Gould Scientific Run-time Library

Release 4.2

Reference Manual

July 1985

Publication Order Number 323-004020-000

GOULD
Electronics

This manual is supplied without representation or warranty of any kind. Gould Inc., Computer Systems Division therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this publication or any material contained herein.

LIMITED RIGHTS LEGEND

for

PROPRIETARY INFORMATION

The information contained herein is proprietary to Gould CSD and/or its vendors, and its use, disclosure or duplication is subject to the restrictions stated in the Gould CSD license agreement Form No. 620-06(1/82) or the appropriate third-party sublicense agreement. The information is provided to government customers with limited rights as described in DAR 7-104.9A.

MPX-32 and CONCEPT/32 are Trademarks of Gould Inc.

HISTORY

The Scientific Runtime Library Release 4.1 Reference Manual, Publication Order Number **323-004020-000,** was printed July 1984.

Publication Order Number **323-004020-001** (Change 1, Release 4.2) was printed July 1985.

The updated manual contains the following pages:

* Zero in this column indicates an original page.

On a change page, the portion of the page affected by the latest change is indicated by a vertical bar in the outer margin of the page. However, a completely changed page will not have a full length bar, but will have the change notation by the page number.

CONTENTS

CHAPTER 4 MPX-32 COMPATIBLE OR NATIVE MODE SUBPROGRAMS

CHAPTER 5 MPX-32 NATIVE MODE SUBROUTINES

CHAPTER 6 SUPPORT FOR OTHER STANDARDS

TABLES

Documentation Conventions

The following notations are used in this document to express source text syntax. Notational symbols are not a part of the text entered by the user; they are merely aids designed to make the syntax of the statements easier to understand.

Notation	Meaning
lowercase letters	Lowercase letters identify a generic element that must be replaced with a user-selected value or a returned value. For example, in the statement: CALL STATUS (u, s, n) u is replaced by a user-selected value, while s and n are returned values.
UPPERCASE LETTERS	Uppercase letters represent keywords and must be entered as shown for input. In the control statement: CALL M:END the words CALL and M:END represent keywords.
[ ]	Brackets indicate that the enclosed elements are optional CALL X:DISCON (task, [istatus] , [$n]) istatus and $n are optional. Several elements placed one under the other inside a pair of brackets indicate the user may select one or none of the elements. [HI LO]

. . .	A horizontal ellipsis indicates repetition. GO TO (x_1,x_2,x_3,...,x_n)
. . .	A vertical ellipsis indicates that statements or instructions have been omitted. J=3 . . . GO TO 450
( )	Parentheses are part of the source text entered by the user and must appear as shown in statement syntax when entered. CALL M:DELTIM (id) A two character timer id is required and must be enclosed by parentheses
Numbers and special characters	Numbers that appear on the line (i.e., not subscripts), special symbols, and punctuation marks other than dotted lines, brackets, braces, and underlines appear as shown in output messages and must be entered as shown when input. CALL M:PFAD (filename, i, *label) An asterisk * is required before the user-selected value *label.

CHAPTER 1

INTRODUCTION

1.1 Scientific Run-time Library

The Scientific Run-time Library (SRTL) is a collection of subprograms that provides mathematical library routines, operating system service routines, and language extensions for FORTRAN 77+. The SRTL routines may also be accessed by Pascal, COBOL, and Assembly programs, if the proper calling interface is maintained. The SRTL contains routines which conform to the following FORTRAN standards:

- ANSI - X3.9-1978 (American National Standards Institute)
- MIL-STD - 1753-1978 (Military Standard)
- ISA/S61.1-1976 (Instrument Society of America)
- ISA/S61.2-1978

There are four possible installation modes for the SRTL. They are: native/hardware assisted, native/non-hardware assisted, compatible/hardware assisted, and compatible/non-hardware assisted.

The native and compatible aspects of the compiler are indicative of the file system interface being selected. Code is contained in the native mode SRTL for accessing available MPX-32 resources through the MPX-32 file system in accordance with methods native only to MPX-32 Rev. 2 and Rev. 3. Code is contained in the compatible mode SRTL for accessing resources in accordance with methods that are compatible with MPX-32 Rev. 1.

The hardware assisted and non-hardware assisted aspects of the SRTL allow for the selection or non-selection of certain hardware floating point instructions. These instructions are available on CONCEPT 32/67, 32/87 and 32/97 machines. Code contained in the hardware assisted mode SRTL utilizes refined math algorithms incorporating special register to register hardware floating point instructions. Code contained in the non-hardware assisted mode SRTL retains the math algorithms contained in previous releases of the SRTL. Either mode may be used on machines having the special register to register instructions although the hardware assisted mode is recommended for those machines. Machines not having the special instructions are restricted to using the non-hardware assisted mode.

In order to catalog programs compiled under FORTRAN 77+, the SRTL must be assigned as one of the available object libraries to the cataloger. Unlike the compiler which can use options for mode selection, the mode of the SRTL is dependent solely upon which installation of the SRTL is assigned to the cataloger at catalog time. The FORTRAN 77+ compiler and the SRTL have the same possible installation modes. It is therefore recommended that the modes selected at installation for the SRTL match those selected for the compiler installation.

Table 1-1
FORTRAN 77+/SRTL Installation Modes

		REV. 2	REV. 3
SERIES 32/7X	NON-HW ASSIST ONLY	compatible native *	
CONCEPT 32/27	NON-HW ASSIST ONLY	compatible native	compatible native *
CONCEPT 32/67	HW ASSIST		compatible native *
	NON-HW ASSIST		compatible native
CONCEPT 32/87	HW ASSIST	compatible native	compatible native *
	NON-HW ASSIST	compatible native	compatible native
CONCEPT 32/97	HW ASSIST		compatible native *
	NON-HW ASSIST		compatible native
* - RECOMMENDED			

For more information about the MPX-32 file system, refer to the MPX-32 Reference Manual.

During library installation, the user can place either the native or compatible, hardware assisted or non-hardware assisted set of routines into a library. For convenience it is advisable to place the set most frequently used in MPXLIB/MPXDIR. If the other sets are also desired, they can be placed under other library/directory names. The alternate libraries can be specified for use at catalog time.

1.2 Description of Scientific Run-Time Library Routines

The routines in the SRTL are divided into four classes:

- Mathematical Library
- MPX-32 System Service
- Supplemental User Support
- Supplemental Language Support

1.2.1 Mathematical Library Routines

The mathematical library class of routines contains the FORTRAN intrinsic and bit manipulation functions. These functions can be directly referenced from FORTRAN 77+. The mathematical library also contains other routines for run-time support of data type conversion, exponentiation, multiplication, and division. These routines are not available directly from FORTRAN 77+, but can be referenced from Assembler programs if the proper calling sequence is maintained. References to these routines are generated as needed by the FORTRAN 77+ compiler.

The mathematical library routines are described in more detail in Chapter 2 and Section 6.2.

1.2.2 MPX-32 System Service Routines

The system service routines provide access to the MPX-32 System Services. These routines are the M:, X:, and X_ routines described in Chapters 3, 4, and 5, which contain descriptions of subprograms that provide MPX-32 services to FORTRAN 77+ programs. Chapter 3 describes subprograms which are available to MPX-32 users only in compatible mode. Chapter 4 describes subprograms which are available to MPX-32 users in either compatible or native mode. Chapter 5 describes subprograms which are available only in native mode.

The following notes refer to the use of the MPX-32 service subprograms:

- Attempts to mix modes, i.e. using a compatible mode only subprogram while executing using native mode, will result in an RS99 error.
- The subprograms are not reentrant. Use caution when processing break interrupts during I/O routines.

1.2.3 Supplemental User Support Routines

The supplemental user support routines provide additional language support for FORTRAN 77+. These routines allow for such functions as: obtaining the address of a variable, direct access to MPX-32 SVC services, obtaining the system time and date, task control, file access, and unformatted random access I/O. These routines are described in Section 4.4 and Sections 6.3 through 6.6.

1.2.4 Supplemental Language Support Routines

The supplemental language support routines are not directly available to FORTRAN 77+. References to these routines are generated by the FORTRAN 77+ compiler to support language features such as READ and WRITE.

1.3 Scientific Run-time Differences

Release 4.1 of the Scientific Run-time Library differs from previous releases as follows:

- Regular FORMAT statements are now converted to format item tables. These are readily used by the run-time library routines to eliminate redundant re-evaluation of formats during run time I/O.

- Extended memory addressing has been improved.

- The logical functions IAND, IOR, IEOR, and NOT will now be expanded inline unless these functions are specifically defined as EXTERNAL to the program, or declared INTRINSIC and referenced in a subprogram where the name of the function was passed as an argument to the subprogram.

- Date and time of assembly and product identification information are stored in the object code for all SRTL routines, and are now available at the user's option at catalog and library editing time. This will better identify native, compatible, hardware assisted and non-hardware assisted routines.

- The utility of BUFFERIN/BUFFEROUT has been modified so that the user can select a "sector specifier" as an optional parameter in order to do random I/O on disc files only.

CHAPTER 2

MATHEMATICAL LIBRARY ROUTINES

2.1 Support Functions

The mathematical library is a collection of mathematical and utility functions that provides full computational support for FORTRAN 77+ and Assembly programs. The functions are listed in Table 2-1 at the end of this chapter.

Common features of the functions are:

- Each is referenced by its name, which in all cases is one to eight alphanumeric characters, the first of which is alphabetic.
- Each returns a single value; i.e., each returns one value to the expression from which it was referenced.
- Each is referenced by an expression containing a function name or by a direct call to a specified routine name.
- Each FORTRAN 77+ intrinsic function name is predefined to the compiler and is automatically typed.

2.2 Run-time Support Routines

In addition to the FORTRAN intrinsic and bit manipulation functions, the mathematical library contains other routines for run-time support of data type conversion, exponentiation, multiplication and division. These routines are distinguished by the presence of a period in the routine name (e.g., C.IR). These routines are not available to the FORTRAN user directly; calls are generated inline by the compiler as they are needed. However, they are available to Assembly language users.

The FORTRAN 77+ compiler generates inline code for the following intrinsic functions.

ABS	SIGN	DIM	REAL	SNGL
IABS	ISIGN	IDIM	DREAL	DBLE
DABS	DSIGN	JDIM	AIMAG	CMPLX
JABS	JSIGN	CONJG	DIMAG	DCMPLX

The compiler will not generate inline code under the following circumstances:

- the function is declared EXTERNAL
- the function is declared INTRINSIC and referenced in a subprogram where the name of the function was passed as an argument to the subprogram.

In those cases, the FORTRAN 77+ compiler generates a branch and link instruction for those references to the function. Note that the use of inline code results in faster execution at run-time.

2.3 Standard Calling Sequences

All mathematical library routines use a standard calling sequence. The calling sequences for FORTRAN are described in the FORTRAN 77+ Reference Manual. The calling sequences for Assembly are described in the following sections.

2.3.1 Single Argument

For single argument routines, the calling sequence is:

```
LA      1,Arg
BL      Routine
```

2.3.2 Multiple Argument

For multiple argument routines, the calling sequence is:

```
BL      Routine
DATAW   n       n is the number of arguments.
ACs     Arg1    generate address pointers to arguments
ACt     Arg2            .
 .       .              .
 .       .              .
 .       .              .
ACu     Argn    s, t, ..., u are B, H, W, or D, depending on the mode of the
        argument
```

2.4 Exceptions to the Standard Calling Sequence

The routines in the mathematical library for run-time support of data type conversion, exponentiation, multiplication, and division do not follow the standard calling sequences. The calling sequences for these routines are described in the following sections.

R in the calling sequences is defined as follows:

R7	passes halfword integer, integer, and single precision floating-point values.
R6,R7	pass doubleword integer, double precision floating-point, and single precision complex values.
R4,R5,R6,R7	pass double precision complex values.

These general purpose registers are used to pass arguments to the routines. The result produced by the routines is returned in these same registers.

