TN-16

SYSTEM SPECIFIC PROGRAMME INPUTS FOR DOCUMENTED PROGRAMMES

SATISH CHANDRA
DIRECTOR

STUDY GROUP

DEEPA KARAWADE

NATIONAL INSTITUTE OF HYDROLOGY
JAL VIGYAN BHAWAN
ROORKEE-247667(UP)

1985-86

CONTENTS

ABSTRACT

FORTRAN is now used on all major computer systems. Major versions in vogue are FORTRAN, FORTRAN II and FORTRAN IV. Each new version made a few changes in the basic instruction of FORTRAN and included additional features. A FORTRAN program developed for a particuar computer can not be executed on other type of computers. To make it executable for other computers, some statements of the FORTRAN program are required to be modified according to the FORTRAN compiler of that computer.

This report described various modifications, necessary for successfully implementing a FORTRAN program on DEC-2050, PDP-11 and UNIVAC-1100 which has been developed on VAX-11 and vice versa.

1.0 INTRODUCTION

Today computers, with their fantastic speed and accuracy, are becoming powerful tools for problem solving in diverse fields including scientific, research, business, medicine, school, house management etc.

The problem of working with the computers is that they can interpret and execute only those instructions which are written in the machine language (low level language), which is very complex and cumbersome to use. However,high level languages like FORTRAN, COBOL, BASIC etc. have been developed which are much easier for humans to use. FORTRAN, the most widely used language for scientific and research work has been discussed here.

1.1 The FORTRAN language

FORTRAN(an acronym for FORmula TRANslation) is the most widely used of a class of high-level languages called scientific and algebraic languages. It is available for use on almost all computers. Although not limited to mathematical problems, it is especially useful for problems that can be stated in terms of formulas or arithmetic procedures.

FORTRAN is a machine-independent language for instructing a computer. In other words, the programmer using FORTRAN does not need to know any machine-level details for the computer being used. The language is procedure-oriented, designed for instructing the computer in a problem-solving procedure. The language consists of a vocabulary of symbols and words and a grammer of rules for writing procedural instructions. The symbols, words and rules utilize many common mathematical and English-language conventions so that the language is fairly easy to learn and to understand. The rules are, however, precise and must be followed with care. In other words learning FORTRAN is like learning a special-purpose language. There are rules of construction and vocabulary to learn, and one becomes proficient by doing rather than by much reading.

1.2 Review of FORTRAN Development

FORTRAN was developed in 1957 by IBM in conjunction with some major users, but it is now used on all major computer systems. FORTRAN has changed and evolved. This evolutionary process resulted, during the development period, in several FORTRANs of increasing complexity. Major versions were called FORTRAN, FORTRAN II and FORTRAN IV. Each new version made a few changes in the basic instructions and included additional features. In 1966, a voluntary FORTRAN standard, American National Standard(ANS) FORTRAN, was adopted. The International Standards Organisation(ISO) also defined standard FORTRAN.

A revised American National Standard (ANS) FORTRAN was adopted in 1977. This 1977 standard adds features to the previous 1966 standard FORTRAN, clarifies some ambiguities, and makes a few minor changes. The teaching-oriented compilers were designed to provide excellent error-diagnostic messages for students, do fast execution of small student programs, and relax some error-prone features of FORTRAN. The new 1977 FORTRAN standard adopted the most significant features of the teaching oriented FORTRANs, so the American National Standard FORTRAN is recommended as the basis for all FORTRAN programming, by students as well as by professional programmers.

2.0 PROBLEM DEFINITION

A FORTRAN program, developed for a particular computer can not be executed on other type of computers. To make it executable for other computers, some statements of the FORTRAN program are required to be modified according to the FORTRAN compiler of that computer. The objective of the present work is to discuss the various necessary changes for successfully implementing a FORTRAN programme on DEC-20/PDP-11/UNIVAC-1100 which has been developed on VAX-11 and vice versa.

The compatibility between VAX-11 FORTRAN and other systems FORTRAN will be of great use to FORTRAN programmers.

The VAX-11 is a family of DIGITAL's 32 bit minicomputers. It is a fully integrated computer system with a powerful virtual memory operating system. VAX-11 supports a 32-bit word architecture thereby establishing a virtual address space of 2.32 M bytes for user application.

The DEC-2050 is a medium scale computer and features a high performance, virtual memory system that provides a multitasking, multiprogramming environment to support concurrency, time sharing and batch processing. The TOPS-20 operating system supports a 36-bit word architecture and 2.4M bytes of high speed memory.

The PDP-11 computer family is a wide range of compatible processors complemented by a variety of peripheral devices and software. A variety of operating systems, languages and communications software are available for the PDP-11 computer family such as RT-11, DSM-11, RSTS/E, RSX-11 etc.

The Sperry UNIVAC 1100 series computer is a time sharing, multiprocessor system which supports a 9 bit-byte and 36 bit word architecture.

3.0 METHODOLOGY

The VAX-11 FORTRAN is based on the American National Standard FORTRAN 77(ANSI x 3.9-1978). The DEC-2050 and PDP-11 FORTRAN is based on the previous standard (ANSI x 3.9-1966). The UNIVAC FORTRAN has also been written in accordance with the specification of ANSI x 3.9 -1966 FORTRAN and is a superset of it. As a result there are certain incompatibilities listed below:

3.1 Open Statement

An OPEN statement either connects an existing file to a logical unit, or creates a new file and connects it to a logical unit. In addition, it can specify file attributes that control file creation and/or subsequent processing.

3.1.1 In the VAX-11 FORTRAN, the OPEN statement contains the keyword BLANK which controls the interpretation of blanks in the numeric field. BLANK='ZERO' specifies all blanks other than leading blanks to be treated as zeroes. BLANK='NULL'specifies all blanks to be treated as blanks. If the total field is blank it is treated as zero. The default value is 'NULL'.

