NASA CONTRACTOR REPORT 166574

CDC to CRAY FORTRAN Conversion Manual

Charlotte McGary
Donna Diebert

CONTRACT NAS2-11555

October, 1983

NASA

NASA CONTRACTOR REPORT 166574

CDC to CRAY FORTRAN
Conversion Manual
TN-83-7104-802-1

Charlotte McGary
Donna Diebert
Informatics General Corporation
1121 San Antonio Road
Palo Alto, CA 94303

Prepared for
Ames Research Center under
CONTRACT NAS2-11555

October, 1983

NASA

TABLE OF CONTENTS

0.0 SCOPE

This document is intended to be an aid in converting source code from FORTRAN IV Extended as used on the CDC 7600 under the SCOPE operating system to FORTRAN 77 as used on the CRAY-1S/CRAY-XMP under COS. Although there are many similarities between CDC and CRAY FORTRAN, there are several differences which can cause significant problems for the user who is not experienced in the use of both. This document discusses many important differences between these two versions of FORTRAN, and gives examples concerning their usage on both computers.

The author would like to acknowledge the following people who contributed to this document: Donna Diebert, Dave Saunders, and Gary Villere, all with Informatics General Corporation.

This work was performed under contract number NAS2-11555, Ames Research Center, by Informatics General Corporation.

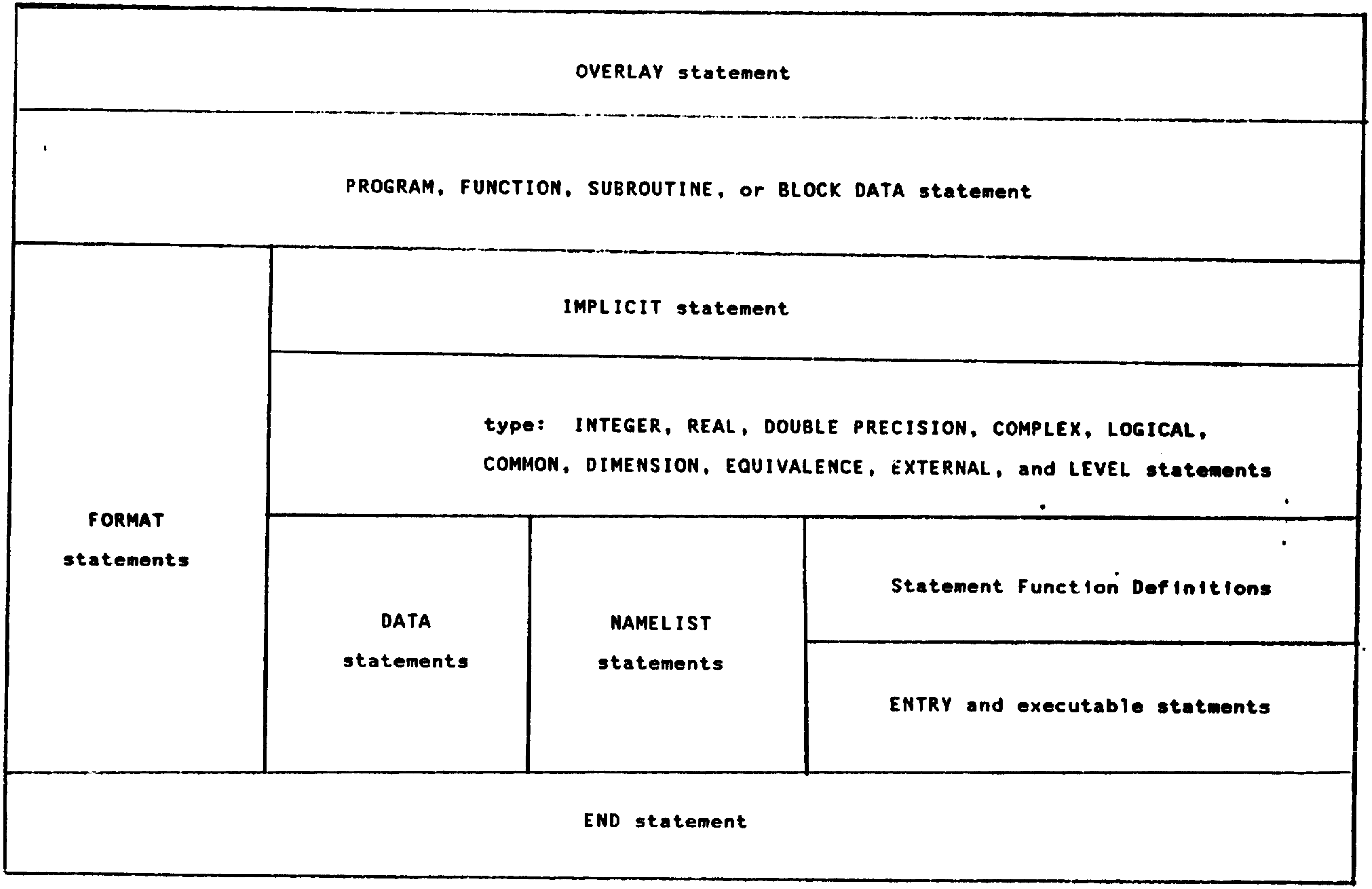
Section 1
GENERAL FORTRAN DIFFERENCES BETWEEN THE CDC 7600 AND THE CRAY