2.4.1 Conversion Routines

For conversion routines (C.DJ, C.JD, C.IR, C.RI), the calling sequence is:

```
Load    R,Arg                EXAMPLE:  LW   7,REALWORD
BL      C.xx                           BL   C.RI
```

2.4.2 Exponentiation Routines

For exponentiation routines (P.ZZ, P.DI, P.CC, P.CI, P.RR, P.RI, P.II, P.JJ), the calling sequence is:

```
Load    R,Base               EXAMPLE:  LD   6,DBLEREAL
LA      1,Exponent                     LA   1,EXPONENT
BL      P.xx                           BL   P.DI
```

2.4.3 Multiplication Routines

For multiplication routines (M.ZZ, M.CC, M.JJ), the calling sequence is:

```
Load    R,Multiplier         EXAMPLE:  LD   6,CMPLXARG
LA      1,Multiplicand                 LA   1,MULTCAND
BL      M.xx                           BL   M.CC
```

2.4.4 Division Routines

For division routines (D.ZZ, D.CC, D.JJ), the calling sequence is:

```
Load    R,Dividend           EXAMPLE:  LD   4,REALPART
LA      1,Divisor                      LD   6,IMAGPART
BL      D.xx                           LA   1,DIVISOR
                                       BL   D.ZZ
```

2.5 Argument Checking and Error Conditions

Generally, the mathematical library routines do not check to determine if the number of arguments passed equals the number of arguments required or if the passed arguments are of appropriate type.

If any reference to a mathematical library routine results in an error condition, the routine may continue execution rather than abort. In this situation, the routine attempts to provide a reasonable result. However, a run-time diagnostic message is written to the Diagnostic Output (DO) logical file code by a call to L.ERR, and condition code 1 is set on return to flag the error condition.

2.6 Register Preservation

In the case of a user-generated reference to a mathematical library routine, except the run-time support routines, all registers are volatile and may be assumed to be destroyed. Exceptions are noted in the program descriptions of the individual mathematical library routines.

In the case of the run-time support routines, all registers are preserved except 0, 1, and R, where R is the register(s) in which the result is returned. The conversion routines preserve register 1.

2.7 Accuracy of Results

The maximum relative error in single precision floating-point routines is approximately 2^{-19}. In double precision, the maximum relative error is approximately 2^{-49}.

Any exceptions to these accuracy figures are noted in the individual program descriptions for the mathematical library routines.

2.8 Mathematical Library Usage Conventions

The following conventions are applicable to various mathematical library routines.

2.8.1 Zero Division and Exponentiation

Any nonzero positive or negative number divided by zero returns the largest positive or smallest negative number, respectively. In both cases, the overflow bit (condition code 1) is set. Zero divided by zero (0/0) and zero raised to the zero power (0^0) are both defined to be one.

2.8.2 Calling Double precision Routines

When calling double precision routines, the user must specify double precision arguments, because no mode conversions are performed.

2.8.3 Complex Routines

All complex routines, except CMPLX, DCMPLX, P.CC, P.ZZ, M.CC, M.ZZ, D.CC, and D.ZZ, are single-argument routines.

2.8.4 Testing for Overflow

Test for arithmetic overflow by calling the subroutine OVERFL as follows:

```
CALL OVERFL(I)
```

where

I An INTEGER variable that assumes a value of 1 or 2 as follows:

1 if any hardware arithmetic exception trap has occurred since the last call to OVERFL.

2 if no hardware arithmetic exception trap has occurred since the last call to OVERFL.

A call to this subroutine resets the overflow indicator. This test is valid only if the arithmetic exception trap is enabled and the arithmetic exception handler (or an equivalent) is installed.

The value returned by a function may also indicate the occurrence of an overflow condition by being equal to the maximum positive (MAXPOS) or maximum negative (MAXNEG) value.

Test for arithmetic overflow or other error conditions in the assembler by testing the overflow bit (condition code 1) immediately upon return from a given math library routine. If this bit is set, an arithmetic exception or other error condition has occurred.

2.8.5 Common External Temporary Storage

Common external temporary storage (L.TEMP0 through L.TEMP25) and some common constants for mathematical library routines are provided by a data area, MLSTOR.

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	DIMAG	1	Double complex	Double real	To obtain the imaginary part of a double precision complex floating-point number.	None		
Nearest whole number	(A) ANINT	1	Real	Real	To find the nearest whole number.	None		
	(A) DNINT	1	Double	Double	To find the nearest whole number.	None		
Nearest integer	(A) IDNINT	1	Double	Integer	To find the nearest integer.	None		
	(A) NINT	1	Real	Integer	To find the nearest integer.	None		
Conversion	C.DJ	1	Double real	Double integer	To convert from double precision floating-point to double precision integer format.	None	|ARG|>2**63	±7FFFFFFFFFFFFFFF (CC1 bit set)
	C.IR	1	Integer	Real	To convert from integer to single precision floating-point format.	None	|ARG|>2**28 |ARG|>2**24 Refer to Note 14.	|Error|<128 |Error|<8
	C.JD	1	Double integer	Double real	To convert from double precision integer to double precision floating-point format.	None	|ARG|>2**56	|Error|<28
	C.JR	1	Double integer	Real	To convert from double precision integer to single precision floating-point format.	None		
	C.RI	1	Real	Integer	To convert from single precision floating-point to integer format.	None	|ARG|>2**31	±7FFFFFFF (CC1 bit set)
	(A) DBLE	1	Real	Double real	To convert a single precision floating-point quantity to a double precision floating-point format. Refer to note 3.	None		

Mathematical Library Routines (Sheet 10 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
Complex conjugate	(A) CONJG	1	Complex	Complex	To obtain the conjugate of a single precision complex floating-point number. Refer to note 6.	None		
	DCONJG	1	Double complex	Double complex	To obtain the conjugate of a double precision complex floating-point number. Refer to note 6.	None		
Complex form	(A) CMPLX	1 or 2	Integer Real Double real	Complex	To express one or two integer, single precision floating-point, or double precision floating-point numbers in complex form. Refer to note 4.	None		
		1	Complex Double complex	Complex	To express one complex or double precision complex number in complex form. Refer to note 4.	None		
	DCMPLX	1 or 2	Integer Real Double real	Double complex	To express one or two integer, single precision floating-point, or double precision floating-point numbers in double precision complex form. Refer to note 4.	None		
		1	Complex Double complex	Double complex	To express one complex or double precision complex number in double precision complex form. Refer to note 4.	None		
Complex real part	DREAL	1	Double complex	Double real	To obtain the real part of a complex double precision floating-point number.	None		
	(A) REAL	1	Complex	Real	To obtain the real part of a single precision complex floating-point number. Refer to note 2.	None		
Complex imaginary part	(A) AIMAG	1	Complex	Real	To obtain the imaginary part of a single precision complex floating-point number. Refer to note 6.	None		

Mathematical Library Routines (Sheet 9 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	(A) TANH	1	Real	Real	To compute the hyperbolic tangent of a single precision floating-point number.	None	ARG≥12 Ln 2	=1
Logarithm	(A) ALOG	1	Real	Real	To compute the natural logarithm of a single precision floating-point number.	LOG 039	ARG≤0	=80000001
	(A) ALOG10	1	Real	Real	To compute the common logarithm of a single precision floating-point number.	ALOG10 039	ARG≤0	=80000001
	CDLOG	1	Double complex	Double complex	To compute the principal value of the natural logarithm of a double precision complex floating-point number.	CDLOG 007	Any arithmetic overflow	=0 (CC1 bit set)
	(A) CLOG	1	Complex	Complex	To compute the principal value of the natural logarithm of a complex floating-point number.	CLOG 031	Any arithmetic overflow	=0 (CC1 bit set)
	(A) DLOG	1	Double real	Double real	To compute the natural logarithm of a double precision floating-point number.	DLOG 017	ARG≤0	=8000000000000001
	(A) DLOG10	1	Double real	Double real	To compute the common logarithm of a double precision floating-point number.	DLOG10 017	ARG≤0	=8000000000000001
Exponential	CDEXP	1	Double complex	Double complex	To compute e^z for double precision complex floating-point z.	CDEXP 004	Any arithmetic overflow	=0 (CC1 bit set)
	(A) CEXP	1	Complex	Complex	To compute e^c for complex floating-point c	CEXP 030	Any arithmetic overflow	=0 (CC1 bit set)
	(A) DEXP	1	Double real	Double real	To compute e^d for double precision floating-point d.	None	ARG<-256 Ln 2	=0
							ARG>256 Ln 2	=7FFFFFFFFFFFFFFF
	(A) EXP	1	Real	Real	To compute e^r for single precision floating-point r.	None	ARG<-256 Ln 2	=0
							ARG>256 Ln 2	=7FFFFFFF

Mathematical Library Routines (Sheet 8 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Results
Hyperbolic Cosine	(A) COSH	1	Real	Real	To compute the hyperbolic cosine of a single precision floating-point number.	None		
	(A) DCOSH	1	Double real	Double real	To compute the hyperbolic cosine of a double precision floating-point number.	None		
Tangent	(A) TAN	1	Real	Real	To compute the tangent of a single precision floating-point number.	None		
	(A) DTAN	1	Double	Double	To compute the tangent of a double precision floating-point number.	None		
Arctangent	(A) ATAN	1	Real	Real	To compute the principal value of the inverse tangent of a single precision floating-point number.	None		
	(A) DATAN	1	Double real	Double real	To compute the principal value of the arctangent of a double precision floating-point number.	None		
	(A) ATAN2	2	Real	Real	To compute the inverse tangent of the quotient of two single precision floating point numbers, adjusting for proper quadrant.	None	Arithmetic overflow	+ (π/4) (CC1 bit set)
	(A) DATAN2	2	Double real	Double real	To compute the arctangent of the quotient of two double precision floating-point quantities, adjusting for proper quadrant.	None	Arithmetic overflow	$\pm$ (π/4) (CC1 bit set)
Hyperbolic tangent	(A) DTANH	1	Double real	Double real	To compute the hyperbolic tangent of a double precision floating-point number.	None	ARG$\geq$28 Ln 2	=1

Mathematical Library Routines (Sheet 7 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	CDSIN	1	Double Complex	Double Complex	To compute the sine of a double precision complex floating-point number.	CDSIN 005	Any arithmetic overflow	=0 (CC1 bit set)
Arc sine	(A) ASIN	1	Real	Real	To compute the arc-sine of a single precision floating-point number.	ASIN 037	|ARG|>1	=0 (CC1 bit set)
	(A) DASIN	1	Double	Double	To compute the arc-sine of a double precision floating-point number.	DASIN 092	|ARG|>1	=0 (CC1 bit set)
Hyperbolic Sine	(A) SINH	1	Real	Real	To compute the hyperbolic sine of a single precision floating-point number.	None	|ARG|>256 Ln 2	±7FFFFFFF (CC1 bit set)
	(A) DSINH	1	Double real	Double real	To compute the hyperbolic sine of a double precision floating-point number.	None	|ARG|>256 Ln 2	±7FFFFFFFFFFFFFFF (CC1 bit set)
Cosine	(A) COS	1	Real	Real	To compute the cosine of a single precision floating-point number.	None		For best results, ARG should be in the range (0, 2π)
	(A) CCOS	1	Complex	Complex	To compute the cosine of a complex floating-point number.	CCOS 033	Any arithmetic overflow	=0 (CC1 bit set)
	(A) DCOS	1	Double real	Double real	To compute the cosine of a double precision floating-point number.	None		For best results, ARG should be in the range (0, 2π)
	CDCOS	1	Double complex	Double complex	To compute the cosine of a double precision complex floating-point number.	CDCOS 006	Any arithmetic overflow	=0 (CC1 bit set)
Arccosine	(A) ACOS	1	Real	Real	To compute the arc cosine of a single precision floating-point number.	ACOS 037	|ARG|>1	=0 (CC1 bit set)
	(A) DACOS	1	Double	Double	To compute the arc cosine of a double precision floating-point number.	DACOS 092	|ARG|>1	=0 (CC1 bit set)

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	(A) MAX1	n≥2	Real	Integer	To find the maximum value in a variable length list of single precision floating-point numbers and convert the result to integer format.	None		
	(A) DMAX1	n≥2	Double real	Double real	To find the maximum value in a variable length list of double precision floating-point values.	None		
Minimum	(A) AMIN0	n≥2	Integer	Real	To find the minimum value in a variable length list of integers and convert the result to single precision floating-point format.	None		
	(A) AMIN1	n≥2	Real	Real	To find the minimum value in a variable length list of reals.	None		
	(A) MIN0	n≥2	Integer	Integer	To find the minimum value in a variable length list of integers.	None		
	(A) MIN1	n≥2	Real	Integer	To find the minimum value in a variable length list of single precision numbers and convert the result to integer format.	None		
	(A) DMIN1	n≥2	Double real	Double real	To find the minimum value in a variable length list of double precision floating-point values.	None		
Sine	(A) SIN	1	Real	Real	To compute the sine of a single precision floating-point number.	None		For best results ARG should be in the range (0, 2π)
	(A) CSIN	1	Complex	Complex	To compute the sine of a complex floating-point number.	CSIN 032	Any arithmetic overflow	=0 (CC1 bit set)
	(A) DSIN	1	Double real	Double real	To compute the sine of a double precision floating-point number.	None		For best results, arguments should be in the range (0, 2π)

