp ) THEN
        !
        block
        !
        ELSE
            !
            block
            !
END IF

```

5.2.3 DO loops

DO sne v = ariexp , ariexp , ariexp 4-3

DO sne v = ariexp , ariexp 4-3

```

DO sne , v = ariexp , ariexp , ariexp      4-3
DO sne , v = ariexp , ariexp                4-3

```

5.2.4 CONTINUE statements

```
CONTINUE                                     4-4
```

5.2.5 Subroutine CALL statements

```
CALL name                                   4-5
```

```
CALL name ( exp , ... , exp )              4-5
```

5.2.6 RETURN statements

```
RETURN                                     4-6
```

5.2.7 PAUSE statements

```
PAUSE                                       4-7
```

```
PAUSE i
```

```
PAUSE 'text'                               4-7
```

5.2.8 STOP statements

```
STOP                                       4-8
```

```
STOP i                                     4-8
```

```
STOP 'text'                               4-8
```

5.3 TYPE DECLARATIONS

```
INTEGER name , ... , name                 8-2
```

```
INTEGER * 2 name , ... , name              8-2
```

```
INTEGER * 4 name , ... , name              8-2
```

```
REAL name , ... , name                     8-2
```

```
REAL * 4 name , ... , name                 8-2
```

```
REAL* 8 name , ... , name                  8-2
```

```
COMPLEX name , ... , name                  8-2
```

```
COMPLEX * 8 name , ... , name              8-2
```

```
DOUBLE PRECISION name , ... , name         8-2
```

LOGICAL name , ... , name	8-2
LOGICAL * 1 name , ... , name	8-2
LOGICAL * 2 name , ... , name	8-2
LOGICAL * 4 name , ... , name	8-2
CHARACTER name * m, ... , name * m	8-2
CHARACTER * m name , ... , name	8-2
BYTE name , ... , name	8-2
IMPLICIT type (ac) , ... , type (ac)	8.1

ac is a list of single alphabetic characters
or a range (which is represented by the
first and last characters separated by a
minus sign)

5.4 EXTERNAL DECLARATIONS

EXTERNAL name , ... , name	8-6
EXTERNAL * name , ... , * name	8-6

5.5 STORAGE ALLOCATION

5.5.1 ARRAY storage declaration

DIMENSION name (d) , ... , name (d)	8-3
VIRTUAL name (d) , ... , name (d)	8-3
type name (d) , ... , name (d)	8-2

d array declarator. 1-7 integer constants
separated by commas; or if name is a
dummy argument in a subprogram, 1-7
integer variables or integer constants.

5.5.2 COMMON statements

COMMON vn , ... , vn	8-4
COMMON /name/ vn ,... ,vn /name/ vn ,... , vn	8-4
COMMON /name/ vn ,... ,vn , /name/ vn ,... , vn	8-4
COMMON // vn , ... , vn	8-4

// blank common

5.5.3 DATA statements

DATA vlist /dlist / , ... , vlist /dlist / 8-7

DATA vlist /dlist/ ... vlist /dlist/ 8-7

vlist list of array names, array elements,
or variables, separated by commas.
dlist one or more of the following forms
separated by commas :

constant
f*constant

where f is a repetition factor.

5.5.4 EQUIVALENCE statements

EQUIVALENCE (v , ... , v) , ... , (v , v) 8-5

5.6 PROGRAM MODULES

5.6.1 MAIN programs

PROGRAM name 8-9

5.6.2 FUNCTION subprograms

FUNCTION name (vn , ... , vn) 9-2-2

FUNCTION name 9-2-2

type FUNCTION name (vn , ... , vn) 9-2-2

type FUNCTION name 9-2-2

5.6.3 SUBROUTINE subprograms

SUBROUTINE name (vn , ... , vn) 9-2-3

SUBROUTINE name 9-2-3

5.6.4 BLOCK DATA subprograms

BLOCK DATA 9-2-5

BLOCK DATA name 9-2-5

5.6.5 ENTRY point

ENTRY name	9-2-4
------------	-------

ENTRY name (vn , ... , vn)	9-2-4
------------------------------	-------

5.7 STATEMENT FUNCTIONS

name(vn , ... , vn) = exp	9-2-1
----------------------------	-------

5.8 INPUT/OUTPUT STATEMENTS

5.8.1 PRINT statements

PRINT fn, list	5-4-2
----------------	-------

PRINT fn	5-4-2
----------	-------

PRINT *, list	5-4-2
---------------	-------

5.8.2 PUNCH statements not available on VAX.

5.8.3 WRITE statements

TYPE fn , list	5-4-2
----------------	-------

TYPE * , list	5-5-2
---------------	-------

WRITE (u, fn) list	5-4-2
---------------------	-------

WRITE (u, fn)	5-4-2
----------------	-------

WRITE (u) list	5-6-2
----------------	-------

WRITE (u)	5-6-2
-----------	-------

WRITE (u, *) list	5-5-2
---------------------	-------

WRITE (u ' expi , fn) list	5-7-2
------------------------------	-------

5.8.4 READ statements

ACCEPT fn , list	5-4-1
------------------	-------

ACCEPT * , list	5-5-1
-----------------	-------

READ (u, fn)	5-4-1
---------------	-------

READ (u, fn) list	5-4-1
--------------------	-------

READ fn, list	5-4-1
---------------	-------

READ (u) list	5-6-1
---------------	-------

READ (u)	5-6-1
READ (u, *) list	5-5-1
READ * , list	5-5-1
READ (u ' expi , fn) list	5-8-1

5.8.5 BUFFERING --- feature not available.

5.8.6 NAMELIST statements --- not available

5.8.7 ENCODE/DECODE statements

ENCODE (m, fn, v) list	5-9
DECODE (m, fn, v) list	5-9

5.8.8 FILE MANIPULATION

REWIND u	6-3
BACKSPACE u	6-4
ENDFILE u	6-6
OPEN (unit = iv ,)	6-1
CLOSE (unit = iv ,)	6-2
DEFINE FILE	6-7
FIND (u ' expi)	

5.9 FORMAT STATEMENTS

sn FORMAT (fs , ... , fs)

Iw	Integer
Ow	Octal
Zw	Hexadecimal
Fw.d	Single precision floating point
Ew.d	Single precision floating point with exponent
Dw.d	Double precision with exponent
Gw.d	General floating point

Lw	Logical
Aw	Alphanumeric
nH	Hollerith
nX	Space over n collumns
Tn	Tab over to collumn n
nP	Scale factor
/	Separator, skip line
,	Separator
' ... '	Character string
Q	Number left in iolist
\$	No carrage return after print
;	Stop if no more to read
< exp >	Variable

6.0 BASIC FEATURES

The following table compares some of the basic characteristics of FTN4 and VAX FORTRAN 4-Plus.

CATEGORY	FTN4	FORTRAN IV-Plus	Compatible
CHARACTER SETS			
alphabetic numeric special collating seq	A-Z 0-9 blank,=+~*/ ().\$# display code	A-Z,a-z 0-9 blank,=+~*/ ().'"\$_:<>% ASCII	A_Z 0-9 blank,=+~*/ ().\$ ---
STATEMENT FIELDS			
Comments delimiters Labels columns characters leading zeros	C,*, \$ in col 1 1-5 numeric ignored	C or c in col 1 ! anywhere 1-5 numeric ignored	C in col 1 1-5 numeric ignored
Continuations column any character but max number comments in between blank cards in between Statement Field columns Max Length of Input Multiple Statements per line	6 0,blank 19 allowed not allowed 7-72 80 allowed	6 0,blank variable allowed not allowed 7-72 88 not allowed	6 0,blank 19 allowed not allowed 7-72 80 not allowed