There is no BLANK keyword in OPEN statement of DEC-2050 FORTRAN and the interpretation of blanks is equivalent to BLANK='ZERO'.

There is no BLANK keyword in the PDP-11 FORTRAN.

There is no OPEN statement in UNIVAC FORTRAN and the unit reference number is linked to the external file name using commands. The interpretation of blanks is equivalent to BLANK='ZERO'.

3.1.2 The STATUS or TYPE keyword in the open statement in VAX-11 FORTRAN specifies the initial status of the file ('OLD','NEW','SCRATCH' or 'UNKNOWN'). The default value is 'UNKNOWN'. There is no STATUS or TYPE keyword in the DEC-20 FORTRAN.

In PDP-11 FORTRAN the keyword TYPE is used, instead of STATUS and it has the same values 'OLD','NEW','SCRATCH' or 'UNKNOWN'.

3.1.3 The file is spcefied by the FILE or NAME keyword in VAX-11 FORTRAN where as in DEC-2050 FORTARN only FILE keyword is used and in PDP-11 FORTRAN only NAME keyword is used

3.1.4 The device on which the file exists or it is to be created is specified by the keyword DEVICE='DSK' in DEC-20 FORTRAN. There is no such keyword in VAX-11 or PDP-11 FORTRANs.

The general form of OPEN statemnets used in various FORTRANs are as follows:

i) VAX-11: OPEN(UNIT=n, FILE='filespec 'STATUS='v')

ii) DEC-20: OPEN(UNIT=n,DEVICE='DSK',FILE='filespec')

iii)PDP-11:OPEN(UNIT=n,NAME='filespec',TYPE='v')

where'v'='OLD' or 'NEW.'

The key words and their values in the OPEN statements of the VAX -11, DEC--2050 and PDP-11 FORTRAN are listed below:

(i) VAX-11

Keyword	Values	Function	Default
ACCESS	'SEQUENTIAL'	Access method	'SEQUENTIAL'
	'DIRECT'		
	'KEYED'		
	'APPEND'		

ASSOCIATEVARIABLE	v	Next direct access record	
BLANK	'NULL' 'ZERO'	Interpretation of blanks	'NULL'
BLOCKSIZE	e	Physical block Size	System default
BUFFERCOUNT	e	Number of I/O Buffers	System default
CARRIAGECONTROL	'FORTRAN' 'LIST' 'NONE'	Print control	'FORTRAN' (formatted) 'NONE' (Unformatted)
DISPOSE or DISP	'KEEP'or'SAVE' 'DELETE' 'PRINT' 'PRINT/DELETE' 'SUBMIT' 'SUBMIT/DELETE'	File disposition at close	'KEEP'
ERR	s	Error transfer label	
EXTENDSIZE	e	File allocation increment	Volume or System default
FORM	'FORMATTED' 'UNFORMATTED'	Format type	Depends on ACCESS keyword
FILE or NAME	c	File name specification	
INITIALSIZE	e	File allocation	

IOSTAT	v	Input/output Status	
KEY	e1:e2(:INTEGER) (:CHARACTER)	Key field definitions	
	e	Direct access record limit	
NOSPANBLOCKS	-	Records do not span blocks	
ORGANIZATION	'SEQUENTIAL' 'RELATIVE' 'INDEXED'	File structure	'SEQUENTIAL'
READONLY	-	Write protection	
RECL or RECORD SIZE	e	Record length	As specified at file creation
RECORDTYPE	'FIXED' 'VARIABLE' 'SEGMENTED'	Record	Depends on ORGANIZATION, ACCESS, and FORM keywords
SHARED	-	File sharing allowed	
STATUS or TYPE	'OLD' 'NEW' 'SCRATCH' 'UNKNOWN'	File status at open	'UNKNOWN'
UNIT	e	Logical unit number	
USEROPEN	p	User program option	

Key: e is a numeric expression

v is an integer variable name

e1 is the first byte position of a key

e2 is the last byte position of a key

p is an external function

s is a statement label

c is a character expression, numberic array name, numeric variable name, or numeric array element name

(ii) DEC-2050

Argument	Values Required
UNIT=	Integer variable or constant
MODE=	Literal constant or variable
DIRECTORY=	Literal or variable
FILESIZE	Integer constant or variable
BUFFERCOUNT	Integer constant or variable
ASSOCIATEVARIABLE	Integer variable
ACCESS	'SEQIN','SEQOUT', 'SEQUINOUT', 'RANDIN', 'RANDOM','APPEND'or variable
FILE=	Literal constant or variable
DIALOG=	Literal or array
BLOCKSIZE	Integer constant or variable
VERSION=	Octal constant or variable
DEVICE=	Literal constant or variable
PROTECTION=	An octal constant or integer variable
DISPOSE=	Literal constant or variable
RECORDSIZE	Integer constant or integer variable

PARITY=	Literal constant or variable
DENSITY=	Literal constant or variable

(iii) PDP-11

Keyword	Function	Values
UNIT	logical unit number	e
NAME	file specification	n
TYPE	file type	'OLD' 'NEW' 'SCRATCH' 'UNKNOWN'
ACCESS	access method	'SEQUENTIAL' 'DIRECT' 'APPEND'
READONLY	read-only file access	
FORM	file format	'FORMATTED' 'UNFORMATTED'
RECORDSIZE	direct access record length	e
ERR	error condition transfer label	s
BUFFERCOUNT	number of buffers	e
INITIALSIZE	file allocation size	e
EXTENDSIZE	file extension increment	e
NOSPANBLOCKS	unspanned records	
SHARED	shared file access	
DISPOSE or DISP	file disposition	'SAVE' 'KEEP' 'PRINT' 'DELETE'

		'DELETE'
ASSOCIATEVARIABLE	associated variable name	v
CARRIAGECONTROL	carriage control type	'FORTRAN' 'LIST' 'NONE'
MAXREC	number of direct access records	e
BLOCKSIZE	physical block size	e

e	is a numeric expression.
n	is a variable name, array name, array element name, or alphanumeric literal.
s	is an executable statement lable.
v	is an integer variable name.