1.0 GENERAL FORTRAN DIFFERENCES BETWEEN THE CDC 7600 AND THE CRAY

While there are many similarities between CDC and CRAY FORTRAN, there are significant differences due to the fact that the CDC 7600 uses FORTRAN IV Extended and the CRAY uses FORTRAN 77. This section will deal with some basic differences between the two.

Diagrams of the general ordering of statements in CDC and CRAY program units are shown in figures 1-1 and 1-2, respectively.

CDC	CRAY
60-bit words	64-bit words
10 characters per word (max)	8 characters per word (max)
Variable names: 1-7 letters	Variable names: 1-8 letters (ANSI FORTRAN standard provides for a maximum of 6 letters)
Maximum of 3 dimensions for an array	Maximum of 7 dimensions for an array
Max. array size = 131,071 words	Max. array size = 4,194,304 words
RANGE OF CONSTANTS:	RANGE OF CONSTANTS:
Integer: $-(2^{59} - 1) <= I <= 2^{59} - 1$ 18 decimal digits	Integer: $-2^{63} <= I < 2^{63}$ 19 decimal digits
Real: $10^{-293} <= \|R\| <= 10^{322}$ 14 decimal digits of precision	Real: $10^{-2466} <= \|R\| < 10^{2466}$ 14 decimal digits of precision
Complex: $10^{-293} <= \|C_{real}\| <= 10^{322}$	Complex: $10^{-2466} <= \|C_{real}\|, \|C_{imag}\| < 10^{2466}$
Hollerith data	Hollerith data supported but CHARACTER data preferred
No block IF structure is available	Block IF structure is available
Only one IMPLICIT statement allowed per program unit	No limit imposed on the number of IMPLICIT statements allowed per program unit



OVERLAY statement

PROGRAM, FUNCTION, SUBROUTINE, or BLOCK DATA statement

FORMAT statements

IMPLICIT statement

type: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, COMMON, DIMENSION, EQUIVALENCE, EXTERNAL, and LEVEL statements

DATA statements

NAMELIST statements

Statement Function Definitions

ENTRY and executable statments

END statement

Figure 1-1 Order of Statements in a CDC Program Unit

<table>
<tr><td colspan="4">PROGRAM, FUNCTION, SUBROUTINE, or BLOCK DATA statement</td></tr>
<tr><td rowspan="4">ENTRY
and
FORMAT
statements</td><td colspan="3">IMPLICIT statement</td></tr>
<tr><td colspan="3">type: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER,LOGICAL, COMMON, DIMENSION, EQUIVALENCE, INTRINSIC, POINTER and SAVE statements</td></tr>
<tr><td rowspan="2">DATA
statements</td><td rowspan="2">NAMELIST
statements</td><td>Statement Function Definitions</td></tr>
<tr><td>Executable statments</td></tr>
<tr><td colspan="4">END statement</td></tr>
</table>

Figure 1-2 Order of Statements in a CRAY Program Unit

Section 2

SPECIFIC FORTRAN DIFFERENCES

2.0 SPECIFIC FORTRAN DIFFERENCES

This section discusses CDC and CRAY differences in the usage of intrinsic functions, program units, arrays, LEVEL and MOVLEV statements, ENTRY statements, and DO loops.

2.1 INTRINSIC FUNCTIONS

Most of the standard intrinsic functions are identical on both the CDC and the CRAY. The following functions are the exception.