VARIABLE NAMES			
leading character remaining chars length	alphabetic alphanumeric 7	alphabetic alphanumeric,\$ no limit, unique to 15	alphabetic alphanumeric 7

CATEGORY	FTN4	FORTTRAN IV-Plus	Compatible
ARRAYS			
Subscripts max number of lower bounds negative indexing	3 not allowed not allowed	7 allowed allowed	3 not allowed not allowed
Elements omission of subscripts	allowed	not allowed except in EQUIVALENCE	not allowed
Max array size	2**17-1	no realistic limit	2**17-1
Storage sequence	first fastest (by columns)	first fastest (by columns)	first fastest (by columns)
CONSTANTS			
Integer range representation +.- *,/	-2 ⁵⁹ -1:2 ⁵⁹ -1 60 bit 1's comp 48 bit 1's comp	-2 ³¹ :2 ³¹ -1 32 bit 2's comp 32 bit 2's comp	--- --- ---
Floating Point range representation single precision mantisa exponent # decimal digit sign bit double precision mantisa exponent # decimal digit	10 ⁻²⁹³ :10 ³²² 48 bit 1's comp 11 bit excess 2000(8) 14 bit 59 96 bit 1's comp 11 bits 29	10 ⁻³⁷ :10 ³⁸ hidden normal 24 bit 2's comp 8 bit excess 200(8) 7 bit 31 56 bit 2's comp 8 bits 16	--- ----
Complex	same as F.P.	same as F.P.	
Hollerith forms:justify/fill	nH ...:left/∅ nL ...:left/0 nR ...:right/0 #...#:left/∅	nH ...:left/∅ --- Radix-50 '...':left/∅	nH ...:left/∅ --- --- ---
char/word bits/char	10 6	4 8	4 ---

CATEGORY	FTN4	FORTRAN IV-Plus	Compatible
CONSTANTS(cont.)			
Logical forms	.TRUE.,.FALSE., .T.,.F.	.TRUE.,.FALSE.	.TRUE.,.FALSE.
representaion			
true	-1 (777...776)	-1 (377...777)	---
false	+0 (000...000)	0 (000...000)	---
test			
true	bit 59 = 1	bit 0 = 1	
false	bit 59 = 0	bit 0 = 0	
Character data type	no	yes	no
Octal	11...11B	'11...11'0 "11...11	---
Hex	not allowed	'11...11'X	---
MISC.			
Max embedded DO	50	20	20
Max arguments to subroutine	63	60	60
Max number of COMMON blocks	125	no realistic limit	125
Max literal length	70	255	70
Max FORMAT smatement paren depth	20	8	8
Max number of files open	50	no realistic limit	50

CHARACTER CODES

character	A	D	char	A	D	char	A	D	char	A	D
NUL	0		SP	40	55	@	100	74	`	140	
SHO	1		!	41	66	A	101	1	a	141	
STX	2		"	42	64	B	102	2	b	142	
ETX	3		#	43	60	C	103	3	c	143	
EOT	4		\$	44	53	D	104	4	d	144	
ENQ	5		%	45	63	E	105	5	e	145	
ACK	6		&	46	70	F	106	6	f	146	
BEL	7		'	47	70	G	107	7	g	147	
BS	10		(	50	51	H	110	10	h	150	
HT	11		)	51	52	I	111	11	i	151	
LF	12		*	52	47	J	112	12	j	152	
VT	13		+	53	45	K	113	13	k	153	
FF	14		,	54	56	L	114	14	l	154	
CR	15		-	55	46	M	115	15	m	155	
SO	16		.	56	57	N	116	16	n	156	
SI	17		/	57	50	O	117	17	o	157	
DLE	20		0	60	33	P	120	20	p	160	
DC1	21		1	61	34	Q	121	21	q	161	
DC2	22		2	62	35	R	122	22	r	162	
DC3	23		3	63	36	S	123	23	s	163	
DC4	24		4	64	37	T	124	24	t	164	
NAK	25		5	65	40	U	125	25	u	165	
SYN	26		6	66	41	V	126	26	v	166	
ETB	27		7	67	42	W	127	27	w	167	
CAN	30		8	70	43	X	130	30	x	170	
EOM	31		9	71	44	Y	131	31	y	171	
SUB	32		:	72	0	Z	132	32	z	172	
ESC	33		;	73	77	[	133	61	{	173	
FS	34		<	74	72	\	134	75		174	
GS	35		=	75	54	]	135	62	}	175	
RS	36		>	76	73	^	136	76	~	176	
US	37		?	77	71	_	137	65	DEL	177	

A=ASCII(octal)

D=DISPLAY CODE (octal)

Internal representation of numbers

{ S=sign bit, exp=exponent, frac n=n'th part of fraction }

CDC 60-bit floating point

VAX 32-bit floating point { REAL*4 }

VAX Double precision floating point { REAL*8 }

7.0 Section VII:

This section describes differences between the two dialects FORTRAN IV-Plus and FTN4 not captured in previous sections. The order in which material is presented in this section parallels that of section 3 and 5. While those sections emphasized syntactic differences, this focuses on differences in interpretation and semantic differences.

For portable programs, the following order of statements should be observed. Horizontal lines separate statement classes which may not be intermixed. Vertical sections may occur anywhere indicated.

!	C	!	PROGRAM, FUNCTION, SUBROUTINE, or BLOCK DATA	!
!	O	!		!
!	M	!	IMPLICIT	!
!	M	!		!
!	E	!	F ! type, DIMENSION, COMMON, EQUIVALENCE,	!
!	N	!	EXTERNAL	!
!	T	!	R	!
!	S	!	M ! D ! statement function definitions	!
!		!	A ! A !	!
!		!	T ! T ! ENTRY	!
!		!	A ! and executable statements	!
!		!		!
!		!	END statement	!
!		!		!

7.1 Assignment statements :

Assignment statements containing arithmetic, logical, or masking expressions should port with little difficulty. There are, however, some differences in interpretation. Ignoring differences caused by word length and rounding errors, the following points should be considered.

- o Exponentiation: $A**B**C$ is interpreted by FORTRAN IV-Plus as $A**(B**C)$, and as $(A**B)**C$ by FTN4. Also, the permissible types of base and exponents are more restrictive on the VAX as indicated by the following table.

EXPONENT				
BASE	integer	real	double precision	complex
integer	BV	B	B	B
real	BV	BV+	BV+	B
double	BV	BV+	BV+	B
complex	BV			B

B=OK for FTN4, V=OK for FORTRAN IV-Plus,
V+=OK for VAX if base is positive.

- o Specification of Constants: The syntax for specifying octal constants is incompatible. See the table in section 6 for their syntatic representation. Also, hexadecimal constants are not acceptable to FTN4.
- o Evaluation of Logical Expressions: FORTRAN IV-Plus will evaluate logical expressions until the result is known. This does not always require complete evaluation: Consider $X=A.OR.B(I)$, where A, B, and X are of type logical. If A is true, independent of the value of B(I), X will be true. FORTRAN IV-Plus will, in cases like this, not evaluate B(I). If B(I) is a logical function call, the function call will not be made and any side-effects (alteration of global or common variables) caused by function B will not be produced. FTN4 evaluates all elements of an expression to determine the result. ~~NOTE -- FORTRAN IV-Plus's~~ evaluation is always complete when operands are non-logical, so masking expressions may be used with confidence.
- o Masking .vs. Logical Expressions: Although logical and masking expressions use the same operators, FTN4 will not allow a mixture of operand types. Thus, a masking expression looks the same as a logical expression except that none of the operands may be of type logical. FORTRAN IV-Plus does not provide masking expressions as such; logical operators and operands may be interspersed with other arithmetic operators and operands.