3.2 DO STATEMENT

The DO statement controls interative porcessing. There are two types of DO statements in VAX FORTRAN

1) The indexed DO

2) The pretested indenfinite DO or the DO WHITE statement. The DEC-20, PDP-11 and UNIVAC-1100 support only the indexed DO statements. The general form of indexed DO is:

DO s v= e_1, e_2, e_3

where s is the label of an executable statement.

v is an integer or real variable

e_1, e_2, e_3 are airthmetic expresions

The numbers of executions of Do loop, called the interaction count is given by:

$$(e_2 - e_1 + e_3) / e_3$$

In VAX-11 FORTRAN, if the iteration count of the DO loop is zero or negative, the DO loop is not executed at all.

In the DEC-20, UNIVAC-1100 and PDP-11 FORTRAN, the DO loop is always executed at least once.

In the UNIVAC FORTRAN, the DO loop index can be only an integer variable and the DO loop parameter may be integer expressions. In the PDP-11, DEC-20 and VAX-11 FORTRAN they may be real variable and real expressions.

In VAX-11, PDP-11 and DEC-20 FORTRAN, the DO loop iteration count is calculated at start of DO loop and decremented at each step. When it count = 0, execution of DO loop is terminated.

IN UNIVAC FORTRAN, first the DO loop is executed once, then the DO loop index,'i', is incremented by e3 and if (e2-i)-e3 is negative, execution of DO loope is stopped.

The DO WHILE statement available in VAX FORTRAN has the form DO s WHILE(e) where s is a statement label(optional) and e is a logical expression

Example DO WHILE(I.GT J)

```
        ARRAY(I,J)=10
        I = I-1
        END DO
```

3.3 IF statement

The IF statement conditionally transfer control, or conditionally execute a statement or block of statements.

All the four systems support the arithmetic IF and logical IF statements but the block IF statements are supported by the VAX-11

FORTRAN only. The block IF statements conditionally execute blocks of statements. The four block IF statements are

1) IF THEN

2) ELSE IF THEN

3) ELSE

4) END IF

The block IF construct has the form:

```
IF(C) THEN
       : block
       :
ELSE IF (C) THEN
       block
       :
       :
ELSE
       : block
END IF
```

where e is a logical expression and block is a sequence of complete Fortran statements.

A block IF construct may contain any number of IF THEN ELSE statments.

<u>Example:</u>

```
IF(A.GT.B) THEN
       D=B
       F= A-B
ELSE IF (A.GT.C)THEN
       D=C
       F=A-C
       ELSE IF (A.GT.2)THEN
```

```
        D=2
        F=A - 2
ELSE
        D=0.0
        F=A
END IF
```

3.4 Array dimensions

In VAX-11 and PDP-11 , UNIVAC-1100 FORTRAN, an array can have upto 7 dimensions, whereas in DEC-20 FORTRAN an array can have any number of dimensions.

3.5 Symbolic Names

VAX-11 FORTRAN allows symbolic names upto 31 characters consisting of letters, digits, dollar sign($) and underline(_) but the first character must be a letter. PDP-11 and DEC-20 FORTRAN allow symbolic names of any alphanumeric combination of one to six characters only. If the symbolic name consists of more than 6 characters then first six characters are considered and the remaining are ignored.

3.6 Data Types

VAX-11 FORTRAN and UNIVAC-11 support REAL*16, COMPLEX *16 and CHARACTER data types. There is no CHARACTER data type and double precision COMPLEX in PDP-11 and DEC-20 FORTRAN.

3.6.1 Numeric Data

VAX-11 supports the following data types:

1. Integer *2, Integer *4, Integer *8

2. Real *4

3. Real*8(Double precision)

4. Real *16

5. Complex *8(a pair of Real *4 values)

6. Complex *16(a pair of complex *8 values)

7. Logical

8. Octal and hexadecimal

9. Hollorith.

The DEC-2050 and PDP-11 supports:

1. Integer

2. Real

3. Double precision

4. Complex

5. Logical

6. Literal

7. Octal

8. Hollerith

3.6.2 Character data:

In VAX-11 and UNIVAC-1100 FORTRAN, character data is specified by a CHARACTER declaration statement. There can be character substrings, character operators, character expressions, and character assignment statements. In PDP-11 and DEC-20 FORTRAN there is no such data type. The only character operator is the concatenation operator // which is in VAX and UNIVAC systems.

3.7 EXTERNAL statement

In VAX-11 and UNIVAC-1100 FORTRAN, the EXTERNAL statement specifies that the procedure is a FORTRAN supplied function. There is no INTRINSIC statement in UNIVAC. If a FORTRAN supplied function name appears in an EXTERNAL statement it is treated as a user supplied function.

3.8 Format descriptors

The X format edit descriptor in VAX-11 FORTRAN does not modify the character position skipped and length of output record is not extended. In DEC-20 FORTRAN the X-editor writes blanks and may extend the output record.

The VAX-11 supports the following field descriptions

Integer	- Iw, Zw,Iwm,Zwm
Octal	-Ow, Owm
Logical	-Lw
Real and Complex	- Fw.d,Ew.d,Dw.d,Gw.d,Ew.dEc,Gw.d Ec
Character	-Aw
Character and Hollerith constant	-nH
Edit descriptors	- nX,Tn,TLn,TRn,nP,Q,$,BN,BZ,S,SP,SS.