CDC	No. Arg.	Notes/ Limitations	CRAY	No. Arg.	Notes/ Limitations
SHIFT	2	$0<=Arg_2<=60$	SHIFT	2	None
		$Arg_2<0$	SHIFTL	2	Zero filled
		$(0<=\lvert Arg_2\rvert<=60$ for both conditions)			No equivalent right shift is available.
MASK	1	0<=Arg<=60 (Left justified)	MASK	1	0<=Arg<=63 (Left justified)
					64<=Arg<=128 (Right justified)
LOCF	1	None	LOC	1	None
RANF	1	None	RANF	0	None
RANGET+	1	None	RANGET	0	None

+ This is a utility subprogram rather than an intrinsic function. It is included here because it corresponds to an intrinsic function.

2.2 PROGRAM UNITS

2.2.1 PROGRAM statement

Although the PROGRAM statement used on the CDC 7600 may be used on the CRAY, the parameters specified after the program name are ignored on the CRAY. The following examples show two CDC uses of the PROGRAM statement and the CRAY equivalent for each.

Example 2-1. Associate INPUT and OUTPUT with logical unit numbers 5 and 6, respectively.

CDC usage:

```
PROGRAM XAMPL1(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
```

CRAY usage:

```
PROGRAM XAMPL1
```

or

```
PROGRAM XAMPL1(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
```

(Normally the user would assign filenames to logical unit numbers in the CCL. However, the CRAY has been set up at AMES to automatically assign logical unit numbers 5 and 6 to $IN and $OUT, respectively.)

Example 2-2. Associate the local file MYFILE with logical unit number 1, which will be used as an input file to the program.

CDC usage: Place the following command in the JCL before the program is executed.

```
ATTACH,MYFILE,MYFILE,ID=XXX,CY=1.
```

The FORTRAN PROGRAM statement will be as follows:

```
PROGRAM XAMPL2(INPUT,OUTPUT,TAPE1=MYFILE)
```

CRAY usage: Place the following commands in the CCL before the program is executed.

```
ACCESS,DN=MYFILE,PDN=MYFILE,ID=XXX,ED=1.
ASSIGN,DN=MYFILE,A=FT01.
```

The FORTRAN PROGRAM statement will follow the same format as in example 2-1.

```
PROGRAM XAMPL2
     or
PROGRAM XAMPL2(INPUT,OUTPUT,TAPE1=MYFILE)
```

2.2.2 Statement Functions

The STATEMENT FUNCTION is used in basically the same manner and subject to most of the same restrictions on both the CDC 7600 and the CRAY. However, two important differences have been noted and are described below.

STATEMENT FUNCTION NAMES AND EXPRESSIONS

CDC usage

STATEMENT FUNCTION NAMES AND EXPRESSIONS MUST BE OF THE SAME TYPE.
If a statement function is declared as one type function but is defined with a parameter of a different type, conversion is automatically performed when the function is evaluated. Therefore, the function may be referenced with a different type of variable than that with which it was defined.

CRAY usage

STATEMENT FUNCTION NAMES AND EXPRESSIONS CAN BE OF DIFFERENT TYPES.
Since a statement function name and expression can differ in type, an error occurs if the function is referenced with a different type variable than that with which it was defined. Therefore, the function and variable with which it is referenced must agree in type.

Example 2-3a.

A statement function is declared as INTEGER:

```
   INTEGER FCN
```

Statement Function Definition:

```
   FCN(R) = R + I * 2
```

Statement Function Reference:

```
   NUMBER = FCN(N) + 1
```

Example 2-3b.

A statement function is declared as INTEGER:

```
   INTEGER FCN
```

Statement Function Definition:

```
   FCN(J) = J + I * 2
```

Statement Function Reference:

```
   NUMBER = FCN(N) + 1
```

STATEMENT FUNCTION DEFINITIONS

CDC usage

A statement function definition may reference another statement function even if that function has not previously been defined.

Example 2-4a.

Statement Function Definitions:

```
   MYFCN1(I) = I * MYFCN2(I)
   MYFCN2(I) = I + 1
```

CRAY usage

A statement function definition may reference another statement function only if the statement function being referenced has been previously been defined.

Example 2-4b.

Statement Function Definitions:

```
   MYFCN2(I) = I + 1
   MYFCN1(I) = I * MYFCN2(I)
```

2.3 ARRAYS

2.3.1 Dimension Declarators

ACTUAL ARRAYS:

On the CDC 7600, the value of a subscript (expressed as a constant) may never be zero or negative. On the CRAY, however, subscripts are allowed to be either zero or negative. Hence, the lower and upper bounds of an array may be expressed in the DIMENSION statement on the CRAY (omission of the lower bound causes the default of 1 to be used). Examples follow.