- o Logical testing: Although FTN4 tries to prohibit assignment of values other than true and false to a logical variable, such assignments can occur. For example a function call, declared logical in the calling routine, may return some other type of result. As an example, if (via a function call) the integer 3 is assigned to a logical variable and that variable is subsequently used in a logical test (such as a logical IF), FORTRAN IV-Plus will perceive the value as being true (it tests the low order bit), while FTN4 will perceive the value as being false (it tests

the high order, sign bit).

		DEFINED		TEST	
FTN4	!	true	-1 (777...776)	!	negative value
	!	false	0 (000...000)	!	positive value
FOR4+	!	true	-1 (777...777)	!	1 in bit zero
	!	false	0 (000...000)	!	0 in bit zero

In summary, avoid assigning values other than TRUE and FALSE to logical variables.

- o Array references: The abbreviated form of a multi-dimensional array reference allowed by FTN4 (e.g. if A is a three dimensional array then referencing A(2) is the same as referencing A(2,1,1)) is not permitted by FORTRAN IV-Plus (except in EQUIVALENCE statements).

Type conversions for mixed mode expressions differ only in the exceptions noted above concerning logical variables. The use of the PARAMETER statement, as well as CHARACTER expressions and assignment statements should be avoided -- they are unavailable with FTN4, and thus will not port. Multiple assignment statements (A=B \$ B=C) and cascading assignment statements (A=B=C), permitted by FTN4, should not be used .

7.2 Control statements :

Control statements interrupt the normal sequential flow of a program and, conditionally, transfer control to some other portion of the program.

- 7.2.1 GOTO statements: The error handling of assigned and computed GOTO's differs between FORTRAN IV-Plus and FTN4. For computed GOTO's, when the index value into the statement label list is out of range for the list given, FORTRAN IV-Plus will branch to the first label, while FTN4 will give a fatal run time error. For assigned GOTO's, if the label indicated by the control variable does not correspond to one of the elements in the list of statement labels, FTN4 will produce unpredictable results, FORTRAN IV-Plus will branch to the label anyway, if it exists. With FORTRAN IV-Plus, the list of statement labels for the computed GOTO is optional. Rules:

1. Do not use assigned GOTO's without a statement label list.
2. With both computed and assigned GOTO's, insure a 1-1 correspondence between the label indicated by the control variable

and the statement label list.

- 7.2.2 IF statements: Both FTN4 and FORTRAN IV-Plus provide one form of an IF statement which the other lacks: FTN4 has the two branch IF, and FORTRAN IV-Plus provides a block IF-THEN-ELSE-ENDIF construct. Neither of these should be used if code is to be ported.

The cautions concerning the evaluation of logical exssions , given in section 7.1 ,apply to logical IF statements.

- 7.2.3 DO loops: FORTRAN IV-Plus offers a more flexible form of the DO loop than does FTN4. FTN4 requires that the parameters (initial, final, and increment) of a DO loop be simple variables or constants and, furthermore, must all be positive. FORTRAN IV-Plus has none of these restrictions -- all parameters may be arbitrarily complicated arithmetic expressions and have any sign. The parameters are converted (using standard type conversions) to the type of the control variable which may be integer, real, or double precision -- FTN4 requires the control variable to be of type integer.

Although the control variable assumes a meaningful value with both FTN4 and FORTRAN IV-Plus when a DO loop is exited by a jump outside its range, with FTN4 the value is (possibly) undefined after the loop is exited by a natural fall through the bottom. In order to port programs the following rules should be followed:

1. Use only integer control variables.
2. Use simple variables or constants for parameters.
3. Do not allow any of the parameters to assume negative values (an increment of zero will not produce an infinite loop with FORTRAN IV-Plus)
4. Do not redefine the control variable or any parameter within the extent of the loop.
5. Do not rely on the control variable having a meaningful value if the loop is exited by dropping through (as opposed to jumping out).

- 7.2.4 CONTINUE statements: No known problems.

- 7.2.5 Subroutine CALL statements: FTN4's unwieldy CALL ... RETURNS construct, which allows automatic branching upon return from a subroutine, should not be used.

FORTRAN IV-Plus offers two data types for manipulation of characters: Hollerith and CHARACTER. Constants for both are enclosed in single quotes. FORTRAN IV-Plus is able to determine which interpretation is intended (they have different internal representations) by examining the data type of the variable it is being associated with. When a quoted constant is used as an argu-

ment to a subprogram, however, the data type of the variable it is associated with is unknown and FORTRAN IV-Plus assumes CHARACTER. This will cause problems because of the difference in internal representation and the fact that CHARACTER variables must be declared. The solution -- always pass Hollerith constants to subprograms in the nH form, not the quoted (or # 'ed) form. (DEC plans to eliminate this problem some time in the future).

7.2.6 RETURN statements: Do not use FTN4's RETURN i (in conjunction with a RETURNS list) feature. With both dialects a RETURN statement is implied before the END statement in a subprogram, and may be omitted.

7.2.7 PAUSE statements: If an integer variable is specified FORTRAN IV-Plus prints it in decimal, while FTN4 will print it in octal. The PAUSE statement has further implications on the BKY-7600, and its use is not recommended.

7.2.8 STOP statements: As with PAUSE statements, if an integer variable is specified FORTRAN IV-Plus prints it in decimal, while FTN4 will display the value in octal.

7.3 Type Declarations

The compatible data types are:

```

      INTEGER
      REAL
      COMPLEX
      DOUBLE PRECISION
      LOGICAL
  
```

With FORTRAN IV-Plus INTEGER, REAL, and LOGICAL are the same as INTEGER * 4, REAL * 4, and LOGICAL * 4; DOUBLE PRECISION is the same as REAL * 8 (assuming programs are compiled with the default /I4 switch).

FTN4 allows only one IMPLICIT statement per module.

7.4 EXTERNAL declarations

When the name of a user supplied function is the same (lexically and in data type) as the name of a FORTRAN intrinsic (library) function the FTN4 user must declare the name EXTERNAL in each module in which it is to supersede the library function. In the same situation, FORTRAN IV-Plus users must prefix the name with an asterisk in an EXTERNAL statement.

FTN4 does not require intrinsic functions to be declared EXTERNAL if they are to be passed as arguments. FORTRAN IV-Plus requires all functions passed as arguments to be declared exter-

nal -- including intrinsic functions. For a list of library functions, see section 8.

1. Declare as EXTERNAL all functions passed as arguments .
2. Do not use either FTN4 or FORTRAN IV-Plus library function names as subprogram names.

7.5 Storage Allocation

7.5.1 Array storage declarations: The data type used in array declarations should be one of the compatible data types mentioned above. For compatibility with FTN4, a maximum of three dimensions should be used. For arrays with variable dimensions, the array and its size must be dummy parameters (FORTRAN IV-Plus allows the size to be passed in COMMON). The VIRTUAL statement in FORTRAN IV-Plus is equivalent to the DIMENSION statement.

7.5.2 COMMON statement: With both FTN4 and FORTRAN IV-Plus, when two or more modules initialize the same COMMON variable, the initialization by the last module loaded holds. FORTRAN IV-Plus allows extension of a COMMON block's length by any module, FTN4 requires that the longest occurrence be the first in the load (link) sequence. FORTRAN IV-Plus, but not FTN4, allows initialization of blank COMMON by a DATA statement.

Although not a good practice, both dialects allow COMMON block names to be used also as subprogram names . The following list of pointers should be followed to improve portability:

1. Do not initialize blank COMMON with a DATA statement.
2. ~~Avoid using COMMON block names which are also used as sub-~~
routine names, and do not use FTN4 numbered COMMON blocks.
3. Make all COMMON blocks with the same name the same length. If different lengths are used for the same block, load the longest first.
4. If one COMMON statement declares multiple COMMON blocks, do not separate the blocks with a comma.

7.5.3 DATA statements: When a variable in a DATA statement and the constant to which it is being initialized differ in type, the two dialects may produce different results. FTN4 will assign the constant to the variable with no type conversion , while FORTRAN IV-Plus will convert the constant to the type of the variable.

FORTRAN IV-Plus requires that the number of elements to be initialized equals the number of constants provided. FTN4 will disregard excess constants and set excess variables to an 'uninitialized' value. With both dialects, if a variable is initialized by a DATA statement more than once, the last definition holds. Neither dialect allows single literal spillover into multiple elements, as some FORTRAN dialects do.

On the VAX, memory is preset to zero, so uninitialized variables assume the value zero. This is not so with the BKY system (unless the SETCORE,ZERO control card is used).

1. Do not use implied DO loops in DATA statements (FTN4 allows this)
2. Use only simple repetition factors such as 'integer*constant'.
3. Insure an exact match between the number of elements to be initialized and the number of constants.
4. The variable being initialized and its corresponding constant should agree in type.
5. Do not use the parenthesized form of the DATA statement provided by FTN4.
6. Do not initialize variables in blank COMMON with a DATA statement.
7. Do not rely on memory being preset to any particular value.

7.5.4 EQUIVALENCE statements: Both dialects allow the extension of a COMMON block beyond its last element due to EQUIVALENCE relations. Remember, however, that with FTN4 the longest occurrence of a block must be loaded first.

The EQUIVALENCE statement is the only construct in which FORTRAN IV-Plus allows multiple subscripts to be omitted. In the EQUIVALENCE statement, multiple subscripts may be replaced by a single subscript.

7.6 Program Modules

7.6.1 PROGRAM statement: FTN4 requires the first statement (other than comments) in every mainline be a PROGRAM statement. If the PROGRAM statement is not found, the statement:

PROGRAM START.(INPUT,OUTPUT)

is assumed. FORTRAN IV-Plus has the PROGRAM statement, but it is not necessary to use it. FTN4's PROGRAM statement also contains file association information which is not found in VAX's FORTRAN. See section 7.8 for more information on how file associations are made in the two dialects.

7.6.2 FUNCTION subprograms: FORTRAN IV-Plus allows parameterless functions -- a feature which FTN4 lacks. To avoid problems, all functions should have at least one parameter. For information on functions with multiple entry points see section 7.6.5. For information concerning conflicts between intrinsic (library) functions and user defined functions see section 7.4 .

7.6.3 SUBROUTINE subprograms: Again, do not use FTN4's CALL ... RETURNS construct.

7.6.4 BLOCK DATA subprogram: Within a BLOCK DATA subprogram all COMMON blocks present should be defined with the maximum length they will assume. This is required by FORTRAN IV-Plus.

7.6.5 ENTRY points: The ENTRY statement supported by FORTRAN IV-Plus allows the specification of parameters in addition to those used by the module containing the ENTRY point. The following diagram illustrates this point:

To avoid conflicts with FTN4, ENTRY points with parameters should not be used.

ENTRY points in FUNCTION subprograms pose another problem. With FTN4 all function values, independent of the ENTRY point into the FUNCTION, are returned by assigning values to the FUNCTION name. As a consequence, the data type returned by all entry points is the same. With FORTRAN IV-Plus, if a FUNCTION is entered at a given ENTRY point, the return value must be assigned to that ENTRY point name. This scheme allows different entry points to return results of different data types. Porting between the VAX and the BKY system is made easier if there is only one exit point per module.

-
1. ~~Do not declare ENTRY points with parameters.~~
 2. Within one module, all ENTRY points should be of the same type.
-

7.7 Statement Functions

The expansion of code for statement functions is done inline by FTN4. FORTRAN IV-Plus constructs a small function subprogram.

7.8 Input / Output Statements

Before getting involved in I/O statements, it is useful to investigate the environment in which the I/O is to take place. The following description attempts to outline the file environment in which each dialect exists. Some detail has been sacrificed to preserve brevity and simplicity.

The concept of a file is basic, and both the VAX and the BKY system provide services which incorporate this concept. In both systems, the file serves as an identifier with which to reference a group of logically

associated bits. Unfortunately the specifics vary as to of how this is incorporated into FORTRAN dialects (FTN4 and FORTRAN IV-Plus) and the operating systems (BKY and VMS). To some extent the approach taken by each system is a consequence of the one machine being designed for time-sharing and the other for batch processing.

To contrast basic differences between the file structures some characteristics of files on each system are compared below.

- o Operations done to files -- The BKY system can perform the following operations on files - CREATE, READ, WRITE, LOAD, EXECUTE, REPOSITION, and RELEASE. The VAX is capable of all these operations except REPOSITION. A file on the BKY system may be repositioned to many different places, corresponding to different markers within the file. Files also retain their position from one operation (control card) to the next. After each operation on the VAX, all files are 'closed', which effectively repositions the file to the beginning (BOI mark).
- o Lifetime of a file -- The BKY file system allows the programmer to store files which are frequently referenced on devices which have fast access times. To do this efficiently files become more volatile as the time required to access them decreases. The fastest access occurs when a file is buffered in from disk. Most disk files remain intact for the duration of the run but are destroyed by the operating system after that. CACHE files on the BKY-7600, and COMMON files on the 7600 and the 6000's, provide less volatile file storage while still taking advantage of the speed of the disk -- files will generally remain a few days after a run (reference) before being deleted by the operating system. There are no user owned permanent disk files on the BKY system. For permanent file storage on the BKY-system, PSS or another form of mass storage is used. All file storage on the VAX is permanent until deleted by the user.
- o File identification -- All files on the BKY system are identified by name, not device or function. A file name consists of a maximum of seven characters. Files on the VAX are uniquely identified by a device name, a directory, a file name, an extension (file type), and a version number. For a definition of these terms see section 9.1. On the VAX, a logical name may be used to refer to files or devices by other than their specific names. When a reference is made to a logical name, the system translates the logical name into its defined equivalent name. A logical name must not contain a period, colon, semicolon, or square bracket. Logical names may be defined by the system at login time, or by the user through the use of the ASSIGN command. To display the value of currently defined logical names the command SHOW LOGICAL is used.
- o Default devices -- All files on the BKY-7600 are disk files, on the 6000's all files are disk files unless otherwise specified (for

example by REQUESTing a tape) . On the VAX, if no device name or directory is given, the users default disk and directory are assumed.

FTN4 provides an integrated mechanism to interface with files on the BKY system, namely the PROGRAM statement. The PROGRAM statement

1. Associates a file with the logical unit number *n* via the file=TAPEn construct.
2. Defines a positional dependence between file names, allowing files to be changed by the control card which causes execution of the program.
3. Communicates pertinent information such as buffer size.

FORTTRAN IV-Plus associates each logical unit number *n* with the logical name FOR00*n*. If this name is present in the logical name translation table at run time, its defined equivalent name is used. If FOR00*n* is not in the logical name translation table at run time, it defaults to the users disk file named FOR00*n*.DAT .

The FORTRAN IV-Plus OPEN statement is capable of associating, at run time, a logical unit number with any file on the system. The basic format of the OPEN statement is:

```
OPEN(UNIT=n,NAME='file_name')
```

For more information on the OPEN statement see section 6-1 of the FORTTRAN IV-Plus Language Reference Manual.

The following example illustrates some of the points made above. The first program fragment is a BKY-7600 job and the second is its VAX equivalent.

7600-fragment

```

      .
      .
      FTN4.                                {compile program}
      STAGE,TAPE5,<reel number>.           {stage in tape5 to disk}
      LINK,F=LGO,X.                       {link and execute}
      STAGE,TAPE6,W,<reel number>.         {stage out tape6 to tape}
      .
      .

```

7/8/9

```
PROGRAM FOO(INPUT,OUTPUT,TAPE5,TAPE6)
```

```

      .
      .
      READ 1000,LIST1                      {read from 'INPUT'}
      PRINT 1000,LIST2                     {write to 'OUTPUT'}

```

READ(5) LIST3
WRITE (6) LIST4

{read TAPE5}
{write TAPE6}

7/8/9

VAX-fragment

\$ TYPE FOO>FOR
PROGRAM FOO

{print FORTRAN source }

OPEN(UNIT=6, NAME='DUMMY')

{associate unit 6 with
logical name DUMMY }

READ 1000,LIST1

{read from device associated
with logical name SYS\$INPUT}

PRINT 1000,LIST2

{write to device associated
with logical name SYS\$OUTPUT}

READ (5) LIST3

{read from file associated
with logical name FOR005}

WRITE (6),LIST4

{write to file associated
with logical name FOR006}

END

\$ FORTRAN FOO

{compile program}

\$ LINK FOO

{link program}

\$ ALLOCATE MTA0:

{reserve tape drive}

\$ MOUNT MTA0: TAPE5 FOR005

{mount tape with label TAPE5
and associate logical name
FOR005 with that volume }

\$ ASSIGN TAPE6.DAT DUMMY

{associate logical name
DUMMY with disk file TAPE6.DAT }

\$ RUN FOO

{execute program }

\$ DISMOUNT MTA0:

{remove tape5 from drive}

\$ INITIALIZE MTA0: TAPE6

{declar new volume TAPE6
on device MTA0: }

\$ MOUNT MTA0: TAPE6 TAPE6

{mount that volume and
associate logical name TAPE6 with i
{copy file from disk to tape}

\$ COPY TAPE6.DAT TAPE6

{remove tape from drive}

\$ DISMOUNT MTA0:

{release tape drive so others
may use it }

\$ DEALLOCATE MTA0:

- 7.8.1 PRINT,
 7.8.2 PUNCH {not available with FORTRAN IV-Plus },
 7.8.3 WRITE, and
 7.8.4 READ statements: I/O statements may be classified by the type of I/O to be performed (formatted, unformatted, or list directed) and by whether the unit number is specified by the statement or implied. List directed I/O is similar to formatted I/O except the format statement is constructed automatically according to the data type of the elements in the I/O list. The following table gives examples of compatible constructs in each class.

	READ	WRITE	PRINT
Unit specified			
formatted	READ(u,fn)list	WRITE(u,fn)list	
unformatted	READ(u) list	WRITE(u) list	
list directed	READ(u,*) list	WRITE(u,*) list	
Unit implied			
formatted	READ fn,list		PRINT fn,list
unformatted			
list directed	READ *,list		PRINT *,list

READ statements with implied unit numbers cause a file named 'INPUT' to be read on the BKY-system and cause the file associated with the logical name SYS\$INPUT to be read on the VAX. PRINT statements cause output to an implied unit number associated with the file 'OUTPUT' on the BKY-system and to the file associated with the logical name SYS\$OUTPUT on the VAX.

- 7.8.5 BUFFER IN, BUFFER OUT statements: FTN4's BUFFER IN and BUFFER OUT statements offer a facility to perform asynchronous I/O (on the 7600 they are equivalent to unformatted READ and WRITE statements). FORTRAN IV-Plus on the VAX offers no special statements for performing asynchronous I/O. Rather, asynchronous I/O is performed by calling 'system service' routines. See the VAX/VMS System Service Reference Manual for details.

- 7.8.6 NAMELIST I/O : The NAMELIST facility is completely absent from FORTRAN IV-Plus. If programs are to port, this feature of FTN4 should not be used.

7.8.7 ENCODE DECODE statements : Aside from the problems concerning the number of characters stored per word, ENCODE / DECODE statements should port. See section 7.9 for more information.

7.8.8 File Manipulation Statements : REWIND, BACKSPACE, and ENDFILE statements should port. An ENDFILE writes a CTRL-Z to the file associated with the unit on the VAX, and an EOF mark on the BKY-system. On the VAX, a file OPENed in append mode may not be BACKSPACed. A REWIND repositions a file at its begining. On the VAX, files are effectively rewound after every operating system command.

For information on the OPEN and CLOSE statement see section 9 of the FORTRAN IV-Plus Language Reference Manual.

7.9 Format Statements: FTN4 allows considerable variation in permissible format specifications. For instance:

FORMAT(X2HX/F10.4/XI3) is the same as
FORMAT(1X,2HX/,F10.4/,1X,I3) .

To insure portability, the following guidelines should be followed.

1. All field specifiers should be seperated by a comma or a slash.
2. Do not use the default field width facility provided by FORTRAN IV-Plus. (FORMAT(I,F,I) is legal on the VAX).
3. Use only the compatible field specifiers listed below

nIw	Fw.d	nH
nOw	Ew.d	nX
nZw	Dw.d	nP
nLw	Gw.d	/
nAw		,
Tn		

good luck

8.0 Libraries