Mathematical Library Routines (Sheet 5 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
Subtraction	(A) DDIM	2	Double real	Double real	To find the positive difference between two double precision arguments.	None		
	(A) DIM	2	Real	Real	To find the positive difference between two single precision floating-point numbers.	None		
	(A) IDIM (Note 16)	2	Integer	Integer	To find the positive difference between two integers.	None		
Multiplication	(A) DPROD	2	Real	Double real	To find the double precision product of two real arguments.	None	Arithmetic overflow	±7FFFFFFFFFFFFFFF (CC1 bit set)
	M.CC	2	Complex	Complex	To multiply two complex floating-point numbers.	M.CC 035	Arithmetic overflow	=0 (CC1 bit set)
	M.JJ	2	Double integer	Double integer	To multiply two doubleword integers.	None	Arithmetic overflow	±7FFFFFFFFFFFFFFF (CC1 bit set)
	M.ZZ	2	Double complex	Double complex	To multiply two double precision complex floating-point numbers.	M.ZZ 009	Arithmetic overflow	=0 (CC1 bit set)
Division	D.CC	2	Complex	Complex	To divide one complex floating-point number by another.	D.CC 036	Arithmetic overflow	=0 (CC1 bit set)
	D.JJ	2	Double integer	Double integer	To divide one integer doubleword by another.	None	Division by zero	±7FFFFFFFFFFFFFFF (CC1 bit set)
	D.ZZ	2	Double complex	Double complex	To divide a double precision complex floating-point number by another.	D.ZZ 010	Arithmetic overflow	=0 (CC1 bit set)
Absolute value	(A) ABS	1	Real	Real	To compute the absolute value of a single precision floating-point number or an integer.	None		
	(A) CABS	1	Complex	Real	To compute the absolute value of a complex floating-point argument. Refer to note 6.	CABS 028	Arithmetic overflow	=0 (CC1 bit set)

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	CDABS	1	Double complex	Double real	To compute the absolute value of a double precision complex floating-point number. Refer to note 6.	CDABS 002	Arithmetic overflow	=0 (CC1 bit set)
	(A) DABS	1	Double real	Double real	To compute the absolute value of a double precision floating-point number.	None		
	(A) IABS (Note 16)	1	Integer	Integer	To compute the absolute value of an integer argument.	None	ARG=MAXNEG	=MAXNEG (<0) (CC1 bit set)
Square root	CDSQRT	1	Double complex	Double complex	To compute the principal square root of a double precision complex floating-point number.	CDSQRT 001	Arithmetic Overflow	=0 (CC1 bit set)
	(A) CSQRT	1	Complex	Complex	To compute the principal square root of a complex floating-point number.	CSQRT 027	Arithmetic Overflow	=0 (CC1 bit set)
	(A) DSQRT	1	Double real	Double real	To compute the positive square root of a double precision floating-point number.	DSQRT 011	ARG<0	SQRT (|ARG|)
	(A) SQRT	1	Real	Real	To compute the positive square root of a single precision floating-point number.	SQRT 049	ARG<0	SQRT (|ARG|)
Modular arithmetic	(A) AMOD	2	Real	Real	To find the remainder when one single precision floating-point number is divided by another.	AMOD 045	|QUOTIENT|>2**24 Arithmetic Overflow Division by zero	=0 (CC1 bit set)
	(A) DMOD	2	Double real	Double real	To compute the remainder obtained in dividing the first double precision floating-point argument by the second.	DMOD 020	|QUOTIENT|>2**52 Arithmetic Overflow Division by zero	=0 (CC1 bit set)
	(A) MOD	2	Integer	Integer	To find the remainder when the first integer is divided by the second.	None	Division by zero	=0 (CC1 bit set)

Mathematical Library Routines (Sheet 2 of 13)

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
Fraction truncation	(A) AINT	1	Real	Real	To truncate the fractional bits from a single precision floating-point quantity. Refer to note 1.	None		
	(A) DINT	1	Double real	Double real	To truncate the fractional bits of a double precision floating-point quantity. Refer to note 1.	None		
Sign transfer	(A) DSIGN	2	Double real	Double real	To transfer the sign of the second double precision floating-point argument to the first double precision floating point argument.	None		
	(A) SIGN	2	Real	Real	To transfer the sign of the second single precision floating-point argument to the first single precision argument.	None		
	(A) ISIGN (Note 16)	2	Integer	Integer	To transfer the sign of the second single precision integer argument to the first single precision integer argument.	None		
Exponent	P.CC	2	Complex	Complex	To compute the principal value of a complex floating-point number raised to a complex floating-point number power.	P.CC 029	Arithmetic overflow	=0 (CC1 bit set)
	P.CI	2	Complex, Integer	Complex	To raise a complex floating-point number to an integer power.	P.CI 034	Arithmetic overflow	=0 (CC1 bit set)
	P.DD	2	Double real	Double real	To compute a double precision floating-point power of a double precision floating-point number.	None	Arithmetic overflow	7FFFFFFFFFFFFFFF (Sign of result = sign of base) (CC1 bit set)
	P.DI	2	Double real, Integer	Double real	To raise a double precision floating-point quantity to an integer power.	None	Any arithmetic overflow	7FFFFFFFFFFFFFFF (CC1 bit set)
	P.II	2	Integer	Integer	To raise an integer value to an integer power.	None	Any arithmetic overflow	+7FFFFFFF (CC1 bit set)

Mathematical Library Routines (Sheet 3 of 13)

Table 2-1

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
	P.JI	2	Double integer, Integer	Double integer	To raise a double precision integer quantity to an integer power.	None	Any arithmetic overflow	7FFFFFFFFFFFFFFF (CC1 bit set)
	P.JJ	2	Double integer	Double integer	To raise a double word integer to a doubleword integer power.	None	Any arithmetic overflow	7FFFFFFFFFFFFFFF (CC1 bit set)
	P.RI	2	Real, Integer	Real	To raise a single precision floating-point number to an integer power.	None	Any arithmetic overflow	±7FFFFFFF (CC1 bit set)
	P.RR	2	Real	Real	To raise a single precision floating-point number to a single precision floating-point power. Refer to note 15.	None	Any arithmetic overflow	±7FFFFFFF (CC1 bit set)
	P.ZI	2	Double complex, Integer	Double complex	To raise a double precision complex floating-point number to an integer power.	P.ZI 008	Any arithmetic overflow	=0 (CC1 bit set)
	P.ZZ	2	Double complex	Double complex	To compute the principal value of of a double precision complex power of a double precision complex number.	P.ZZ 003	Any arithmetic overflow	=0 (CC1 bit set)
Maximum	(A) AMAX0	n≥2	Integer	Real	To find the maximum value in a variable length list of integers and convert the result to single precision floating-point format.	None		
	(A) AMAX1	n≥2	Real	Real	To find the maximum value in a variable length list of reals.	None		
	(A) MAX0	n≥2	Integer	Integer	To find the maximum value in a variable length list of integer values.	None		

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
Length of Character String	(A) LEN	1	Character (note 12)	Integer	To determine the length of a character string.			
Lexically Greater Than or Equal to	(A) LGE	2	Character (note 12)	Logical	To compare two character strings and return a logical value of .TRUE. or .FALSE., depending on the ASCII collating sequence.			
Lexically Greater Than	(A) LGT	2	Character (note 12)	Logical	To compare two character strings and return a logical value of .TRUE. or .FALSE., depending on the ASCII collating sequence.			
Lexically Less Than or Equal to	(A) LLE	2	Character (note 12)	Logical	To compare two character strings and return a logical value of .TRUE. or .FALSE., depending on the ASCII collating sequence.			
Lexically Less Than	(A) LLT	2	Character (note 12)	Logical	To compare two character strings and return a logical value of .TRUE. or .FALSE., depending on the ASCII collating sequence.			
Index of A Substring	(A) INDEX	2	Character (note 12)	Integer	To return an integer value locating substring c_2 within a string c_1. Refer to note 7.			
Inclusive OR	(I) IOR	2	Integer	Integer	To permit interrogation and manipulation of integers on a bit-by-bit basis. Refer to 6.2.1.3.			
Logical Product	(I) IAND	2	Integer	Integer	To permit interrogation and manipulation of integers on a bit-by-bit basis. Refer to 6.2.1.1			
Logical complement	(I) NOT	1	Integer	Integer	To permit interrogation and manipulation of integers on a bit-by-bit basis. Refer to 6.2.1.4.			
Exclusive OR	(I) IEOR	2	Integer	Integer	To permit interrogation and manipulation of integers on a bit-by-bit basis. Refer to 6.2.1.2.			

Function	Routine Name	Number of Arguments	Argument Modes	Result Modes	Routine Purpose	Error Message	Exception Condition	Exception Result
Logical shift	(I) ISHFT	2	Integer	Integer	To shift k places all bits representing argument m. Refer to 6.2.2.1.			
Circular shift	(I) ISHFTC	3	Integer	Integer	To shift circularly k places the rightmost ic bits of argument m. Refer to 6.2.2.2.			
Bit test	(I) BTEST	2	Integer	Logical	To test a specified bit of an integer. Refer to 6.2.3.1.			
Bit set	(I) IBSET	2	Integer	Integer	To set a specified bit of an integer. Refer to 6.2.3.2.			
Bit clear	(I) IBCLR	2	Integer	Integer	To clear a specified bit of an integer. Refer to 6.2.3.3.			
Bit field extraction	(I) IBITS	3	Integer	Integer	To extract a specified subfield of bits. Refer to 6.2.4.1.			

Mathematical Library Routines (Sheet 13 of 13)

Table 2-1

Notes for Table 2-1

The Mathematical Library Routines, which conform to the ANSI X3.9-1978 (American National Standards Institute), are identified by an (A) next to the routine name.

The Mathematical Library Routines, which conform to the ISA/S61.1-1976 (Instrument Society of America), are identified by an (I) next to the routine name.

1. For an integer argument, INT (a) = a. For a real or double precision argument, there are two cases: if $|a| < 1$,INT (a) = 0, but if $|a| \geq 1$,INT (a) equals the integer whose magnitude is the largest integer that does not exceed the magnitude of (a) and whose sign is the same as the sign of (a). For example:

   INT (-3.7)=-3

   For a complex argument, INT (a) is the value obtained by applying the above rule to the real part of (a).

   For a real argument, IFIX (a) is the same as INT (a).

2. For a real argument, REAL (a) is (a). For an integer or double precision argument, REAL (a) is as much precision of the significant part of (a) as a real datum can contain. For a complex argument, REAL (a) is the real part of (a).

   For an integer argument, FLOAT (a) is the same as REAL (a).

3. For a double precision argument, DBLE (a) is (a). For an integer or real argument, DBLE (a) is as much precision of the significant part of (a) as a double precision datum can contain. For a complex argument, DBLE (a) is as much precision of the significant part of the real part of (a) as double precision datum can contain.

4. CMPLX (or DCMPLX) may have one or two arguments. If there is one argument, it must be of type integer, real, double precision, or complex. If there are two arguments, they must both be of the same type and must be of type integer, real, or double precision.

   For an integer, real, or double precision argument:

CMPLX (a)	=	REAL (a) 0E0	real part imaginary part
DCMPLX (a)	=	DBLE (a) 0D0	real part imaginary part

   For a complex argument:

CMPLX (a)	=	a	
DCMPLX (a)	=	DBLE (REAL (a)) DBLE (AIMAG(a))	real part imaginary part

   For a double precision complex argument:

CMPLX (a)	=	REAL (DREAL(a)) REAL (DIMAG(a))	real part imaginary part
DCMPLX(a)	=	a	

For two arguments:

CMPLX (a_1,a_2)	=	REAL(a_1) REAL(a_2)	real part imaginary part
DCMPLX (a_1,a_2)	=	DBLE(a_1) DBLE(a_2)	real part imaginary part

5. ICHAR converts a character to an integer, based on the position of the character in the ASCII collating sequence. The first character in the collating sequence corresponds to position 0 and the last to position n-1, where n is the number of characters in the collating sequence.