The DEC-20 supports:

Integer	-Iw
Octal	-Ow
Logical	-Lw
Real and Complex	-Fw.d,Ew.d, Dw.d, Gw.d
Alpha numeric	-Aw,Rw
Hollerith and Literal Constant	-nH, 'text'
Edit descriptors	-nX,Tw

The UNIVAC supports:

Integer	-Iw,Iw.d,Jw.
Octal	-Ow
Logical	-Lw
Real and Complex	-Fw.d, Ew.d, Ew.d Dc, Dw.d, Gw.d
Alpha Numeric	-Aw,Rw
Literal and Hollerith Constant	-nH,'text'
Edit descriptors	-nX,Tw,nP, \$

The PDP-11 supports:

Integer	- Iw
Octal	-Ow
Real and complex	-Fw.d, Ew.d,Dw.d,Gw.d
Literal and Hollerith constant	- nH
Alpha numeric	-Aw
Edit descriptors	-nX,Tw,Q,\$, nP

3.9 In UNIVAC FORTRAN comment lines can begin only with C in column 1. In line comments are preceded by @. In PDP-11, comments lines begin with C or c in column 1. DEC system allows C, \$,1, or * in column 1, while VAX-11 FORTRAN allows C, * or ! in column 1 for lines to be treated as comment lines. In line comments in PDP-11, DEC-20 and VAX-11 and preceded by !

3.10 Variable and Run Time formats are allowed on VAX, PDP and UNIVAC but not on DEC-20.

3.11 In UNIVAC FORTRAN, the DIMENSION and TYPE declarator statements allow initialisation of variables. This is not allowed on any of the other systems.

3.12 Intrinsic functions available on various systems are listed in Appendix-A.

3.13 Library subroutines available on various system are explained in APPENDIX-B.

4.0 CONCLUSION

FORTRAN language differs from computer to computer in only minor respects. The report has illustrated a comparative study of the FORTRAN language using the facilities of VAX-11, DEC 2050, PDP-11 and UNIVAC-1100. The compatability between VAX-11 FORTRAN and other systems FORTRAN will be of great use to the computer programmers.

REFERENCES

1. DEC-2050 FORTRAN Language reference Mannual

2. Davis and Hoffmann, FORTRAN 77: A Structured, Disciplined Style, International Student Edition.

3. Maynerd, J.Computer Programming made simple Twentieth Century Publications, New Delhi.

4. PDP-11 FORTRAN Language Reference Mannual, Digital Equipment Corporation, USA.

5. VAX-11 FORTRAN Language Reference Mannual, Digital Equipment Corporation, USA.

(i) VAX-11 **Intrinsic Functions**

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Square Root	1	SQRT	SQRT	REAL*4	REAL*4
			DSQRT	REAL*8	REAL*8
			QSQRT	REAL*16	REAL*16
$a^{1/2}$			CSQRT	COMPLEX*8	COMPLEX*8
			CDSQRT	COMPLEX*16	COMPLEX*16
Natural Logarithm	1	LOG	ALOG	REAL*4	REAL*4
			DLOG	REAL*8	REAL*8
			QLOG	REAL*16	REAL*16
$\log_e a$			CLOG	COMPLEX*8	COMPLEX*8
			CDLOG	COMPLEX*16	COMPLEX*16
Common Logarithm	1	LOG10	ALOG10	REAL*4	REAL*4
			DLOG10	REAL*8	REAL*8
$\log_{10} a$			QLOG10	REAL*16	REAL*16
Exponential	1	EXP	EXP	REAL*4	REAL*4
e^a			DEXP	REAL*8	REAL*8
			QEXP	REAL*16	REAL*16
			CEXP	COMPLEX*8	COMPLEX*8
			CDEXP	COMPLEX*16	COMPLEX*16
Sine	1	SIN	SIN	REAL*4	REAL*4
			DSIN	REAL*8	REAL*8
			QSIN	REAL*16	REAL*16
Sin a			CSIN	COMPLEX*8	COMPLEX*8
			CDSIN	COMPLEX*16	COMPLEX*16
Cosine	1	COS	COS	REAL*4	REAL*4
			DCOS	REAL*8	REAL*8
			QCOS	REAL*16	REAL*16
Cos a			CCOS	COMPLEX*8	COMPLEX*8
			CDCOS	COMPLEX*16	COMPLEX*16
Tangent	1	TAN	TAN	REAL*4	REAL*4
			DTAN	REAL*8	REAL*8
Tan a			QTAN	REAL*16	REAL*16
Arc Sine	1	ASIN	ASIN	REAL*4	REAL*4
			DASIN	REAL*8	REAL*8
Arc Sin a			QASIN	REAL*16	REAL*16
Arc Cosine	1	ACOS	ACOS	REAL*4	REAL*4
			DACOS	REAL*8	REAL*8
Arc Cos a			QACOS	REAL*16	REAL*16

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Arc Tangent	1	ATAN	ATAN	REAL*4	REAL*4
			DATAN	REAL*8	REAL*8
Arc Tan a			QATAN	REAL*16	REAL*16
Arc Tangent	2	ATAN2	ATAN2	REAL*4	REAL*4
			DATAN2	REAL*8	REAL*8
Arc Tan a_1/a_2			QATAN2	REAL*16	REAL*16
Hyperbolic Sine	1	SINH	SINH	REAL*4	REAL*4
			DSINH	REAL*8	REAL*8
Sinh a			QSINH	REAL*16	REAL*16
Hyperbolic Cosine	1	COSH	COSH	REAL*4	REAL*4
			DCOSH	REAL*8	REAL*8
Cosh a			QCOSH	REAL*16	REAL*16
Hyperbolic Tangent	1	TANH	TANH	REAL*4	REAL*4
			DTANH	REAL*8	REAL*8
Tanh a			QTANH	REAL*16	REAL*16
Absolute value	1	ABS	HABS	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	HABS	INTEGER*2	INTEGER*2
Truncation	1	INT	HNT	REAL*4	INTEGER*2
$\|a\|$			JINT	REAL*4	INTEGER*4
			HDINT	REAL*8	INTEGER*2
			JIDINT	REAL*8	INTEGER*4
			HQINT	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	HDINT	REAL*8	INTEGER*2
			JIDINT	REAL*8	INTEGER*4
		IQINT	HQINT	REAL*16	INTEGER*2
			JIQINT	REAL*16	INTEGER*4