The libraries provided with the two dialects contain all functions required to be in accordance with the ANSI '66 standard. These functions are listed in alphabetical order below. Functions in this list which are suffixed with an asteriks (*) are not mentioned in the ANSI standard, but are provided (in a compatible form) by both dialects. For all arithmetic functions, loss of precision should be expected in going from the 60 bit BKY machines to the 32 bit VAX.

ABS	CEXP	DLOG	ISIGN
ACOS*	CLOG	DLOG10	MAX0
AIMAG	CMPLX	DMAX1	MAX1
AINI	CONJG	DMIN1	MIN0
ALOG	COS	DMOD	MIN1
ALOG10	COSH*	DSIGN	MOD
AMAX0	CSIN	DSIN	REAL
AMAX1	CSQRT	DSQRT	SIGN
AMINO	DABS	DTANH*	SIN
AMIN1	DATAN	EXP	SNGL
AMOD	DATAN2	FLOAT	SQRT
ASIN*	DBLE	IABS	TAN*
ATAN	DCOS	IDIM	TANH
ATAN2	DCOSH*	IDINT	
CABS	DEXP	IFIX	
CCOS	DIM	INT	

FORTRAN IV-Plus allows most of these functions to be referenced with a generic name. As an example, if A is of type COMPLEX, then EXP(A) may be used instead of as CEXP(A). The compiler, by checking the type of arguments used, is able to distinguish which function name to substitute for the name of a generic function reference. Because the FTN4 compiler does not have this capability, the function specifically designed for the type of argument being passed should be used.

Those library function names unique to one system are listed in the following table. When two functions are the same(or similar) they are listed on one line. A brief description of each function is also given. Some of the functions in the following table are not part of the standard FTN4 library, these are suffixed with a plus sign. For further information and specifics see:

For the VAX:

- o VAX FORTRAN IV-Plus Language Reference Manual, Appendix B
- o VAX FORTRAN IV-Plus User's Guide, Appendix C
- o VAX Common Runtime Procedure Library Reference Manual
- o VAX System Services Reference Manual

And for the BKY-system:

- o BKY FTN4 Reference Manual, Section I-8
 - o WRITEUPS, subset FTN4LIB
 - o WRITEUPS, subset FTN4LST
 - o WRITEUPS, subset FTN4
-
-

FORTTRAN IV-Plus	FTN4	description	type
Mathmactical Functions			
ANINT		nearest integer	R->R
DACOS		double precision arc-cosine	D->D
DASIN		double precision arc-sin	D->D
DFLOAT		double precision float	I->D
DNINT		nearest integer	D->D
DPROD		double precision product	R,R->D
DSINH		double precision hyperbolic siD->D	
DTAN		double precision tangent	D->D
IDNINT		nearest integer	D->I
NINT		nearset integer	R->I
RAN	RANF	random number generator (0.,1.)	
	RANGET	get seed from RANF	
	RANSET	set seed for RANF	
	RGEN	another generator (0.0,1.0)	
	STOGEN	set seed for RGEN	
	LODGEN	get seed from RGEN	
SINH		hyperbolic sin	
Logical Functions			
	IRIGHT+	see right	
	RIGHT+	right shift, end off, zero fill	
	ILEFT+	see left	
	LEFT+	left shift, end off, zero fill	
IAND	AND	logical and	
IEOR	XOR	exclusive or	
IOR	OR	logical or	
ISHFT	SHIFT	shift	
NOT	COMPL	complement	
CHARACTER Functions			
CHAR		CHARACTER*1 of ASCII value	
ICHAR		ASCII value of CHARACTER*1	
INDEX		index of substring	
LEN		length of string	

FORTTRAN IV-Plus	FTN4	description
Day of Year		
DATE IDATE	DATE DATE6+ DATE7+ JDATE+	date of year in different forms
Time of Day		
SECNDS TIME	SECONDS TIME HOUR+	seconds (real) time of day hh.mm.ss
I/O Subprograms		
END=	CLOSMS CONNECT+ DELETE+ DISCONN+ DISPOSE+ EOF IOCHEC LABEL LENGTH LENGTX NXTAPE+	close mass storage connect terminal /6000's delete file disconnect terminal /6000's dispose file to queue check unit status check unit status set tape label attributes multiple tape request
SYS\$...	OPENMS PACK+ Q9.... READEC READMS RETURNS+ REWUL RQTAPE STATUS STINDX T9TTYOP+ UNIT UNPACK+ WRITEC WRITMS	open file on mass storage place info into buffer general I/O routines read from extended core read from mass storage return a file rewind and unload file request a tape to be mounted accounting information use subindex for random file perform a terminal action test status of a unit get info from buffer write to extended core write to mass storage
Error Handeling		
LIB\$... ERRSNS ERR=	DUMP ERRSET PDUMP STRACE SYSTEM SYSTEMC WARN+	perform a dump " " trace routines special error handling facility

FORTTRAN IV-Plus	FTN4	description
Miscellaneous Subprograms		
%LOC	DISPLA IF76+ JOBCARD+ LEGVAR LOCF NARG+ OVERLAY SLITE SLITET SETFLL+ SETFLS+ SSWTCH	message to dayfile true if 7600 get jobcard image test to see if variable is legal return location of variable number of arguments overlay directives set light sense light change memory requirements sense switch

The ERR= and END= specifications given for the VAX are not really library functions, but are listed in the table because their action is similar to some of FTN4's library routines. The ERR= specification may occur in any OPEN, CLOSE, READ, or WRITE statement. If an error occurs during that operation (that of OPENing a file, etc...), and the ERR= is present, control is transferred to the statement label indicated by the integer constant following the equal sign. The END= may occur in READ statements only. In this case, when an end of file is encountered control is transferred to the label following the equal sign. As an example, the statement