Example 2-5. Set the variable B equal to the third element of the six-element array A.

CDC usage

```
DIMENSION A(6)

B = A(3)
```

CRAY usage

```
DIMENSION A(6)

B = A(3)

    or

DIMENSION A(1:6)

B = A(3)

    or

DIMENSION A(-2:3)

B = A(0)
```

ADJUSTABLE ARRAYS:

Although the CDC 7600 does not allow an array subscript to be zero, the compiler does allow the value of zero to be passed to a subroutine as the dimension declarator of an adjustable array. However, if this array is then used in any calculations within that subroutine, an execution error will occur.

On the CRAY, the dimension declarator(s) must be defined such that the adjustable array will have at least one element in it.

The following example gives a situation on the CDC as discussed above, and the changes necessary to modify that situation to work on the CRAY.

CDC usage

Example 2-6a.

In calling program:

```
  CALL SUB1(A,0,0)
```

In subroutine SUB1:

```
  SUBROUTINE SUB1(A,N,M)

  DIMENSION M(N)
```

CRAY usage

Example 2-6b.

In calling program:

```
  CALL SUB1(A,1,0)
```

In subroutine SUB1:

```
  SUBROUTINE SUB1(A,N,M)

  DIMENSION M(N)
```

2.3.2 Type Declaration - Adjustable Arrays

The CDC 7600 compiler allows a formal parameter used as the dimension declarator of an adjustable array to be typed AFTER it is used to dimension the array. This causes a fatal error on the CRAY, however, since the CRAY requires that the type declaration statement for this variable occur BEFORE it is used to dimension the adjustable array. The following example illustrates this difference.

Example 2-7a.

CDC usage

```
SUBROUTINE MYSUB(A,R,M)
DIMENSION M(R)
INTEGER R
```

Example 2-7b.

CRAY usage

```
SUBOUTINE MYSUB(A,R,M)
INTEGER R
DIMENSION M(R)
```

2.3.3 Array Usage

References to multidimensional arrays may not be shortened on the CRAY as allowed on the CDC 7600.

Example 2-8a.

CDC usage

```
   DIMENSION ARRAY(2,4)
   DO 10 I=1,8
   ARRAY(I) = 0.0
10 CONTINUE
```

Example 2-8b.

CRAY usage

```
   DIMENSION ARRAY(2,4)
   DO 10 I=1,8
   ARRAY(I,1) = 0.0
10 CONTINUE
```

2.4 LEVEL and MOVLEV Statements

The LEVEL and MOVLEV statements associate variables with Large Core Memory (LCM) on the CDC 7600. There are no equivalent statements on the CRAY, nor are any necessary due to the relatively large amount of memory available on the CRAY. All LEVEL statements should be deleted from CDC programs when converting to the CRAY. The MOVLEV statements may be retained on the CRAY if a dummy MOVLEV routine which just copies N words from one array to another is added to the program.

2.5 ENTRY Statement

Differences in CDC and CRAY usage of entry points are discussed below.

CDC usage	CRAY usage
No argument list (assumes the same calling sequence)	In Subroutine Subprograms: Argument list is optional. Arguments need not agree with those specified in FUNCTION, SUBROUTINE, or other ENTRY statements in the same subprogram.

	In Function Subprograms: Argument list is required, even if it is null, e.g. ENTRY NAME()
Function entry names must agree in type with the name appearing in the FUNCTION statement of the subprogram in which the ENTRY statement occurs.	Function entry names may differ in type from the name appearing in the FUNCTION statement of the subprogram in which the ENTRY statement occurs.

2.6 DO LOOPS

A DO LOOP on both the CDC 7600 and the CRAY takes on the following form:

DO sn $i=m_1,m_2,m_3$

where sn = terminal statement number
m_1 = initial value
m_2 = terminal value
m_3 = optional increment value (default = 1)

Differences between the CDC and CRAY usage of these parameters are discussed below.

CDC usage	CRAY usage
i must be the name of an integer variable with a positive, non-zero value.	i may be the name of an integer, real, or double precision variable.
m_1,m_2,m_3 must each be the name of an integer variable with a positive, non-zero value such that m_1+m_3 , $m_2+m_3 <= 2^{17} - 1$	m_1,m_2,m_3 may be the names of integer, real, or double precision variables. If m_1,m_2, and m_3 are integers, then m_1,m_2,m_3 and $(m_2 - m_1 + m_3)$ must be $<= \|2^{23} - 1\|$

If necessary, m_1, m_2, and/or m_3 are converted to the same type variable as i.

m_1, m_2, and m_3 may be positive or negative.

m_3 may not be zero.