   The value of ICHAR (a) is an integer in the range:

   $0 \leq$ ICHAR (a) ≤ 127, where (a) is a character argument of length one. The position of that character in the collating sequence is the value of ICHAR.

   For any characters c_1 and c_2 capable of representation in the processor, (c_1.LE.c_2) is true only if (ICHAR (c_1).LE.ICHAR (c_2)) is true and (c_1.EQ.c_2) is true only if (ICHAR (c_1).EQ.ICHAR (c_2)) is true.

   CHAR (i) returns the ith position of the ASCII collating sequence. The character argument is of length one. i must be an integer expression whose value must be in the range $0 \leq i \leq 127$:

ICHAR (CHAR (i)) = i	for $0 \leq i \leq 127$
CHAR (ICHAR (c)) = c	for any character c capable of representation in the processor.

6. A complex value is expressed as an ordered pair of reals (a_r,a_i), where a_r is the real part and a_i is the imaginary part.

7. INDEX (a_1,a_2) returns an integer value indicating the substring identical to string a_2. If a_2 occurs more than once in a_1, the starting position of the first occurrence is returned.

   If a_2 does not occur in a_1, the value zero is returned. Note that zero is returned if LEN (a_1) < LEN (a_2).

8. All angles are expressed in radians.

9. If the operands for LGE, LGT, LLE, and LLT are of unequal length, the shorter operand is considered as if it were extended on the right with blanks to the length of the longer operand.

10. M.VE causes a character string to move from a specified source to a specified destination. If the source string is longer than the destination, the source string is truncated. If the source string is less than the destination, the source is left justified and padded with blanks.

11. C.MP accomplishes the following (LGE, LGT, LLE, LLT are entries within C.MP):

    (a) Logically compares two character strings and returns the resulting condition codes for compiler generated calls to C.MP.

    (b) Logically compares two character strings in one of the following four ways:

        .LGE., .LGT., .LLE., .LLT.

12. When used in ASSEMBLY, character arguments consist of a pair of parameter words. The first word specifies the address of the first byte of the character item. The second word specifies the address of the length of the character item.

13. When used in ASSEMBLY, CHAR requires three actual parameter words. The first two specify a target character item (in storage) that is to receive the value of CHAR. The third parameter corresponds to the actual argument, I, in CHAR(I). The integer value of I cannot exceed 127.

14. For exception conditions, the returned value differs from the correct return value by less than the margin indicated.

15. If the base value is negative, the absolute value of the result is returned.

16. The following intrinsic functions will accept integer*8 arguments: JABS, JDIM, JINT and JSIGN.

CHAPTER 3

MPX-32 COMPATIBLE MODE SUBROUTINES

3.1 M: and X: Subroutines

This chapter contains descriptions of compatible mode subroutines that provide MPX-32 subroutines to FORTRAN 77+.

The M: subroutines described in this chapter are equivalent to the subroutines under RTM. The X: subroutines described in this chapter are additional compatible mode subroutines. These subroutines must not be used by programs using the native mode file system features.

Any attempt to mix modes; i.e., compatible with native, will result in an RS99 error with the entry point displayed in an extended abort message.

A delimiting comma must be present in the calling sequence of M: subroutines only if an omitted optional argument is followed by other arguments. A delimiting comma must be present in all X: subroutines whenever an optional argument is not included in the calling sequence.

All error codes are expressed as decimal integers.

If a task is multicopied, any later reference to that task must be by task number.

An argument list that does not contain the arguments *label (a denial return address) and an istatus, results in an RSxx abort code message and program termination if the subroutine performs unsuccessfully. If only istatus is specified, the istatus value should be checked to determine if the subroutine performed successfully. If a denial return address is specified and the subroutine performs unsuccessfully, control will transfer to the specified address.

3.1.1 M:ALOC1

The M:ALOC1 subroutine dynamically allocates a previously created permanent file and assigns a Logical File Code (LFC) to it.

Calling Sequence

CALL M:ALOC1 (lfc, filename,[*label],[password],[blocked],[wait],[istatus])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled.

label — The statement label to which control is returned if a request is denied due to file unavailability.

password — An INTEGER*8 variable that specifies the password. This argument is a place holder provided for compatibility and, if specified, is ignored.

blocked — A LOGICAL variable. If present and specified true (.TRUE.), the file is allocated blocked. Unblocked is the default.

wait — A LOGICAL variable. If present and specified true (.TRUE.), the allocation is performed in the wait mode. No-wait is the default. In the wait mode,. the calling task is queued for the requested resource and suspended until the resource becomes available.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Permanent file nonexistent.
2	Illegal file password specified.
3	No File Assignment Table/File Pointer Table (FAT/FPT) space available.
4	No blocking buffer space available.
5	Shared memory table entry not found.
6	Invalid shared memory table password specified.
7	Dynamic common specified in $ASSIGN1.
8	Unrecoverable I/O error to System Master Directory (SMD).
9	System General Object (SGO) assignment specified by terminal task.
10	No User Terminal (UT) file code exists for terminal task.
11	Invalid Resource Requirement Summary (RRS) entry.
13	Assigned device not on system.
14	Device in use by requesting task.
15	SGO or System Control (SYC) assignment not allowed by real-time task.
16	Common memory conflicts with allocated task.
17	Duplicate LFC allocation attempted.
18	Incompatible call.

The following error codes are returned if allocation is requested with no wait and one of the conditions exists. If allocation is requested with wait mode, no error code is returned and the system continues waiting.

30	Permanent file exclusively closed.
31	File Lock Table (FLT) full.
32	Nonshared device busy (already allocated).
33	Disc space is not available.
70	Allocation error.

Programming Considerations

Static LFC assignments (via job control or cataloging) may not be overridden until the LFC has been deallocated, typically with the M:DALOC subroutine.

3.1.2 M:ALOC2

The M:ALOC2 subroutine dynamically allocates a System Listed Output (SLO) or System Binary Output (SBO) file and assigns a Logical File Code (LFC) to the file.

Calling Sequence

CALL M:ALOC2 (lfc, sfc, size,[*label],[wait],[istatus])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

sfc — An INTEGER variable that specifies the system file code (either SLO or SBO). This argument must be three ASCII characters, left justified and blank filled.

size — An INTEGER variable that specifies the allocation required in physical record blocks (192 words each).

label — The statement label to which control is returned if a request is denied because of file unavailability.

wait — A LOGICAL variable. If present and specified true (.TRUE.), the allocation is performed in the wait mode. No-wait is the default.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Permanent file nonexistent.
2	Illegal file password specified.
3	No File Assignment Table/File Pointer Table (FAT/FPT) space available.
4	No blocking buffer space available.
5	Shared memory table entry not found.
6	Invalid shared memory table password specified.
8	Unrecoverable I/O error to System Master Directory (SMD).
9	System General Object (SGO) assignment specified by terminal task.
10	No User Terminal (UT) file code exists for terminal task.
11	Invalid Resource Requirement Summary (RRS) entry.
13	Assigned device not on system.
14	Device in use by requesting task.
15	SGO or System Control (SYC) assignment not allowed by real-time task.
16	Common memory conflicts with allocated task.
17	Duplicate LFC allocation attempted.
18	Incompatible call.

The following error codes are returned if allocation is requested with no-wait and one of the conditions exists. If allocation is requested with wait mode, no error code is returned and the system continues waiting.

30	Permanent file exclusively closed.
31	File Lock Table (FLT) full.
32	Nonshared device busy (already allocated).
33	Disc space not available.
70	Allocation error.

Programming Considerations

Static LFC assignments (via job control or cataloging) may not be overridden until the LFC has been deallocated, typically with the M:DALOC subroutine.

SGO and SYC files may not be assigned with this subroutine.

SLO and SBO are blocked files.

3.1.3 M:ALOC3

The M:ALOC3 subroutine dynamically allocates a physical device and assigns a Logical File Code (LFC) to the device.

Calling Sequence

CALL M:ALOC3 (lfc, dt,[channel],[param],[*label],[mo],[vol],[blocked],[wait],[istatus],[da])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

dt — An INTEGER variable or constant that specifies the device type code (refer to Chapter 5). If the dt parameter is a constant, it must represent a value corresponding to one specified in Section 5.2. For example, the following calling sequence would cause a line printer to be allocated:

```
CALL M:ALOC3(OUT,10,2Z7A,,*40)
```

channel — An INTEGER variable that specifies the channel number (hexadecimal) of the particular device on which the file is to reside. Channel is a configuration restraint established at the time of system generation (SYSGEN) and configuration.

param — An INTEGER variable that specifies the four-ASCII-character reel ID, if the device is magnetic tape, or the number of 192-word blocks to allocate, if the device is a disc. Otherwise, the value of param is null.

label — The statement label to which control is returned if allocation is denied because of device unavailability.

mo — An INTEGER variable or constant that specifies the mount-only option for magnetic tapes:

0	Allocate and mount
1	Mount only

vol — An INTEGER variable or constant that specifies the magnetic tape volume number (1 through 255).

blocked — A LOGICAL variable. If present and specified true (.TRUE.), the peripheral device or file is allocated blocked. Unblocked is the default.

wait — A LOGICAL variable. If present and specified true (.TRUE.), the allocation is performed in the wait mode. No-wait is the default.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Permanent file nonexistent.
2	Illegal file password specified.
3	No File Assignment Table/File Pointer Table (FAT/FPT) space available.
4	No blocking buffer space available.
5	Shared memory table entry not found.
6	Invalid shared memory table password specified.
8	Unrecoverable I/O error to System Master Directory (SMD).
9	System General Object (SGO) assignment specified by terminal task.
10	No User Terminal (UT) file code exists for terminal task.
11	Invalid Resource Requirement Summary (RRS) entry.
13	Assigned device not on system.
14	Device in use by requesting task.
15	SGO or System Control (SYC) assignment not allowed by realtime task.
16	Common memory conflicts with allocated task.
17	Duplicate LFC allocation attempted.
18	Call was incompatible.

The following error codes are returned if allocation is requested with no wait and one of the conditions exists. If allocation is requested with wait mode, no error code is returned and the system continues waiting.

30	Permanent file exclusively closed.
31	File Lock Table (FLT) full.
32	Nonshared device busy (already allocated).
33	Disc space not available.
70	Allocation error

da An INTEGER variable that specifies the device subaddress. The default value is zero.

Programming Considerations

A message, MOUNT TAPE (reel identification) (volume) ON UNIT (magnetic tape device number), is written to the operator's console for magnetic tape use.

Static LFC assignments (via job control or cataloging) may not be overridden until the LFC has been deallocated, typically with the M:DALOC subroutine.

If the device is a disc drive, a formatted volume must be mounted. A temporary file will be created by the subroutine. The disc drive is treated as a formatted medium and a mount message will not be issued.

3.1.4 M:ALOC4

The M:ALOC4 subroutine equates a new Logical File Code (LFC) to an existing logical file code.

Calling Sequence

CALL M:ALOC4 (lfc, pfc,[*label],[istatus])

lfc — An INTEGER or CHARACTER expression or variable that specifies the new logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

pfc — An INTEGER or CHARACTER expression or variable that specifies the existing logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

label — A statement label to which control is returned if a request is denied.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Permanent file nonexistent.
2	Illegal file password specified.
3	No File Assignment Table/File Pointer Table (FAT/FPT) space available.
4	No blocking buffer space available.
5	Shared memory table entry not found.
6	Invalid shared memory table password specified.
8	Unrecoverable I/O error to System Master Directory (SMD).
9	System General Object (SGO) assignment specified by terminal task.
10	No User Terminal (UT) file code exists for terminal task.
11	Invalid Resource Requirement Summary (RRS) entry.
13	Assigned device not on system.
14	Device in use by requesting task.
15	SGO or System Control (SYC) assignment not allowed by realtime task.
16	Common memory conflicts with allocated task.
17	Duplicate LFC allocation attempted.
18	Incompatible call.

The following error codes are returned if allocation is requested with no wait and one of the conditions exists. If allocation is requested with wait mode, no error code is returned and the system continues waiting.

30	Permanent file exclusively closed.
31	File Lock Table (FLT) is full.
32	Nonshared device busy (already allocated).

33	Disc space not available.
70	Allocation error.

Programming Consideration

Static LFC assignments (via job control or cataloging) may not be overridden until the LFC has been deallocated, typically with the M:DALOC subroutine.