```
READ(5,100, END= 200, ERR= 300) LIST
```

would transfer control to label 200 if an end of file were encountered during the read, and to label 300 if there were an input conversion error. The FTN4 code fragment

```

      .
      .
      READ(5,9000) LIST
      IF(EOF(5)) 200,100,200
100      CONTINUE
      .
      .
C-----EOF ON UNIT 5
200      CONTINUE
      .
      .

```

is equivalent to the FORTRAN IV-Plus fragment

```

      .
      .
      READ(5,9000, END=200) LIST
      .
      .
C-----EOF ON UNIT 5
200      CONTINUE
      .
      .

```


9.0 Summary of VAX-11/780 Control Language (DCL)

9.1 Notation: The following notation is used to identify devices and files on the VAX.

device_name : the name of a physical or logical device. Physical device names are of the form:

xxln:

where

xx is the device identifier. { LP - line printer, DB - disk drive, TT - teletype, MT - mag tape drive }
 l is a single letter indicating the controller to which the device is attached.
 n is the line number of the controller to which the device is attached.

A logical_device is any logical name which is associated with a physical device. Some of the logical devices defined by the system are:

TT: - users terminal
 MT: - a mag tape drive
 DISK: - users disk
 SYS\$INPUT - where system reads are done
 SYS\$OUTPUT - where system writes are directed
 SYS\$error - where error messages are directed
 SYS\$CONTROL - initial login value of SYS\$OUTPUT

volume When information resides on a medium which is separable from a device (such as a disk pack or reel of tape) the medium is called a volume and is identified, during mounting operations, by a volume label.

directory is a name used to reference particular partitions within a volume. Each user has two directory names, both associated with the same partition. One consists of "[name]", and the other "[project num, prog num]". The default directory is the one which the user is logged in under.

file_spec A file_spec specifies one or more files. All files reside within some directory on a volume. A volume must be mounted to access a file. A complete file specification consists of:

device_name:[directory]file_name.ext;version

the defaults are:

device_name - users default disk, automatically assigned at login time.

directory - users default directory, automatically assigned at login time
 file_name - none { FORTRAN I/O with logical unit n defaults to a file in users default disk area named FOR00n.DAT } A file name may be a maximum 9 characters long. The legal characters are (A-Z,a-z,\$,_), the case of the characters is not significant.
 extension - maximum of three letters long, no default. The legal characters are the same as for file names.
 version - most recent

A file extension may not be given without a file name. A *, called a wildcard, may be used in any field except the device field to indicate all possible field values. Thus

[*]*.FOR;*

refers to all versions of all files in all directories on the users default device with the extension .FOR . Because one volume may contain files belonging to many users, protection schemes prevent unprivileged users from damaging any files outside his directory partition.

9.2 Basic Commands: In the list of basic commands below the full command followed by its minimal abbreviation and its basic parameters. Text enclosed in curly brackets is optional. For a complete description of the commands see:

- o VAX/VMS Command Language User's Guide, or
- o use the HELP command, specifying as a topic the command about which information is desired.

ALLOCATE AL device_name{:} {logical_name{:}}
reserve a device, and establish a logical name for it.

ASSIGN AS device_name{:} logical_name{:}
equate a logical name with a device name or another logical name.

COPY COP input_file, ... output_file
copy input file(s) to output file(s).

DEALLOCATE DEAL device_name{:}
return reserved device to system pool.

DEASSIGN DEAS logical_name{:}
cancel a logical name assignment

DELETE DEL file_spec, ...
delete files or queue entries

DIRECTORY DIR file_spec, ...
display information about indicated files.

DISMOUNT DIS device_name{:}
release previously mounted volumes on indicated device. The device still remains reserved.

EDIT ED file_spec
Enter SOS editor

FORTTRAN F file_spec, ...
Compile files indicated by file_spec with FORTRAN compiler. The default extension is .FOR. The compiler produces an object file of the same name, with the extension .OBJ.

HELP H topic
gives information describing the use of the system routine 'topic'.

INITIALIZE INIT device_name{:} volume_label
write a label to initialize a mass storage volume.

LINK LIN file_spec, ...

link compiled or assembled files, to produce an executable image. The default extension is .OBJ. LINK produces an image file of the same name, with the extension .EXE . as input, but with extension .EXE .

LOGOUT LO

log off a terminal

MACRO MA

file_spec, ...
invoke MACRO assembler on specified files. The default extension is .MAR . The assembler produces a file with same name, with the extension .OBJ .

MOUNT MO

device_name {volume_label} {logical_name{:}}
requests a volume to be mounted on the specified device.

PRINT PR

file_spec, ...
Queues specified files to a line printer.

PURGE PU

file_spec
delete all but the most recent version of the specified files.

RUN R

file_name
loads and begins execution of an executable image produced by the LINK command. Default extension is .EXE .

SET SE

sets various attributes, see HELP SET for information.

SHOW SH

displays various attributes, see HELP SHOW for information.

TYPE T

file_spec, ...
display indicated files at the users terminal.

10.0 Transferring Programs

10.1 Transferring programs from the BKY System to the VAX

The two utilities used in transferring programs (character data) from the BKY system to the VAX are ENCODE and MTREAD.

ENCODE runs on the 6000's at BKY. It accepts as input a packed display code local file and produces its ASCII (or EBCDIC) equivalent on a nine track tape. The tape is written unlabeled, with fixed length logical records (lines). The ENCODE control card has the form:

ENCODE, I=infile, B=outfile, L=listfile, RL=record length,
BF=blocking factor, NF=number of files, NR=number of
records, M=character set.

The defaults are:

I=TAPE1	L=OUTPUT	BF=45	NR=99999999
B=TAPE2	RL=80	NF={all}	M=EBCDIC

The only parameter that must be specified when preparing a tape for the VAX is M=ASCII.

A typical control card sequence for writing FORTRAN programs (record length 80) out to tape would be:

```

      .
      .
      REQUEST,TAPEN,<reel #>,D9,NT,TR,W,NN.
      .
      .

```

```

      .
      .
      FETCHPS,BKYLGOB,ENCODE,ENCODE.
      .
      .

```

```

      .
      .
      LIBCOPY,<my library>,FTN4PGM,<my subset>.
      .
      .

```

```

      .
      .
      ENCODE,I=FTN4PGM,B=TAPEN,L=VAXTAPE,M=ASCII.
      .
      .

```

```

      .
      .
      RETURN,TAPEN.
      .

```

```

      .
      .
      DISPOSE,VAXTAPE=PR,DT=I,R=[info],M=DP.
      .
      .

```

The list file (VAXTAPE) should be kept with the tape -- it contains a record of the files on the tape and information about the tape format. For more information about the use of ENCODE see BKY library HANDBOOKS,

subset UTILITY.

To gain insight into how non-standard a particular FTN4 program is it may be compiled by the FTN4 compiler with the parameter EL=A. This will list all non-ANSII constructs used in the program. One possibility for cleaning up FTN4 before porting is to run it through the BKY utility CLEAN. For more information on CLEAN see BKY library HANDBOOKS, subset CLEAN.

To read a tape written by ENCODE on the VAX the utility MTREAD is used. MTREAD will handle a variety of tape formats. For our purpose, the parameters to MTREAD must specify the same tape format used by ENCODE -- i.e. a fixed record length ASCII tape.

Before running MTREAD the tape drive is reserved with the ALLOCATE command, and the tape is mounted with the MOUNT command. MTREAD asks for various parameters, all responses must be in UPPER case. The following is a typical command sequence, underlined portions are typed by the VAX. A 2 represents a carriage return.

```

.
.
$ ALLOCATE MTA0: 2
  MTA0: allocated
$ MOUNT/DENSITY:1600/FOREIGN MTA0: 2
$ RUN [SYSEXE]MTREAD 2
MAGTAPE UNIT TO ASSIGN : MTA0: 2
ASCII DATA OR BINARY ? [A/B] : ASCII 2
LOGICAL RECORD SIZE IN BYTES : 80 2
FILE POSITION : 1 2
DISK FILE NAME : FTN4PGM.FOR 2
111 RECORDS TRANSFERED
FILE POSITION : 0 2
.
.
$ DISMOUNT MTA0: 2
$ DEALLOCATE MTA0: 2

```

The record length (size) should be the same as the RL parameter used with ENCODE. If no RL parameter was used with ENCODE, the record length for MTREAD should be 80. The file position is the number of files from the beginning of the tape to, and including, the file you wish to transfer. A file position of zero terminates MTREAD. The DISMOUNT and DEALLOCATE commands return the tape drive to the device pool so someone else can use it. Further information on MTREAD is contained in the VAX file [SYSDOC]MTREAD.DOC.

10.2 Transferring programs from the VAX to the BKY system

The utilities used in transferring programs(character data) from the VAX to the BKY system are MTWRITE and CODE9. A tape written with either MTWRITE or ENCODE may be read with either MTREAD or CODE9. In transferring programs from the VAX to the BKY system, MTWRITE ,on the VAX, writes the tape and CODE9 on the 6000's reads the tape.

MTWRITE produces an unlabeled ASCII tape with fixed length records. As with MTREAD, the tape drive must first be reserved and mounted with the ALLOCATE and MOUNT commands. Again, all responses to MTWRITE must be in UPPER case. MTWRITE produces a log file containing a synopsis of what was transferred to tape. A typical command sequence to write a FORTRAN IV-Plus program on tape with MTWRITE follows:

```

.
.
.
$ ALLOCATE MTA0: 2
  MTA0: Allocated
$ MOUNT/FOREIGN/DENSITY:1600 MTA0: 2
$ RUN [SYSEXE]MTWRITE 2
  Log file name : log.lis 2
  Tape unit to assign : MTA0: 2
  Logical Record Length : 802
  Blocking Factor : 452
  First disk file to be written : mypgm.for 2
  Next disk file to be written : 2
$ DISMOUNT MTA0: 2
$ DEALLOCATE MTA0: 2
.
.
.

```

As a precaution, before writing a tape, convert all tabs to multiple spaces . This can be done by using DETAB. For further information about MTWRITE see [SYSDOC]MTWRITE.DOC.

To read tapes written by MTWRITE on the 6000's the utility CODE9 is used. CODE9 converts 9 track ASCII tape files to 6-bit packed display code local files. The control card has the form:

```

CODE9,I=infile,B=outfile,L=listfile,RL=record length,
      BF=blocking factor,NF=number of files,M=character
      set,NB=number blocks/file,PE=number of parity errors,
      BS=buffer size,LE=number of length errors.

```

The defaults are

I=TAPE1	L=OUTPUT	BF=1	NB=999999
B=TAPE2	RL=80	NF=1	M=EBCDIC
PE=0	BS={dont use}		LE=10

PE=0 BS={dont use} LE=10

As with ENCODE, the only parameter which must be specified is M=ASCII.

To read the first file off a tape written with MTWRITE using a record length of 80 and a blocking factor of 45, the following sequence of control cards might be used.

```

      .
      .
REQUEST,MTVAX,<reel #>,D9,NT,PD,NN.
      .
      .
FETCHPS,BKYLGOB,CODE9,CODE9.
      .
      .
CODE9,I=MTVAX,B=FORPGM,L=LIST,BF=45,RL=80,M=ASCII.
      .
      .
LIBRITE, <my library>, FORPGM, <my subset>, <group>,
      G=<group name>.
      .
      .

```

See HANDBOOKS, subset UTILITY for further information on CODE9.

BIBLIOGRAPHY

1. American National Standard FORTRAN. American National Standards Institute (ANSI) X3.9-1966 .
 2. Garner, W.R. . Characteristics of FORTRAN . NASA tech brief 73-10322 / LAR-11177 Jan 77 .
 3. Messerli, R., Metcalf, M., Ranjard, F. . Basic Coding Standards in FORTRAN . CERN April 76 .
 4. Ripley, G.D., White, J.Wm., A Survey and Analysis of Mini-Computer FORTRAN Dialects. University of Arizona, CS-tech report. May 77 .
 5. Stevens, D. A Comparison of FTN and FORTRAN H Extended. CERN /DD/US/3 June 76.
 6. White, J.W., Ripley, G.D. . How Portable are Mini-Computer FORTRAN Programs ? . DATAMATION July, 77.
-

0 0 0 0 5 1 0 4 4 6 4

This report was done with support from the Department of Energy. Any conclusions or opinions expressed in this report represent solely those of the author(s) and not necessarily those of The Regents of the University of California, the Lawrence Berkeley Laboratory or the Department of Energy.

TECHNICAL INFORMATION DEPARTMENT
LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA
BERKELEY, CALIFORNIA 94720