[+] If $m_1 > m_2$ the loop is executed at least once.

[+] If $m_1 > m_2$ and $m_3 > 0$ or

$m_1 < m_2$ and $m_3 < 0$, then

the loop is not executed unless the ON=J parameter was specified on the CFT control statement.

[+] This is a significant difference between the CDC and the CRAY treatment of DO LOOPS as it can cause values computed in a CDC program to differ from those output by its CRAY counterpart. In order to obtain the same results from both programs, it is often necessary to force the CRAY to execute each DO LOOP at least once as on the CDC 7600. This can be done by compiling the program with the ON=J parameter as mentioned above.

Section 3

INPUT/OUTPUT

3.0 INPUT/OUTPUT

3.1 Utilities

Use of the following I/O utilities on the CDC and the CRAY are compared.

CDC	CRAY
UNIT(lun)	UNIT(lun)
returns:	returns:
	-2.0 record partly read
-1. transfer successful	-1.0 transfer successful
+0. EOF encountered	0.0 EOF or EOD encountered
+1. Parity error encountered	+1.0 Parity error encountered
	+2.0 Unit error encountered

(Applies only to buffered input/output operations. "lun" is the logical unit number assigned to the file being accessed.)

EOF(lun)	EOF(lun)
returns:	returns:
0. no EOF encountered	-1.0 EOD encountered
non-zero EOF encountered	+1.0 EOF encountered
	0.0 otherwise

("lun" is the logical unit number assigned to the file being accessed. Use of the END=xxx and ERR=xxx parameters on the READ statement are recommended instead of this utility.)

NW=LENGTH(lun)	NW=LENGTH(lun)
returns:	returns:
Number of words transferred (NW) in previous BUFFER IN or READMS call to the file designated by logical unit number "lun".	Number of words transferred (NW) to or from unit "lun". If an EOF or EOD is read from logical unit number "lun" a zero is returned.

(Applies only to buffered input/output operations.)

3.2 Format Specifications

Following are brief descriptions of differences between CDC and CRAY usage of several format specifications.

CDC usage	CRAY usage
Gw.d w >= d + 6	Gw.d w > d + 4
Ow Input field may contain a maximum of 20 octal digits.	Ow Input field may contain a maximum of 22 octal digits.
* and = edit descriptors allowed.	* and = edit descriptors not allowed.
' edit descriptor is not allowed.	' edit descriptor allowed.
nH descriptor can be used for input and output.	nH can only be used for output.
-nX allowed.	-nX not allowed.
n(/) repeats / n times.	n/ repeats / n times.
Tn is the only tab specification available.	TLn and TRn are available in addition to Tn.
BN and BZ editing are not available.	BN and BZ editing available.
S, SP, and SS editing are not available.	S, SP, and SS editing available.
V editing available.	V editing not available.
= edit descriptor available.	= edit descriptor is not available. (However, a function subroutine which provides some of the features of this descriptor is available from Gary Villere of Informatics General Corp. upon request.)

Section 4

SYSTEM UTILITIES

4.0 SYSTEM UTILITIES

Differences between the CDC and CRAY versions of the following system utilities are compared.

CDC	CRAY
CALL DATE(idate) idate is returned as 10Hbmm/dd/yyb, where b denotes a blank	CALL DATE(idate) idate is returned as 8Hmm:dd:yy
CALL JDATE(idate) idate is returned as 5Ryyddd	CALL JDATE(idate) idate is returned as 5Hyyddd
CALL SECOND(cputim) cputim is returned as a real number accurate to two decimal places	CALL SECOND(cputim) cputim is returned as a real number
CALL TIME(itime) itime is returned as 10Hbhh.mm.ss.b, where b denotes a blank	CALL CLOCK(itime) itime is returned as 8Hmm:hh:ss
CALL SYSTEM(ierrno,msg) ierrno = error number msg = execution time errror message issued by user (Hollerith constant)	CALL ABORT(msg) msg = execution time error message issued by user (Hollerith constant <= 9 8-character words)
CALL REMARK(msg) msg = message issued by user to be placed in dayfile (Hollerith constant <= 9 10-character words)	CALL REMARK(msg) msg = message issued by user to be placed in logfile (Hollerith constant <= 8 8-character words)