3.1.5 M:ALOC5

The M:ALOC5 subroutine dynamically allocates a permanent system file and assigns a Logical File Code (LFC) to the file.

Calling Sequence

CALL M:ALOC5 (lfc, filename,[*label],[password],[blocked],[wait],[istatus])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

filename — An INTEGER*8 variable that specifies the permanent file name. The file name must be one to eight ASCII characters, left justified and blank filled.

label — The statement label to which control is returned if a request is denied due to file unavailability.

password — An INTEGER*8 variable that specifies the password. This argument is a place holder provided for compatibility and, if specified, is ignored.

blocked — A LOGICAL variable. If present and specified true (.TRUE.), the peripheral device or file is allocated blocked. Unblocked is the default.

wait — A LOGICAL variable. If present and specified true (.TRUE.), the allocation is performed in the wait mode. No-wait is the default.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Permanent file nonexistent.
2	Illegal file password specified.
3	No File Assignment Table/File Pointer Table (FAT/FPT) space available.
4	No blocking buffer space available.
5	Shared memory table entry not found.
6	Invalid shared memory table password specified.
8	Unrecoverable I/O error to System Master Directory (SMD).
9	System General Object (SGO) assignment specified by terminal task.
10	No User Terminal (UT) file code exists for terminal task.
11	Invalid Resource Requirement Summary (RRS) entry.
13	Assigned device not on system.
14	Service in use by requesting task.
15	SGO or System Control (SYC) assignment not allowed by real-time task.
16	Common memory conflicts with allocated task.

17 Duplicate LFC allocation attempted.
18 Incompatible call.

The following error codes are returned if allocation is requested with no wait and one of the conditions exists. If allocation is requested with wait mode, no error code is returned and the system continues waiting.

30 Permanent file is exclusively closed.
31 File Lock Table (FLT) full.
32 Nonshared device busy (already allocated).
33 Disc space not available.
70 Allocation error.

Programming Consideration

Static LFC assignments (via job control or cataloging) may not be overridden until the LFC has been deallocated, typically with the M:DALOC subroutine.

3.1.6 M:CLOSE

The M:CLOSE subroutine will close a file associated with a specified Logical File Code (LFC).

Calling Sequence

```
CALL M:CLOSE (lfc)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

3.1.7 M:CREATE

The M:CREATE subroutine creates a permanent file that remains defined to the system until it is deleted.

Calling Sequence

CALL M:CREATE (filename, blocks, devtype,[channel],[restr],[password],[sys],[nosav], [speed],[type],[null],[zeroing],[istatus],[da])

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled.

blocks — An INTEGER variable that specifies the size of the file as a multiple of 192-word blocks.

devtype — An INTEGER variable that specifies the type of disc on which the file is to reside. The end of Chapter 5 contains valid disc device type codes. Note that the device type code is an integer, not the two-character device mnemonic.

channel — An INTEGER variable that specifies the channel number of the particular device on which the file is to reside.

restr — An INTEGER variable that specifies the file access restrictions. This argument is a place holder provided for compatibility and, if specified, is ignored.

password — This argument is a place holder provided for compatibility and, if specified, is ignored.

sys — An INTEGER variable that specifies the type of file. This argument must be a single ASCII character, left justified and blank filled. Options are:

'S'	System file
Default	User file (system file if a user name is not associated with the calling program)

nosav — An INTEGER variable that specifies if the file is to be saved. This argument must be a single ASCII character, left justified and blank filled. Options are:

'N'	File is not saved without special FILEMGR or VOLMGR option
Default	File is saved

speed — An INTEGER variable that specifies the speed of the file. This argument must be a single ASCII character, left justified and blank filled. Options are:

'F'	Fast file
Default	Slow file

A file specified as a fast file is one for which the file definition will be retrieved from the System Master Directory (SMD) with one access. If the file is specified as slow, the scatter storage mechanism that builds SMD entries uses backup algorithms if the file name maps into an existing active file (collision mapping). Each backup algorithm used requires an additional disc access.

type — A two-digit (one byte) hexadecimal value you specify to associate a type with the file being created. Some examples of system file types are:

ED	Editor save file
EE	Editor store file
FE	Editor work file
FF	SYSGEN-created file
BA	BASIC file
CA	Cataloged load module

null — A null parameter that must be specified in the form of ,, if additional parameters are to be used after this point.

zeroing — An INTEGER variable that specifies whether allocated disc space is to be zeroed. This argument must be a single ASCII character, left justified and blank filled. Options are:

'Z'	Zeroing
Default	No zeroing

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	File already exists.
2	Fast file specified and collision mapping occurred with an existing directory entry.
3	Restricted access specified, but no password entered.
4	Disc space unavailable.
6	Specified device off-line.
7	Directory is full.
8	Specified device type not configured.
10	Access denied.

da — An INTEGER variable that specifies a particular device subaddress, as defined at SYSGEN time, on which the file is to reside.

3.1.8 M:DALOC

The M:DALOC subroutine deallocates a peripheral device or disc file to which the specified Logical File Code (LFC) is assigned. Dynamic deallocation of a peripheral or permanent file releases that resource to other tasks. Deallocation of a temporary file deletes the file. Therefore, use M:PERM to change the status of a temporary file to permanent if you want the file to remain accessible. Deallocation of System Listed Output (SLO) or System Binary Output (SBO) files passes their definitions to system output to be written to their terminal devices. This subroutine deallocates only the specified code when that logical file code has been equated to other logical file codes in the system.

Calling Sequence

```
CALL M:DALOC (lfc)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

Programming Considerations

If I/O is in progress for the requested file, the file is closed automatically.

If the device specified by the logical file code is magnetic tape and only a dismount message (i.e., without deallocation) is required, the LFC parameter should be a four-character specification with the last character an asterisk (e.g., 'ABC*', '4ØØ*', 'A4Ø*').

3.1.9 M:DELETE

The M:DELETE subroutine deletes a specified permanent file or a dynamic (non-SYSGEN created) memory partition.

Calling Sequence

```
CALL M:DELETE (filename,[sys],[password],[istatus])
```

filename — An INTEGER*8 variable that specifies the permanent file name or memory partition. This argment must be one to eight ASCII characters, left justified and blank filled.

sys — An INTEGER variable that specifies the type of file. This argument must be a single ASCII character, left justified and blank filled. Options are:

'S'	System file
Default	User file (system file if user name is not associated with the calling program)

password — An INTEGER*8 variable that specifies the password. This argument is a place holder provided for compatibility and, if specified, is ignored.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	A file of the specified name either does not exist or is a static (SYSGEN created) memory partition.
2	A required password was not specified.

3.1.10 M:LOG

The M:LOG subroutine provides a log of current permanent files and memory partitions. The log is stored as an eight-word System Master Directory (SMD) entry at the address you specify. The SMD entry is copied into the specified array. The password field of the SMD entry contains zero or one to indicate either the absence or presence of a password, respectively.

Calling Sequence

CALL M:LOG (type, smdbuff, filename, status)

type — An INTEGER variable that specifies the type of log to be performed. The contents of this argument are altered by the subroutine. This argument's value must be left justified and blank filled. Values are:

'N'	A single-named system or user file
'O'	A single-named system file
'A'	All permanent user files
'S'	System files only
'U'	User files

If 'N' is specified and a user name is associated with the calling task, an attempt is made to locate the user file directory entry for the given file name. If the user file directory entry cannot be found, then an attempt is made to locate the system file directory entry, if one exists. If a user name is not associated with the calling task, the file is assumed to be a system file.

If 'U' is specified and the calling task has an associated user name, that user's files are logged. All files are logged if the calling task has no associated user name.

This argument cannot be a constant. The M:LOG subroutine destroys the contents of the type argument. Therefore, you must establish the contents before the initiation of a log sequence.

smdbuff — An INTEGER array consisting of eight elements in which the SMD entry is to be stored.

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled. Filename is required if type is 'N' or 'O'.

status — An INTEGER variable within the calling task that will be zeroed by this subroutine if type is 'N' or 'O' and the file cannot be located, or if type is 'A', 'S', or 'U' and all pertinent files have been logged.

Programming Consideration

If type is 'A', 'S', or 'U', this subroutine must be called repeatedly to obtain all pertinent file definitions. The word within the calling program (defined as status in the argument list) will be set to zero when all pertinent files have been logged.

3.1.11 M:OPEN

The M:OPEN subroutine opens the resource associated with the specified Logical File Code (LFC).

Calling Sequence

```
CALL M:OPEN (lfc)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

3.1.12 M:PDEV

The M:PDEV subroutine returns physical device information describing the unit connected to the specified Logical File Code (LFC).

Calling Sequence

CALL M:PDEV (lfc, i)

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

i — An INTEGER*2 array consisting of eight elements that contains, upon return, the physical device information for the specified unit.

Results

i(1) A device mnemonic; see the end of Chapter 5.

i(2) The byte count. The maximum number of bytes per physical record that is transferable to this device.

i(3) A device type code; see the end of Chapter 5.

i(4) The device address.

i(5) A system file type code, as follows:

0	Not a system file
1	System Control (SYC)
2	System General Object (SGO)
3	System Listed Output (SLO)
4	System Binary Output (SBO)

i(6) Zero if the LFC is not assigned to a disc, tape, or Terminal Services Manager (TSM) device. If the LFC is assigned to a disc, the number of 192-word blocks in the file is returned. If the file is assigned to a magnetic tape, the rightmost two characters of the reel identifier (ID) are returned. If the file is assigned to a TSM device, the line size (byte 0) and page size (byte 1) are returned.

i(7) The device subaddress.

i(8) Extended device flag.

0	All other device classes
1	Extended I/O (class F) device

Programming Consideration

If the file is an unopened SYC or SGO file, i(5) is returned equal to one or two, respectively. All other elements of i are meaningless. If the file code is unassigned, i(3), i(4), i(5), and i(6) will return equal to 0.

3.1.13 M:PERM

The M:PERM subroutine changes the status of a file allocated to the calling program from temporary to permanent. The file must be an open temporary, System Listed Output (SLO), or System Binary Output (SBO) file.

Calling Sequence

```
CALL M:PERM  (filename, lfc,[restr],[password],[sys],[nosav],[speed],[type],
              [zeroing],[istatus])
```

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

restr — An INTEGER variable that specifies the file access restrictions. This argument is a place holder provided for compatibility and, if specified, is ignored.

password — This argument is a place holder provided for compatibility and, if specified, is ignored.

sys — An INTEGER variable that specifies the type of file. This argument must be a single ASCII character, left justified and blank filled. Options are:

'S'	System file
Default	User file (system file if a user name is not associated with the calling program)

nosav — An INTEGER variable that specifies if the file is to be saved. This argument must be a single ASCII character, left justified and blank filled. Options are:

'N'	File is not saved without special FILEMGR or VOLMGR option
Default	File is saved

speed — An INTEGER variable that specifies the speed of the file. This argument must be a single ASCII character, left justified, and blank filled. Options are:

'F'	Fast file
Default	Slow file

Fast and slow refer to the speed of file definition retrieval. Under most circumstances the file definition is retrieved a minimal number of times. Therefore, a slow file should be used. If it is necessary to retrieve the file definition repeatedly (i. e., allocating and deallocating a file several times within a task), a fast file should be used.

type — A two-digit (one byte) hexadecimal value which specifies the file type. Some examples of system file types are:

ED	Editor save file
EE	Editor store file
FE	Editor work file
FF	SYSGEN-created file
BA	BASIC file
CA	Cataloged load module

zeroing — An INTEGER variable that specifies whether the allocated disc space is to be zeroed. This argument's value must be a single ASCII character, left justified, and blank filled. Options are:

'Z'	Zeroing
Default	No zeroing

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

1	File of the name specified already exists.
2	Fast file specified and collision mapping occurred with an existing directory entry.
3	Restricted access specified, but no password entered.
4	The file associated with the specified logical file code is not a temporary file.
5	Directory and temporary file are not on the same volume.
7	The directory is full.
9	File name or password contains invalid characters or embedded blanks.

3.1.14 M:PFAD

The M:PFAD subroutine returns information necessary to access memory partitions or disc files directly. Specifically, the subroutine provides either the word address of the beginning of a memory partition or the track, head, or sector address (in the format used by Assembly Language Command Device (CD) and Test Device (TD) instructions) of a disc file.

Calling Sequence