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
		AINT	AINT	REAL*4	REAL*4
			DINT	REAL*8	REAL*8
			QINT	REAL*16	REAL*16
Nearest Integer	1	NINT	ININT	REAL*4	INTEGER*2
			JNINT	REAL*4	INTEGER*4
			HDNNT	REAL*8	INTEGER*2
			JIDNNT	REAL*8	INTEGER*4
			HQNNT	REAL*16	INTEGER*2
			JIQNNT	REAL*16	INTEGER*4
		IDNINT	HDNNT	REAL*8	INTEGER*2
			JIDNNT	REAL*8	INTEGER*4
		IQNINT	HQNNT	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
Conversion to 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
Conversion to 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 (REAL*4-to-integer conversion)	1	IFIX	HFIX	REAL*4	INTEGER*2
			JIFIX	REAL*4	INTETER*4

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Float	1	FLOAT	FLOATI FLOATJ	INTEGER*2 INTEGER*4	REAL*4 REAL*4
(integer-to-REAL*4 conversion)					
REAL*8 Float	1	DFLOAT	DFLOTI DFLOTJ	INTEGER*2 INTEGER*4	REAL*8 REAL*8
(integer-to-REAL*8 conversion					
REAL*16 Float	1	QFLOAT	-	INTEGER*2 INTEGER*4	REAL*16 REAL*16
(Interger to REAL*16 conversion)					
Conversion to COMPLEX*8 or COMPLEX*8 from Two Arguments	1,2 1,2 1,2 1,2 1,2 1 1	CMPLX	- - - - - - -	INTEGER*2 INTEGER*4 REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16	COMPLEX*8 COMPLEX*8 COMPLEX*8 COMPLEX*8 COMPLEX*8 COMPLEX*8 COMPLEX*8
Conversion to COMPLEX*16 or COMPLEX*16 from Two Arguments	1,2 1,2 1,2 1,2 1,2 1 1	DCMPLX	- - - - - - - -	INTEGER*2 INTEGER*4 REAL*4 REAL*8 REAL*16 COMPLEX*8 COMPLEX*16	COMPLEX*16 COMPLEX*16 COMPLEX*16 COMPLEX*16 COMPLEX*16 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

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Maximum	n	MAX	IMAXO	INTEGER*2	INTEGER*2
$max(a_1,a_2...a_n)$			JMAXO	INTEGER*4	INTEGER*4
(retuns the maximum			AMAXI	REAL*4	REAL*4
value from among the			DMAXI	REAL*8	REAL*8
argument list, there must be at least two arguments)			QMAXI	REAL*16	REAL*16
		MAXO	IMAXO	INTEGER*2	INTEGER*2
			JMAXO	INTEGER*4	INTEGER*4
		MAXI	IMAXI	REAL*4	INTEGER*2
			JMAXI	REAL*4	INTEGER*4
		AMAXO	AIMAXO	INTEGER*2	REAL*4
			AJMAXO	INTEGER*4	REAL*4
Minimum	n	MIN	IMINO	INTEGER*2	INTEGER*2
$min(a_1,a_2...a_n)$			JMINO	INTEGER*4	INTEGER*4
(returns the minimum value			AMINI	REAL*4	REAL*4
among the argument list,			DMINI	REAL*8	REAL*8
there must be at least two			QMINI	REAL*16	REAL*16
arguments)		MINO	IMINO	INTEGER*2	INTEGER*2
			JMINO	INTEGER*4	INTEGER*4
		MINI	IMINI	REAL*4	INTEGER*2
			JMINI	REAL*4	INTEGER*4
		AMINO	AIMINO	INTEGER*2	REAL*4
			AJMINO	INTEGER*4	REAL*4
Positive Difference	2	DIM	HDIM	INTEGER*2	INTEGER*
			JIDIM	INTEGER*4	INTEGER*4
			DIM	REAL*4	REAL*4
$a_1-(min(a_1,a_2))$			DDIM	REAL*8	REAL*8
(returns the first argument minus the minimum			QDIM	REAL*16	REAL*16
of the two arguments)		IDIM	HDIM	INTEGER*2	INTEGER*2
			JIDIM	INTEGER*4	INTEGER*4
Remainder	2	MOD	IMOD	INTEGER*2	INTEGER*2
			JMOD	INTEGER*4	INTEGER*4
a_1,a_2 (a_1/a_2)			AMOD	REAL*4	REAL*4
returns the remainder			DMOD	REAL*8	REAL*8
when the first argument			QMOD	REAL*16	REAL*16

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
Transfer or Sign $\|a_1\|$*Sign a_2	2	SIGN	IISIGN JISIGN SIGN DSIGN QSIGN	INTEGER*2 INTEGER*4 REAL*4 REAL*8 REAL*16	INTEGER*2 INTEGER*4 REAL*4 REAL*8 REAL*16
		ISIGN	IISIGN JISIGN	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bitwise AND (performs a logical AND on corresponding bits)	2	IAND	IIAND JIAND	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
Bitwise OR (performs an inclusive OR on corresponding bits)	2	IOR	IIOR JIOR	INTEGER*2 INTEGER*4	INTEGER*2 INTEGER*4
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
Length (returns length of the character expression)	1	–	LEN	CHARACTER	INTEGER*4
Index (C_1,C_2) (returns the position of the substring c_2 in the character expression c_1)	2	–	INDEX	CHARACTER	INTEGER*4
Character (returns a character that has the ASCII value specified by the argument)	1	–	CHAR	LOGICAL*1 INTEGER*2 INTEGER*4	CHARACTER

Functions	Number of Arguments	Generic Name	Specific Name	Type of Argument	Type of Result
ASCII Value (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