M=MEM(3HSCM)[+] or M=MEM(3HLCM)[+] M is returned as the current field length	COS 1.11 Version: INTEGER WC,T,DEL DATA WC,T,M,L,DEL /5*0/ CALL MEMORY(WC,T,M,L,DEL) WC is returned as the current field length COS 1.12 Version: INTEGER WC CALL MEMORY('CURFL',WC) WC is returned as the current field length
M=REQMEM(3HSCM,MREQ)[+] or CALL REQMEM(3HSCM,MREQ)[+] or M=REQMEM(3HLCM,MREQ)[+] or CALL REQMEM(3HLCM,MREQ)[+] MREQ = total field length requested by the user M is returned as the new field length received	COS 1.11 Version: INTEGER WC,T,DEL DATA M,L,DEL /3*0/ T = 1 WC = total field length desired CALL MEMORY(WC,T,M,L,DEL) L will be set if the job has received the maximum amount of memory allowed; An error will occur if more memory is requested than is allowed. COS 1.12 Version: INTEGER WC WC = total field length desired CALL MEMORY('FL',WC)

[+] This utility is only available from AMESLIB on the CDC 7600. It is accessed by the following statements on the JCL:

```
ACCESS,A,AMESLIB,ID=AMESLIB.
LIBRARY,A,*.
```

For a further discription of this utility, see "Computational Division User's Bulletin No. 214" available from the Central Computational Division Document Center.

Section 5

MASS STORAGE UTILITIES

5.0 MASS STORAGE UTILITIES

The mass storage INPUT/OUTPUT utilities discussed in this section are OPENMS, CLOSMS, READMS, WRITMS, and STINDX. On the CDC 7600, these utilities are supplied by the system. On the CRAY, however, these utilities must be accessed through AMESLIB as follows:

```
ACCESS,DN=READMS,PDN=READMS,ID=AMESLIB.
LDR,LIB=READMS.
```

Here the default dataset name of $BLD is used, so the DN keyword is not specified. If the user specifies a dataset name different than $BLD using the B keyword on the CFT control statement, then he must specify that dataset name on the LDR statement using the DN keyword (LDR,DN=dataset,LIB=READMS.).

Basically, these routines are used in the same manner on both machines. However, two notable differences have been found in the usage of OPENMS. Discussions on these differences follow.

CDC

OPENMS allows the user to use a two-dimensional array for the defining the Master Index. It also allows the user to directly specify asstarting location for this array in the call to OPENMS.

Example 5-1.

```
COMMON /INDEX/ INDEX(2,3),N
              .
              .
              .
CALL OPENMS(LUN,INDEX(1,N),
            LENGTH,IT)
```

where LUN, LENGTH, and IT are previously defined.

CRAY

OPENMS does not allow the master index array to be two-dimensional, nor does it allow a starting location to be directly specified in the call to OPENMS.

Example 5-2. For example 5-1 to work on the CRAY do the following:

```
COMMON /INDEX/ INDX1(2),INDX2(2),
               INDX3(2),N
                     .
                     .
IF (N.EQ.1) CALL OPENMS(LUN,INDX1,
                        LENGTH,IT)
IF (N.EQ.2) CALL OPENMS(LUN,INDX2,
                        LENGTH,IT)
IF (N.EQ.3) CALL OPENMS(LUN,INDX3,
                        LENGTH,IT)
```

where LUN, LENGTH, and IT are previously defined.

Section 6
CONVERSION HAZARDS

6.0 CONVERSION HAZARDS

6.1 Subroutines - Agreement in Number of Arguments

On the CDC 7600, if a user passes too many arguments to a subroutine the extra arguments are ignored. This is because the compiler reserves address space in the CALLING program for the table containing the addresses of those arguments.

On the CRAY, however, the compiler reserves the space for the argument address table in the routine being CALLED. Therefore, the amount of space reserved is only enough for the number of formal parameters specified in the subroutine declaration. It is the CALLING routine, however, which fills up this table at execution time. Thus, if a routine is called with too many arguments, the addresses of the extra arguments will over-write the lower portion of the routine immediately PRECEDING the CALLED subroutine. This type of error is very difficult to trace as it often causes a section of code totally unrelated to the called subroutine to fail at execution time.

APPENDIX A

DIFFERENCES IN CDC AND CRAY

DEFAULT FILE NOMENCLATURE

CDC	CRAY
JCL filenames:	
INPUT	$IN
OUTPUT	$OUT
OLDPL	$PL
NEWPL	$NPL
COMPILE	$CPL
LGO	$BLD
VAX <eor>,<eof> separators:	
%%EOR	/EOF
%%EOF	/EOF