```
CALL M:PFAD (filename, i, *label)
```

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled.

i — An INTEGER*8 variable or a two-word INTEGER array defined as follows:

Memory Partition

i(1)	Word address of first word of file
i(2)	Bit 0 = 1 Bits 1-15 = 0 Bits 16-31 = number of pages allocated to the partition

Disc File

i(1)	Bits 0-5 = 0 Bits 6-12 = device address Bits 13-22 = 0 Bits 23-27 = head number Bits 28-31 = sector number
i(2)	Bits 0-5 = 0 Bits 6-12 = device address Bits 13-22 = 0 Bits 23-31 = track number

label — The statement label to which control is returned if the request is denied because the specified file cannot be found in the System Master Directory (SMD).

3.1.15 M:USER

The M:USER subroutine associates a user name with the calling program. Optionally, this subroutine nullifies any user name associated with the calling program. The user name associated with the program is used in any file create, delete, log and allocate subroutines subsequently called.

Calling Sequence

```
CALL M:USER ([username],[key], istatus)
```

username — An INTEGER*8 variable that specifies a one to eight character left justified blank filled directory name on the current volume only. If omitted or zero, this will default to the system directory if present on the current volume or return an error status if not.

key — Is ignored if specified.

istatus — An INTEGER variable set according to the results of the subroutine as follows:

0	Normal completion.
1	Service not performed.

3.1.16 X:EXCL

The X:EXCL subroutine dynamically deallocates any memory partition a task has previously either shared or included and decrements the use count for the memory partition by one. The memory partition is deleted when its use count is zero.

Calling Sequence

CALL X:EXCL (partition, ownername)

partition — An INTEGER*8 or CHARACTER variable that specifies the memory partition name. This argument must be one to eight ASCII characters, left justified and blank filled.

ownername — An INTEGER*8 or CHARACTER variable that specifies the memory partition owner. Ownername must be one to eight ASCII characters, left justified and blank filled.

3.1.17 X:INCL

The X:INCL subroutine dynamically includes a memory partition (i.e., a global common, datapool, or other memory partition) in a task's address space. The task must know the owner name of this memory partition. The use count for this memory partition is incremented by one.

Calling Sequence

CALL X:INCL (partition, ownername,[restr],[password],[istatus],[*label])

partition — An INTEGER*8 or CHARACTER variable that specifies the memory partition name. This argument must be one to eight ASCII characters, left justified and blank filled.

ownername — An INTEGER*8 or CHARACTER variable that specifies the owner name that issued an X:SHARE subroutine call. This argument must be one to eight ASCII characters, left justified and blank filled.

restr — An INTEGER variable that specifies the memory partition access restrictions. This argument must be one or two ASCII characters, left justified and blank filled. Options are:

'RW'	Read/write access (default)
'R'	Read access only

password — An INTEGER*8 or CHARACTER variable that specifies a password. This argument is a place holder provided for compatibility and, if specified, is ignored.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Entry not found in shared memory table.
2	Invalid password specified.
3	Memory requirements conflict with task's address space.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

3.1.18 X:SHARE

The X:SHARE subroutine dynamically allocates a shared memory partition from the partition definition found in the system directory. This definition must have been previously defined via the Volume Manager utility.

The call results in the allocation of a new common area, which will be uniquely identified by the owner name or task number of the caller, and by the memory partition name. The memory type will be specified by the directory definition. Pre-zeroing is not performed by this service. The partition is swappable with the task if the use count equals zero. The partition is deallocated when the allocation count equals zero. The task is suspended until the Shared Memory Table entry is built and the memory allocation is complete.

Calling Sequence

CALL X:SHARE (partition,[restr],[password])

partition — An INTEGER*8 or CHARACTER variable that specifies the memory partition name. This argument must be one to eight ASCII characters, left justified and blank filled.

restr — An INTEGER variable that specifies the file access restrictions. This argument must be one or two ASCII characters, left justified and blank filled. Options are:

'RW'	Read/write access (default)
'R'	Read access only

password — An INTEGER*8 or CHARACTER variable that specifies a password. This argument is a plcae holder provided for compatibility and, if specified, is ignored.

CHAPTER 4

MPX-32 COMPATIBLE OR NATIVE MODE SUBPROGRAMS

4.1 M:, X: and Named Subprograms

This chapter contains descriptions of subprograms that provide MPX-32 services to FORTRAN 77+ programs. These subprograms are available to MPX-32 users in either compatible or native mode.

The following general notes refer to these system subprograms under MPX-32:

- Any attempt to mix modes; i.e., compatible with native, will result in an RS99 error with the entry point displayed in an extended abort message.

- Run-time subroutines are not reentrant. Use caution when processing break interrupts during I/O routines.

- FORTRAN users must process software interrupts synchronously by calling X:SYNCH or unpredictable results occur. No-wait end action routines and message end action routines are handled synchronously by X:SYNCH. Either X:ANYW or X:EAWAIT passes control to these routines.

- Break interrupts are always processed asynchronously, even if the synchronous processing mode has been established (X:SYNCH). Because run-time routines are not reentrant, a task must not invoke any run-time routines from within a break receiver if the task is to resume processing at the end of its break interrupt processing routine.

- If a task is multicopied, any later reference to that task must be by task number.

- An argument list that does not contain the arguments *label (a denial return address) and an istatus, results in an RSxx abort code message and program termination if the subroutine performs unsuccessfully. If only istatus is specified, the istatus value should be checked to determine if the subroutine performed successfully. If a denial return address is specified and the subroutine performs unsuccessfully, control will transfer to the specified address.

  For those SRTL routines that have possible istatus values of 50, 51, 52, 53 where istatus and *label are omitted, the current task will abort with an RTxx rather than an RSxx for those values.

- A delimiting comma must be present in the calling sequence of all M: subprograms and X: subprograms if an omitted optional argument is followed by other arguments. Delimiting commas which lie outside the brackets for optional arguments are also required.

- All error codes are expressed as decimal integers.

4.2 M: Subprograms

The subprograms described in this section begin with an M: prefix. This indicates they are RTM equivalent subprograms available to the MPX-32 user.

4.2.1 M:ABORT

The M:ABORT subroutine allows a task to abort any task. The named task is not aborted until it gains CPU control. If the specified task is not in execution, the request is ignored. To force I/O completion and abort the specified task immediately, use X:DELTSK.

Calling Sequence

```
CALL M:ABORT (itask, icode)
```

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be an INTEGER*8 variable of one to eight ASCII characters, left justified and blank filled. A task number must be right justified and zero filled. Specifying a task name of zero (not ASCII) will abort the calling task.

icode — An INTEGER variable that specifies an abort code consisting of four ASCII characters.

4.2.2 M:ACTIV

The M:ACTIV subroutine activates a specified task. The task assumes the ownername of the caller. The load module must reside in the system directory.

Calling Sequence

```
CALL M:ACTIV (taskname,[istatus],[*label])
```

taskname — An INTEGER*8 variable that specifies the file name for a load module in the system directory. This argument must be one to eight ASCII characters, left justified and blank filled. When running in native mode for load modules not residing on the system, the first word of taskname should contain binary zeros and the second word should contain a pathname vector to the load module.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Invalid attempt to multicopy a unique task.
2	Specified file not in System Directory.
3	File is password protected.
4	File does not contain valid data.
5	No dispatch queue entry available.
6	Read error on System Directory.
7	Read error on load module.
8	No free Map Image Description List (MIDL) space.
9	Insufficient memory.
10	No physical memory available.
50	Missing parameter (e.g., taskname).

label — A statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes; or RTxx, where xx is an istatus code of 50.

4.2.3 M:BLOCK

In compatible mode the M:BLOCK subroutine defines a particular file code as a blocked file. A File Control Block (FCB) is established and bit 5, word 3 of the FCB is set to indicate blocking. The maximum physical record size on a blocked file is 254 bytes. While the MPX-32 Operating System defaults all files to blocked, the Scientific Run-Time Library defaults all files to unblocked.

This subroutine is included in the native mode library but does not define the file code as a blocked file. Use the FORTRAN 77+ OPEN statement to perform blocking in native mode.

Calling Sequence

```
CALL M:BLOCK (lfc)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

Programming Considerations

Despite the fact that blocked is the default under MPX-32, the FORTRAN 77+ environment assumes that files are unblocked.

It is necessary to call M:BLOCK only once for a file. The call must be made prior to any I/O to the logical file code.

The FORTRAN 77+ OPEN statement also provides this function.

4.2.4 M:CLOCK

The M:CLOCK subroutine provides the current time of day as computed from the real-time clock in seconds and number of interrupts.

Calling Sequence

CALL M:CLOCK (iseconds, interrupt)

iseconds — An INTEGER variable that contains, upon return, the value of the current time in seconds.

interrupt — An INTEGER variable that contains, upon return, the number of interrupts since the last second.

4.2.5 M:CONECT

The M:CONECT subroutine indirectly connects a task to an interrupt level so that when the interrupt occurs, the program can be scheduled for execution (resumed). If the task is not active at the time of the call, the task will be preactivated depending on the parameters being passed and then connected to the interrupt.

Note: M:CONECT and M:CONRES are routines that were retained from SRTL running under the RTM operating system. They formerly called different RTM services to accomplish their functions. MPX-32 combined the functions of the two RTM services into one. This gave both M:CONECT and M:CONRES identical functionality. These routines are being maintained for compatibility.

Calling Sequence

CALL M:CONECT(taskname, priority, interrupt, [istatus] , [*label])

Refer to M:CONRESS for parameter descriptions.

4.2.6 M:CONRES

The M:CONRES subroutine indirectly connects a task to an interrupt level so that when the interrupt occurs, the program can be scheduled for execution (resumed). If the task is not active at the time of the call, the task will be preactivated depending on the parameters being passed and then connected to the interrupt.

Note: M:CONECT and M:CONRES are routines that were retained from SRTL running under the RTM operating system. They formerly called different RTM services to accomplish their functions. MPX-32 combined the functions of the two RTM services into one. This gave both M:CONECT and M:CONRES identical functionality. These routines are being maintained for compatibility.

Calling Sequence

CALL M:CONRES(taskname, priority, interrupt, [istatus] , [*label])

taskname — An INTEGER*8 variable that specifies the task name or task number for tasks that are already active, or a load module name (or pathname vector for native mode operation) for tasks that are not yet active. A task name must be one to eight ASCII characters, left justified and blank filled. If the task is not yet active, the taskname is assumed to be the load module name in the system directory. A task number must be right justified in the doubleword and zero filled. If the entire doubleword is zero filled, the calling task is assumed. If the task to be connected is not yet active and calling task is operating in native mode, then the first word of the doubleword must contain a pathname vector and the second word of the doubleword must be zero. In this case, the load module need not be in the system directory.

priority — An integer variable that is now only a place holder for compatibility purposes. Its value is not used by the service, but it must be present.

interrupt — An integer variable that specifies the external hardware priority level to which the program is to be connected (e.g., 2A-7E).

istatus — An integer variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0 Normal completion.
1 Task already connected to an interrupt.
2 Another task already connected to the specified interrupt.
3 Interrupt not SYSGEN specified indirectly connectable.
4 Specified task number not found in dispatch queue or the requesting task is not privileged and the ownername is restricted from access to tasks with a different ownername (via the M.KEY file).

label — The statement label to which control is returned if an error exists.

Programming Considerations

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

The external interrupt level must be designated as 'ITLB' at SYSGEN time to be eligible for having tasks indirectly connected.

The M:CONRES subroutine preactivates the connecting task if it is not already active and if one of the following situations occurred.

- Taskname was specified and the load module for the connecting task is accessible on the system directory.
- A pathname vector was supplied (applies to native mode only).

If M:CONECT or M:CONRES are required to preactivate a task, but the actual interrupt connection is denied, the user is responsible for deleting the residual task. Otherwise, the task will continue in the suspended state indefinitely.

Only one task may be indirectly connected to an external interrupt.

4.2.7 M:CORE

The M:CORE function provides the beginning and ending word addresses of memory that is allocated to a user.

Calling Sequence

```
iaddress = M:CORE (loc)
```

iaddress — An INTEGER variable that is set to the high or low address, depending on the value of the argument loc.

loc — An INTEGER variable that specifies which memory location should be returned. This argument's value must be two ASCII characters, left justified and blank filled. Options are:

'HI'	Places user's last memory location in iaddress
'LO'	Places user's beginning memory location in iaddress
Default	'LO'

4.2.8 M:DELTIM

The M:DELTIM subroutine deletes the timer table entry for a specified timer.