(ii) DEC - 20 **Intrinsic Functions**

Function	Mnemonic	Definition	Number of Arguments	Type of Argument	Type of Function
Absolute value:					
Real	ABS*	arg	1	Real	Real
Integer	IABS*	arg	1	Integer	Integer
Double precision	DABS*	arg	1	Double	Double
Complex to real	CABS	$c=(x^2+y^2)^{1/2}$	1	Complex	Real
Conversion:					
Integer to real	FLOAT*	Sign of arg*	1	Integer	Real
Real to integer	IFIX*	largest integer $\leq \|arg\|$			
Double to real	SNGL		1	Double	Real
Real to double	DBLE*		1	Real	Double
Integer to double	DFLOAT		1	Integer	Double
Complex to real (obtain real part)	REAL*		1	Complex	Real
Complex to real (obtain imaginary part)	AIMAG		1	Complex	Real
Real to complex	CMPLX*	$c=Arg_1+i*Arg_2$	2	Real	Complex
Truncation:					
Real to real	AINT	Sign of arg* largest integer $\leq \|arg\|$	1	Real	Real
			1	Real	Integer
Real to integer	INT*		1	Double	Integer
Remaindering:					
Real	AMOD	The remainder when Arg 1 is divided by Arg 2	2	Real	Real
Integer	MOD*		2	Integer	Integer
Double precision	DMOD		2	Double	Double
Maximum value:					
	AMAX0	$Max(Arg_1, Arg_2, \ldots)$	≥ 2	Integer	Real
	AMAX1*		≥ 2	Real	Real
	MAX0*		≥ 2	Integer	Integer
	MAX1		≥ 2	Real	Integer
	DMAX1		≥ 2	Double	Double

Function	Mnemonic	Definition	Number of Arguments	Type of Argument	Type of Function
Minimum Value:					
	AMINO		≥2	Integer	Real
	AMINI*		≥2	Real	Real
	MINO*	$Min(Arg_1, Arg_2, \ldots)$	≥2	Integer	Integer
	MINI		≥2	Real	Integer
	DMIN1		≥2	Double	Double
Transfer of Sign:					
Real	SIGN*		2	Real	Real
Integer	ISIGN	$Sgn(Arg_2)^* \|Arg_1\|$	2	Integer	Integer
Double precision	DSIGN		2	Double	Double
Positive Difference:					
Real	DIM*	$Arg_1 - Min(Arg_1, Arg_2)$	2	Real	Real
Integer	IDIM		2	Integer	Integer
Exponential:					
Real	EXP	e^{Arg}	1	Real	Real
Double	DEXP		1	Double	Double
Complex	CEXP		1	Complex	Complex
Logarithm					
Real	ALOG	$\log_e(Arg)$	1	Real	Real
	ALOG10	$\log_{10}(Arg)$	1	Real	Real
Double	DLOG	$\log_e(Arg)$	1	Double	Double
	DLOG10	$\log_{10}(Arg)$	1	Double	Double
Complex	CLOG	$\log_e(Arg)$	1	Complex	Complex
Square Root:					
Real	SQRT*	$(Arg)^{1/2}$	1	Real	Real
Double	DSQRT	$(Arg)^{1/2}$	1	Double	Double
Complex	CSQRT	$(Arg)^{1/2}$	1	Complex	Complex
Sine:					
Real(radians)	SIN*		1	Real	Real
Real(degrees)	SIND		1	Real	Real
Double(radians)	DSIN	sin(Arg)	1	Double	Double
Complex	CSIN		1	Complex	Complex
Cosine:					
Real(radians)	COS*		1	Real	Real
Real(degrees)	COSD		1	Real	Real
Double(radians)	DCOS	cos(Arg)	1	Double	Double
Complex	CCOS		1	Complex	Complex

Function	Mnemonic	Definition	Number of Arguments	Type of Argument	Type of Function
Hyperbolic:					
Sine	SINH	sinh(Arg)	1	Real	Real
Cosine	COSH	cosh(Arg)	1	Real	Real
Tangent	TANH	tanh(Arg)	1	Real	Real
Arc sine	ASIN	asin(Arg)	1	Real	Real
Arc cosine	ACOS	acos(Arg)	1	Real	Real
Arc tangent					
Real	ATAN*	atan(Arg)	1	Real	Real
Double	DATAN	datan(Arg)	1	Double	Double
Two REAL arguments	ATAN2*	atan(Arg_1/Arg_2)	2	Real	Real
Two DOUBLE arguments	DATAN2	atan(Arg_1/Arg_2)	2	Double	Double
Complex Conjugate	CONJG	Arg=X+iY, CONJG=X-iY	1	Complex	Complex
Random Number	RAN	Result is a random number in the range of 0 to 1.0.	1 Dummy Argument	Integer, Real, Double, or Complex	Real

(iii) PDP-11 **Intrinsic Functions**

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
ABS(X)	Real absolute value	Real	Real
IABS(I)	Integer absolute value	Integer	Integer
DABS(X)	Double precision absolute value	Double	Double
CABS(Z)	Complex to Real absolute value where Z=(x,y) $CABS(Z)=(x^2+y^2)^{1/2}$	Complex	Real
FLOAT(I)	Integer to Real conversion	Integer	Real
IFIX(X)	Real to Integer conversion IFIX(X) is equivalent to INT(X)	Real	Integer
SNGL(X)	Double to Real conversion	Double	Real
DBLE(X)	Real to Double conversion	Real	Double
REAL(Z)	Complex to Real conversion, obtain real part	Complex	Real
AIMAG(Z)	Complex to Real conversion, obtain imaginary part	Complex	Real
CMPLX(X,Y)	Real to Complex conversion CMPLX(X,Y)=X+i*Y	Real	Complex
	Truncation functions return the sign of the argument * largest integer ≤ |arg|		
AINT(X)	Real to Real truncation	Real	Real
INT(X)	Real to Integer truncation	Real	Integer
IDINT(X)	Double to Integer truncation	Double	Integer
	Remainder functions return the remainder when the first argument is divided by the second		
AMOD(X,Y)	Real remainder	Real	Real
MOD(I,J)	Integer remainder	Integer	Integer
DMOD(X,Y)	Double precision remainder	Double	Double
	Maximum value functions return the largest value from among the argument list, ≥2 arguments.		
AMAXØ(I,J,...)	Real maximum from Integer list	Integer	Real
AMAX1(X,Y,...)	Real maximum of Real list	Real	Real
MAXØ(I,J,...)	Integer maximum of Integer list	Integer	Integer
MAX1(X,Y,...)	Integer maximum of Real list	Real	Integer
DMAX1(X,Y,...)	Double maximum of Double list	Double	Double