Calling Sequences

CALL M:DELTIM (id)

id — An INTEGER variable that specifies the timer entry identification. This argument must be two ASCII characters, left justified and blank filled.

Programming Considerations

Invalid requests are ignored.

Deletion of a timer entry will not delete the associated task.

One shot timers are deleted upon expiration.

4.2.9 M:DUMP

The M:DUMP subroutine provides a listed dump of the caller's Program Status Doubleword (PSD), general-purpose registers, and specified memory limits. Output is to the default spooled output device (specified at SYSGEN) in side-by-side hexadecimal with ASCII format. The PSD and registers precede the specified memory limits. The PSD and registers in the dump contain their values at the time of the M:DUMP call. Any task may request a memory dump.

Calling Sequence

CALL M:DUMP (istart, iend)

istart — An INTEGER*4 variable that specifies the low logical word address of the first location to be dumped.

iend — An INTEGER variable that specifies the high logical address of the last location to be dumped.

Programming Consideration

The start and end addresses are adjusted to inclusive eight-word boundaries prior to the dump.

4.2.10 M:END

The M:END subroutine calculates the total elapsed time from the last call to subroutine M:START to this call, M:END. The total time is written to a System Listed Output (SLO) file. The user may not reassign to another file.

Calling Sequence

CALL M:END

4.2.11 M:ERRFLG

The M:ERRFLG subroutine allows user programs to continue operation if minor run-time errors (i.e., errors found within the run time, not within the operating system) are encountered during execution. The subroutine initializes the variable to 0 (.FALSE.). Run-time routines then set the variable to 1 (.TRUE.) if they encounter minor run-time errors during execution. Minor run-time errors are listed in Appendix B.

Calling Sequence

```
CALL M:ERRFLG (ierror)
```

ierror — An INTEGER or LOGICAL variable that is set to 1 (.TRUE.) if minor run-time errors occur. The variable ierror cannot be type INTEGER*1 or type LOGICAL BIT.

4.2.12 M:HOLD

The M:HOLD subroutine holds a specified program until OPCOM CONTINUE is issued. A HOLD bit is turned on in the Dispatch Queue Entry (DQE) for the program, and the program's current status is retained so that it can be continued from the point it was held.

Calling Sequence

```
CALL M:HOLD (itask, *label)
```

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified within the doubleword and zero filled. Specifying a task name of zero (not ASCII) will hold the caller's program.

label — The statement label to which control is returned if the program is not in the CPU dispatch queue.

4.2.13 M:IOEX

The M:IOEX subroutine permits both the privileged and unprivileged user to designate a statement label to which control will be transferred on normal job termination or on abort processing.

Calling Sequence

CALL M:IOEX (*$label_1$, *$label_2$)

$label_1$ The statement label to which control is transferred upon either normal job termination or on abort processing.

$label_2$ The statement label to which control is returned if an unprivileged user has specified an address outside the user's allocated area.

Programming Considerations

No address validation is provided for the privileged user.

Unprivileged users can enter abort receivers only once.

User files remain open until task exit occurs.

4.2.14 M:IOLEN

The M:IOLEN function returns the transfer count after a read or write. The transfer units returned depend upon the access mode (sequential or direct), the units of the corresponding device type, and whether the I/O operation is blocked or unblocked.

Calling Sequence

itrancnt = M:IOLEN (lfc)

itrancnt The INTEGER variable transfer count after a read or write.

lfc An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

4.2.15 M:LINKJ

The M:LINKJ subroutine links a disc file into the dynamic job stream queue and activates the system input program (J.SSIN) for job processing.

Calling Sequence

CALL M:LINKJ (filename,[*label],[password])

filename — An INTEGER*8 variable that specifies the permanent file name. This argument must be one to eight ASCII characters, left justified and blank filled. If a user name is in effect, it will be applied to the file name search.

label — The statement label to which control is returned if the subroutine cannot be performed.

password — An INTEGER*8 variable that specifies the password. The password must be one to eight ASCII characters, left justified and blank filled.

4.2.16 M:LOAD

The M:LOAD subroutine loads an overlay segment into the transient area specified to the cataloger.

Calling Sequence

CALL M:LOAD (loadmodule, address)

loadmodule — An INTEGER*8 variable that specifies the load module. This argument must be one to eight ASCII characters, left justified and blank filled.

address — An INTEGER variable that returns the transfer address of an overlay segment.

4.2.17 M:LOADX

The M:LOADX subroutine loads an overlay segment into the transient area specified to the cataloger and executes the task.

Calling Sequence

CALL M:LOADX (loadmodule)

loadmodule — An INTEGER*8 variable that specifies the loadmodule. This argument must be one to eight ASCII characters, left justified and blank filled.

4.2.18 M:PGOPT

The M:PGOPT subroutine provides the caller with the 32-bit task option word.

Calling Sequence

```
CALL M:PGOPT(ioption)
```

ioption — An INTEGER variable. Bits 0 through 31 (numbered left to right) correspond to user-defined options 32 through 1, respectively. These bits are set via the job control directive $OPTION.

Programming Consideration

Options 7 and 8 are used by FORTRAN 77+ run-time support routines.

4.2.19 M:PR

The M:PR subroutine dynamically alters either a caller's priority level or the priority level of another task, temporarily or permanently. Valid priority levels for real-time tasks are 1-54 inclusive. Valid priority levels for time distribution tasks are 55-64 inclusive. A real-time task cannot be changed to a time distribution priority level and a time distribution task cannot be changed to a real-time priority level. I/O continues to operate at the base priority level of the cataloged task. Tasks using this service must be privileged.

Calling Sequence

```
CALL M:PR (itask, iprior, *label)
```

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled. Specifying a task name of zero (not ASCII) alters the caller's priority level.

iprior — An INTEGER variable that specifies a priority level to be assigned to the task. (1-54 for a real-time task; 55-64 for a time distribution task.)

label — The statement label to which control is returned if the specified task is not in execution.

Abort Case

RX06 — Unprivileged task has attempted to reset a task priority level or a privileged task has attempted to reset a task priority to a level outside the range of 1 to 64, inclusively.

4.2.20 M:RSUM

The M:RSUM subroutine resumes a suspended task. The suspended task must have been cataloged into the system directory.

Calling Sequence

```
CALL M:RSUM (itask, *label)
```

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled.

label — The statement label to which control is returned if the task is not in the CPU dispatch queue.

Programming Considerations

If the task is not suspended, the request is ignored.

Both the task making the M:RSUM call and the task being resumed must have the same ownername.

4.2.21 M:RTN

The M:RTN subroutine permits return to the root from an overlay after the overlay has completed execution.

Calling Sequence

```
CALL M:RTN
```

Programming Consideration

M:LOADX must bring the overlay into execution. Return is to one word beyond the call to M:LOADX.

4.2.22 M:SET

The M:SET subroutine modifies a task's user status word. The user status word resides in the CPU dispatch queue and has a value of zero until modified by this subroutine. The user status word is removed from the queue, modified as specified, and replaced in the queue. Bit 0 is never set.

Calling Sequence

```
CALL M:SET (ifunction, istatus , itask )
```

ifunction — An INTEGER variable that specifies one of four function codes which determines the type of modification to be performed. Possible function codes are:

1	Set flag
2	Reset flag
3	Set counter
4	Increment counter

istatus — An INTEGER variable that specifies one of four values, depending on the function code (i.e., if function = 1, istatus indicates the bit position).

Function Code	istatus Value
1	Bit position in the status word to be set (1 to 31 numbered left to right)
2	Bit position in the status word to be reset (1 to 31 numbered left to right)
3	Value to which the status word is to be set
4	Value by which the status word is to be incremented

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled. A task name of zero (not ASCII) specifies the calling task's user status word.

4.2.23 M:SSPND

The M:SSPND subroutine suspends the calling task for either a specified number of time units or an indefinite time, as requested (the time unit is set at SYSGEN). A one-shot timer entry resumes a task suspended for a specified time interval. M:RSUM resumes either a task suspended for an indefinite time interval or a task suspended for a specified number of time units that have not yet expired. To suspend another task, use the subroutine X:SUSP.

Calling Sequence

```
CALL M:SSPND (itime, *label)
```

itime — An INTEGER variable interpreted as follows:

- 0 — A suspension for an indefinite period of time.
- +n — The positive number of units to elapse before the caller is resumed. The actual time elapsed can vary by one time unit because of system design.

label — The statement label to which control is returned if timed suspension is requested and no timer entries are available.

4.2.24 M:START

The M:START subroutine stores the current value of the real-time clock for subsequent calculation of elapsed time by M:END.

Calling Sequence

```
CALL M:START
```

4.2.25 M:TDAY

The M:TDAY subroutine returns the time of day as computed from the real-time clock interrupt counter. The counter is initialized by a SYSGEN parameter. The clock rolls over at midnight.

Calling Sequence

```
CALL M:TDAY (itime)
```

itime — An INTEGER*1 array that contains, upon return, the following four elements:

itime(1)	Hours (0-23)
itime(2)	Minutes (0-59)
itime(3)	Seconds (0-59)
itime(4)	Number of interrupts (within current second)

4.2.26 M:TELER

The M:TELER subroutine allows reads from the operator's console.

Calling Sequence

CALL M:TELER (buffer, nbytes)

buffer — An INTEGER variable or array into which the message is transferred. The number of bytes read is placed in the first byte of the buffer, followed by the message.

nbytes — An INTEGER variable that specifies the number of bytes to be read (including byte count in first byte).

Programming Considerations

This subroutine does not execute a carriage return or line feed. CALL CARRIAGE must be used to carriage return/line feed.

The first byte of the user's input buffer contains the number of bytes transferred. You must provide for this extra byte.

4.2.27 M:TELEW

The M:TELEW subroutine allows writes to the operator's console.

Calling Sequence

CALL M:TELEW (buffer, nbytes)

buffer — An INTEGER variable or array from which the message is typed.

nbytes — An INTEGER variable that specifies the number of bytes to be written.

Programming Consideration

This subroutine does not execute a carriage return or line feed. CALL CARRIAGE must be used in order to carriage return/line feed.

4.2.28 M:TESTAT

The M:TESTAT function returns the 32-bit user status wo d of any specified task in execution. The user status word resides in the CPU dispatch queue and is modified by the M:SET subroutine. Bit 0 is never set.

Calling Sequence

```
i = M:TESTAT (itask, *label)
```

i — An INTEGER variable that is set to the status word.

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled.

label — The statement label within the user task to which control is returned if the specified task is not currently in execution.

Programming Consideration

Test your own program user word by using zero (not ASCII) for a task name.

4.2.29 M:TESTIM

The M:TESTIM function returns the negative number of time units remaining until the specified timer entry times out. If the time has expired, the result returned is zero. The timer unit is set at system generation (SYSGEN).

Calling Sequence

```
i = M:TESTIM (id)
```

i — An INTEGER variable. Possible results in i are:

Negative number of time units remaining until timer times out
Zero if timer has expired
Zero if timer does not exist

id — An INTEGER variable that specifies the timer entry identification. This argument must be two ASCII characters, left justified and blank filled.

4.2.30 M:TIME5

The M:TIME5 subroutine creates entries in the timer table to request an interrupt. The timer unit is set at system generation (SYSGEN).

Calling Sequence

CALL M:TIME5 (id, next, period, interrupt, *label)

id	An INTEGER variable that specifies the timer entry identification. This argument must be two ASCII characters, left justified and blank filled.
next	An INTEGER variable that specifies the time to first activation (positive) in timer units.
period	An INTEGER variable that specifies the repetition period (positive) in timer units; if zero, a one-shot timer entry is created.
interrupt	An INTEGER variable that specifies the interrupt level to be requested.
label	The statement label to which control is returned if it is not possible to create a timer entry.

Programming consideration

Only privileged tasks may create the above timer entries.

4.2.31 M:TIME12

The M:TIME12 subroutine creates entries in the timer table to activate or resume a task. If the specified task does not have an associated entry in the dispatch queue when M:TIME12 is called, the task is placed in a preactivation stage and linked to the CPU dispatch queue in a suspend (SUSP) state. When the timer times out, the task is activated. If the task attempts to exit upon normal completion, it returns to the SUSP state.