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
	Minimum value functions return the smallest value from among the argument list, ≥2 arguments.		
AMINØ(I,J,...)	Real minimum of Integer list	Integer	Real
AMIN1(X,Y,...)	Real minimum of Real list	Real	Real
MINØ(I,J,...)	Integer minimum of Integer list	Integer	Integer
MIN1(X,Y,...)	Integer minimum of Real list	Real	Integer
DMIN1(X,Y,...)	Double minimum of Double list	Double	Double
	The transfer of sign functions return (sign of the second argument) * (absolute value of first argument).		
SIGN(X,Y)	Real transfer of sign	Real	Real
ISIGN(I,J)	Integer transfer of sign	Integer	Integer
DSIGN(X,Y)	Double precision transfer of sign	Double	Double
	Positive difference functions return the first argument minus the minimum of the two arguments.		
DIM(X,Y)	Real positive difference	Real	Real
IDIM(I,J)	Integer positive difference	Integer	Integer
	Exponential functions return the value of e raised to argument power.		
EXP(X)	e^X	Real	Real
DEXP(X)	e^X	Double	Double
CEXP(Z)	e^Z	Complex	Complex
ALOG(X)	Returns $\log_e(X)$	Real	Real
ALOG1Ø(X)	Returns $\log_{10}(X)$	Real	Real
DLOG(X)	Returns $\log_e(X)$	Double	Double
DLOG1Ø(X)	Returns $\log_{10}(X)$	Double	Double
CLOG(Z)	Returns $\log_e$ of complex argument	Complex	Complex
SQRT(X)	Square root of Real argument	Real	Real
DSQRT(X)	Square root of Double precision argument	Double	Double
CSQRT(Z)	Square root of complex argument	Complex	Complex

FORM	DEFINITION	ARGUMENT TYPE	RESULT TYPE
COS(X) DCOS(X) CCOS(Z)	Real cosine Double precision cosine Complex cosine	Real Double Complex	Real Double Complex
TANH(X)	Hyperbolic tangent	Real	Real
ATAN(X) DATAN(X) ATAN2(X,Y) DATAN2(X,Y)	Real arc tangent Double precision arc tangent Real arc tangent of (X/Y) Double precision arc tangent of (X/Y)	Real Double Real Double	Real Double Real Double
CONJG(Z)	Complex conjugate, if Z=X+i*Y	Complex	Complex
RAN(I,J)	Returns a random number of uniform distribution over the range Ø to 1. I and J must be integer variables and should be set initially to Ø regenerates the random number sequence. Alternate starting values for I and J will generate different random number sequences. See also Appendix C.3.	Integer	Real

(i) VAX-11 Library Subroutines

(A) DATE

The DATE subroutines obtains the current date as set within the system. The call to DATE has the form

CALL DATE(buf)

where:

buf

is a 9-byte variable, array, array element, or character substring. The ing. The date is returned as a 9-byte ASCII character string of the form.

dd-mm-yy

where:

dd

is the 2-digit date.

mmm

is the 3-letter month specification.

yy

is the last two digits of the year.

(B) IDATE

The IDATE subroutine returns three integer values representing the current month, day, and year. The call to IDATE has the form

CALL IDATE(i,j,k)

If the current date were October 9, 1984, the values of the integer variables upon return would be:

i = 10

j = 9

j = 84

(C) ERRSNS

The ERRSNS subroutine returns information about the most recent error that has occurred during program execution.The cal to ERRSNS has the form

CALL ERRSNS(fnum,rmssts,rmsstv,iunit,condval)

where:

fnum

is an integer variable or array element into which is stored the most recent FORTRAN error number.

A zero is returned if no error has occurred since the last call to ERRSNS, or if no error has occurred since the start of execution.

rmssts

is an integer variable or array element into which is stored the RMS completion status code(STS),if the last error was an RMS I/O error

rmmstv

is an integer variable or array element into which is stored the logical unit number, if the last error was an I/O error.

iunit

is an integer variable or array element into which is stored the logical unit number,if the last error was an I/O error.

condval

is an integer variable or array element in which is stored the actual VAX-11 condition value.

Any of the arguments may be null. If the arguments are of INTEGER*2 type, only the low-order 16 bits of information are returned. The saved error information is set to zero after each call to ERRSNS

(D) EXIT

The EXIT subroutine causes program termination, closes all files, and returns control to the operation system. A call to EXIT has the form

CALL EXIT[(exit-status)]

where:

(exit-status)

is an optional integer argument which can be used to specify the image exit-status value.

(E) SECNDS

The SECNDS function subprogram returns the system time in seconds as a single-precision, floating -point value less the value of its single-precision, floating-point argument. The call to SECNDS has the form

```
y= SECNDS(x)
```

where:

y

is set equal to the time in seconds since midnight, minus the user-supplied value of x.

The SECNDS function can be used to perform elapsed-time computations. For example:

```
C  START OF TIMED SEQUENCE
   TI= SECNDS(0.0)
C
C  CODE TO BE TIMED
c
    DELTA = SECNDS(T1)
```

where:

DELTA

will give the elapsed time.

The voalue of SECNDS is accurate to 0.01 seconds, which is the resolu.-tion of the system clock.

NOTES

1. The time is computed from midnight.

SECNDS also produces correct results for time intervals that span midnight.

2. The 24 bits of precision provides accuracy to the resolution of the system clock for about one day. However, loss of significance can occur if an attempt is made to compute very small elapsed times late in the day. More precise timing information can be obtained using Run Time Library procedures:

```
LIB$INIT_TIMER
LIB$SHOW_TIMER
LIB$STAT_TIMER
```

(F) TIME

The TIME subroutine returns the current system time as an ASCII string. The call to TIME has the form

```
CALL TIME(buf)
```

where buf is an 8-byte variable, array, array element, or character substring.

The TIME call returns the time as an 8-byte ASCII character string of the form

hh: mm: ss

where:

hh

is the 2-digit hour indication.

mm

is the 2-digit minute indication.

ss

is the 2-digit second indication.

For example:

10:45:23

A 24 hour clock is used.

(G) RAN

The RAN function is a general random number generator of the multiplicative congruential type. The result is a floating-point number that is uniformly distributed in the range between 0.0 (inclusive) and 1.0 (exclusive). The call to RAN has the form:

y= RAN(i)

where:

y

is set equal to the value associated, by the function, with the argument i. The argument i must be an INTEGER*4 variable or INTEGER*4 array element.

The argument should initially be set to a large, odd integer value. The RAN function stores a value in the argument that it later uses to calculate the next random number.

There are no restrictions on the seed, although it should be initialized with different values on separate runs in order to obtain different random numbers, the seed is updated automatically. RAN uses the following algorithm to update the seed passed as the parameter:

SEED = 69069 * SEED + 1 (MOD 2**32)

The value of SEED is a 32-bit number whose high-order 24 bits are converted to floating point and returned as the result.

(ii) DEC-20 Library Subroutines

(A) DATE

Places today's date as left-justified ASCII characters into a dimensioned 2-word array.

CALL DATE(array)

where array is the 2-word array. The date is in the form dd-mm-yy where dd is a 2-digit day(if the first digit is 0, it is converted to a blank), mmm is a 3-letter month(e.g.Mar.) and yy is a 2-digit year. The data is stored in ASCII code, left-justified, in the two words.

(B) DEFINEFILE

A DEFINEFILE call can be used to establish and define the structure of each file to be used for random access I/O operations. The format of a DEFINEFILE may be

CALL DEFINEFILE (u,s,v,f,proj,prog)

where

u= logical FORTRAN device numbers.

s = the size of the records which comprise the file being defined. The argument s may be an integer constant or variable.

v= an associated variable. The associated varilable is an integer variable that is set to a value that points to the record that immediately follows the last record transferred. This variable is used by the FIND statement.

At the end of each FIND operation the variable is set to a value that points to the record found. The variable v can not appear in the I/O list of any I/O statement that accesses the file set up by the DEFINEFILE statement

f = file name to be given to the file being defined.

proj= user's project number.

prog= user's programmer number

Example

The statement

CALL DEFINEFILE (1,10,ASCAR,'FORTEL.DAT',0.0)

Establishes a file named FORTEL.DAT on device 01(i.e.,disk) which contains word records. The associated variable is ASCVAR, and the file is in the user's area.

(c) DUMP

Causes particular portions of core to be dumped and is referred to in the form:

CALL DUMP (L_1,U_1,F_1,....,L_n,U_n,F_n)

where L_1 and U_1 are the variable names which give the limits of core memory to be dumped. Either L_1 or U_1 may be upper or lower limits. F_1 is a number indicating the format in which the dump is to be performed: 0=octal, 1=real,2=integer, and 3=ASCII.

If F is not 0,1,2,3, the dump is in octal. If F_n is missing, the last section is dumped in octal. If U_n and F_n are missing, an octal dump is made from L to the end of the job area. If L_n, U_n, and F_n are missing, the entire job area is dumped in octal.

The dump is terminated by a call to EXIT.

(D) ERRSET

Allows the user to control the typeout of execution-time arithmetic error messages, ERRSET is called with one argument in integer mode.

CALL ERRSET(N)

Typeout of each type of error message is suppressed after N occurrences of that error message. If ERRSET is not called, default value of N is 2.

(E) EXIT

Returns control to the Monitor and, therefore, terminates the execution of the program.

(F) ILL

Sets the ILLEG flag. If the flag is set and an illegal character is encountered in floating point/double precision input, the corresponding word is set to zero.

CALL ILL

(G) LEGAL

Clears the ILLEG flag. If the flag is set and an illegal character is encountered in the floating point/double precision input, the corresponding word is set to zero.

CALL LEGAL

(H) PDUMP

CALL PDUMP(L_1,U_1,F_1,....,L_n,U_n,F_n)

The arguments are the same as those for DUMP.PDUMP is the same as DUMP except that control returns to the calling program after the dump has been executed.

(I) RELEAS

CALL RELEAS(unit)

Closes out I/O on a device initialized by the FORTRAN Compiler returns it to the uninitialized state.

(J) SAVRAN

SAVRAN is called with one argument in integer mode. SAVRAN sets its argument to the last random number(interpreted as an integer) that has been generated by the function RAN.

(K) SETABL

CALL SETABL(I,J)

Specifies a character set where I is an integer which gives the

the number of the desired character set. If a character set has been defined by I, the value of J is set to 0; if not, J is set to -1. The desired ASCII character set is defined as 1.

(L) SETRAN

SETRAN has one argument which must be a non-negative integer 2^{31}. The starting value of the function RAN is set to one value of this argument, unless the argument is zero. In this case, RAN uses its normal starting value.

(M) TIME

Returns the current time in its argument(s) in left-justified ASCII characters. If TIME is called with one argument.

CALL TIME(X)

the time is in the form

hh:mm

where hh is the hours(24 hour time) and mm is the minutes.

If a second argument is requested,

CALL TIME(X,Y)

the first argument is returned as before and the second has the form

bss.t

where ss is the seconds, t is the tenths of a second, and b is a blank.