Deletion of a task's associated timer entry has the following effects, depending on the task's current state:

Task active	The task terminates upon normal completion.
Task in SUSP state	The task remains in the SUSP state until the user resumes or deletes it.

Note that if a task is in a dynamic suspend state and the timer entry is deleted, the task will terminate upon normal completion.

Calling Sequence

CALL M:TIME12 (function, id, next, period, taskname, *label)

function An INTEGER variable that specifies either of two function codes:

1	Activate task
2	Resume task

Timer entries for 1 and 2 may be created by any program.

id An INTEGER variable that specifies the timer entry identification. This argument must be two ASCII characters, left justified and blank filled.

next An INTEGER variable that specifies the time to first activation (positive) in timer units.

period An INTEGER variable that specifies the repetition period (positive) in timer units.

taskname An INTEGER*8 variable that specifies the task. This argument must be one to eight ASCII characters, left justified and blank filled.

label The statement label to which control is returned if it is not possible to create a timer entry.

4.2.32 M:TIME34

The M:TIME34 subroutine creates entries in the timer table to set/reset a parameter via a mask. The timer unit is set at system generation (SYSGEN).

Calling Sequence

```
CALL M:TIME34 (function, id, next, period, switch, mask, *label)
```

function An INTEGER variable that specifies either of two function codes:

3	Set bits
4	Reset bits

Timer entries to set or reset bits may be created by any task, provided the bit is within the current user's map and is in a static memory partition. Only privileged users may set bits in a protected area of memory, i.e., the communication region of the operating system.

id An INTEGER variable that specifies the timer entry identification. This argument must be two ASCII characters, left justified and blank filled.

next An INTEGER variable that specifies the time to first activation (positive) in timer units.

period An INTEGER variable that specifies the repetition period (positive) in timer units; if zero, a one-shot timer entry is created.

switch An INTEGER variable that specifies the word in which bits may be set or reset.

mask An INTEGER that specifies the bit configuration of the mask word, ORed for set, ANDed for reset function.

label The statement label to which control is returned if it is not possible to create a timer entry.

4.2.33 M:WAIT

The M:WAIT subroutine suspends the execution of the task until I/O (e.g., BUFFERIN/BUFFEROUT) is complete.

Calling Sequence

```
CALL M:WAIT (lfc)
```

lfc An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

4.3 X: Subroutines

The subroutines described in this section begin with an X: prefix, which indicates additional subroutines available to the MPX-32 user only.

Many of the X: subroutines contain the optional parameters istatus and *label. If both of these parameters are omitted and an error occurs, the current task aborts with an RSxx code, where xx is one of the istatus codes listed in the parameter description for that subroutine.

4.3.1 X:ANYW

The X:ANYW subroutine places the calling task in a state waiting for the completion of any no-wait request, the receipt of a message, or a break interrupt. The task is removed from the associated ready-to-run list and placed in the any-wait list. A return is not made until one of the wait conditions has been satisfied or until the specified time-out value has expired.

Calling Sequence

```
CALL X:ANYW (itime,[istatus],[*label])
```

itime — An INTEGER variable interpreted as follows:

- 0 — If wait for an indefinite period is requested.
- -n — Contains the negative number of time units to elapse before the wait is terminated. The actual time elapsed can vary by one time unit because of system design.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
47	Invalid time interval request.
50	Missing parameter.
51	Invalid parameter.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.2 X:ASYNCH

The X:ASYNCH subroutine, in conjunction with X:SYNCH, enables the task to reenter the normal asynchronous task interrupt mode. Any software interrupts occurring in the asynchronous mode are processed immediately. When interrupt processing is complete, the task resumes execution at the point at which it was interrupted. If the task is already in the asynchronous mode when the subroutine is called, it remains in the asynchronous mode.

Calling Sequence

```
CALL X:ASYNCH
```

Programming Consideration

Main program and interrupt routines in the asynchronous interrupt mode must not call common routines that are not reentrant.

4.3.3 X:BORT

The X:BORT subroutine aborts a task. If the specified task is swapped out, it is not aborted until it regains CPU control. If the specified task is not in execution, the denial return is taken. This subroutine can also abort the calling task. In both cases, the abort may be with an extended message. To force I/O completion and immediately abort a specified task, use the X:DELTSK subroutine. The scope of X:BORT is dependent upon the access restrictions of the owner of the calling task. Three types of task termination are provided by the MPX-32 executive: exit, abort, and delete task execution.

Calling Sequence

```
CALL X:BORT (itask, iabcode,[extcode],[istatus],[*label])
```

itask — An argument that specifies the task name or task number. The task name must be an INTEGER*8 or CHARACTER variable of one to eight characters, left justified and blank filled. The task number must be an INTEGER*8 variable with bits 0 through 31 containing binary zeros and bits 32 through 64 containing an eight-digit hexadecimal task number. A task number of zero specifies the calling task.

iabcode — An INTEGER variable that specifies a one to four ASCII character abort code.

extcode — An INTEGER*8 or CHARACTER variable that specifies the extended abort code message. This argument must be one to eight ASCII characters, left justified and blank filled.

istatus — An INTEGER variable. If specified, istatus is set to a condition code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Request denied, task does not have a Dispatch Queue Entry (DQE).
50	Missing parameter.

label The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task will abort with an RSxx, where xx is one of the listed istatus values; or RTxx, where xx is an istatus code of 50.

4.3.4 X:BRK

The X:BRK subroutine establishes a routine that is entered whenever any task activates the break interrupt using the X:INT subroutine.

Calling Sequence

CALL X:BRK (*label$_1$,[istatus],[*label$_2$])

label$_1$ The statement label of the beginning of the task's break/task interrupt routine.

istatus An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
32	Missing parameter.
33	Parameter out of range (label$_1$ is outside of user's address space).

label$_2$ The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label$_2$ are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.5 X:BRKXIT

The X:BRKXIT subroutine returns control to the point of interruption at the conclusion of a software interrupt routine (i.e., break receiver).

Calling Sequence

CALL X:BRKXIT

4.3.6 X:DELTSK

The X:DELTSK subroutine forces I/O completion and immediately aborts the specified task. The user must specify an abort code and may optionally specify an extended code. To abort a task when it gains CPU control (i.e., not immediately), use the M:ABORT subroutine or the X:BORT subroutine. Three types of task termination are provided by the MPX-32 executive: exit, abort, and delete task execution.

Calling Sequence

CALL X:DELTSK (itask, iabcode,[iextcode],[istatus],[*label])

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled. A task number of zero specifies the calling task.

iabcode — An INTEGER variable that specifies the one to four ASCII character abort code.

iextcode — An INTEGER*8 or CHARACTER variable that specifies the extended abort code message. This argument must be one to eight ASCII characters, left justified and blank filled.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Request denied, task does not have a Dispatch Queue Entry (DQE).
50	Missing parameter.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values; or RTxx, where xx is an istatus code of 50.

4.3.7 X:DISCON

The X:DISCON subroutine disconnects a task previously connected to an interrupt level.

Calling Sequence

CALL X:DISCON (itask,[istatus],[*label])

itask — An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled. A task number of zero specifies the calling task.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
68	Task name/task number not found.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.8 X:DSMI

The X:DSMI subroutine disables software interrupts for messages sent to the calling task. This subroutine is useful for synchronous control of task message interrupt gating.

Calling Sequence

CALL X:DSMI

4.3.9 X:EAWAIT

The X:EAWAIT subroutine allows a task that is processing software interrupts synchronously (i.e., has called X:SYNCH) to proceed as follows:

1. If there are any no-wait operations (e.g., BUFFERIN/BUFFEROUT) that have not yet completed, the task is placed in a wait state where it remains until either the completion of any no-wait operations or the specified time-out value expires.

2. All software interrupts that occurred either before X:EAWAIT was called or during X:EAWAIT, are processed on a priority basis until all interrupts and no-wait operations have been processed.

3. If there are no no-wait operations outstanding, control will return immediately to the task without waiting for the specified time-out value.

If X:EAWAIT is used in the asynchronous processing mode, it has the same effect as X:ANYW. Break interrupts are still handled asynchronously (refer to X:SYNCH).

Calling Sequence

```
CALL X:EAWAIT (itime,[istatus],[*label])
```

itime — An INTEGER variable interpreted as follows:

0	Wait for an indefinite period is requested.
-n	Contains the negative number of timer units to elapse before the wait is terminated. The actual time elapsed can vary by one time unit because of system design.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
47	Invalid time interval request.
50	Missing parameter.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.10 X:ENMI

The X:ENMI subroutine enables task interrupts for messages sent to the calling task. This subroutine removes an inhibit condition previously established by X:DMSI.

Calling Sequence

```
CALL X:ENMI
```

4.3.11 X:EOPT

The X:EOPT subroutine writes the volume record (a header record for each volume of a multi-volume tape) if the tape is positioned at the Beginning-Of-Tape (BOT) on a multivolume magnetic tape. If the tape is positioned at the End-Of-Tape (EOT) on a multivolume magnetic tape, the subroutine performs an erase/write End-Of-File (EOF).

X:EOPT is not applicable to either blocked or system files, e.g., System Control (SYC), System General Object (SGO), System Listed Output (SLO), System Binary Output (SBO).

Calling Sequence

```
CALL X:EOPT (lfc,[istatus],[*label])
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.12 X:FDSPCE

The X:FDSPCE subroutine deallocates a task's most recently acquired extended memory map block(s), thus contracting its address space. Refer to the MPX-32 Reference Manual, Volume 1, for a discussion of memory allocation and logical address space.

Calling Sequence

CALL X:FDSPCE (nask, nget,[istatus],[*label])

nask — An INTEGER variable that specifies the number of map blocks of extended data space to deallocate.

nget — An INTEGER variable that, upon return, contains the number of map blocks actually deallocated.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameters.
51	Parameter out of range (nask must be positive).

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.13 X:FESPCE

The X:FESPCE subroutine dynamically deallocates a task's most recently acquired execution space map block, thus contracting the task's address space.

Calling Sequence

CALL X:FESPCE (nask, nget)

nask — An INTEGER variable that specifies the number of map blocks of execution space to deallocate.

nget — An INTEGER variable that, upon return, contains the number of map blocks actually deallocated.

4.3.14 X:FSLR

The X:FSLR subroutine, in conjunction with X:FSLS, controls a synchronization lock indicator for disc file gating. A call to X:FSLR releases the synchronization lock and polls the queue of tasks waiting to become lock owners.

Calling Sequence

```
CALL X:FSLR (lfc, [istatus],[*label])
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

Native	Compatible	
0	0	Normal Completion.
32	1	Request denied, file lock not set.
29	5	Request denied, LFC not allocated.
30	6	Request denied, specified LFC not assigned to a permanent disc file.
69	69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Considerations

A file lock can be cleared only by the task that set it.

A file lock owned by a task is automatically released when the task terminates.

A file lock is automatically released when the file is deallocated.

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.15 X:FSLS

The X:FSLS subroutine, in conjunction with X:FSLR, provides disc file gating. The X:FSLS and X:FSLR subroutines control a synchronization lock indicator, which allows synchronized access to a disc file concurrently allocated to multiple tasks. A call to X:FSLS sets a synchronous lock. The file identified by the logical file code in the syntax must have been previously allocated to the calling task.

Calling Sequence

```
CALL X:FSLS (lfc,[time],[istatus],[*label])
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

time — An INTEGER variable interpreted as follows:

- +1 — Return immediately with a denial code if the file already has a file lock set.
- 0 — Place the calling task in a wait state until the task owns the synchronous lock. This is the default value.
- -n — Place the calling task in a wait state until either the task owns the file lock or the expiration of n timer units, whichever occurs first.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

Native	Compatible	
0	0	Normal completion.
50	1	Request Denied, file lock already owned.
38	2	Request denied, time out occurred while waiting to become lock owner.
29	5	Request denied, LFC not allocated.
30	6	Request denied, LFC not assigned to a permanent disc file.
69	69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.16 X:FWD

The X:FWD subroutine advances files or records on sequential blocked files. This subroutine performs the following functions for advance records:

- Verifies volume record if Beginning-Of-Tape (BOT) is on multivolume magnetic tape.
- Advances the specified number of records.

X:FWRD performs the following functions for advance file:

- Advances logical records until an End-Of-File (EOF) is found, if the file is blocked.
- Verifies volume record if BOT is on multivolume magnetic tape.
- Advances the specified number of files.

This subroutine is not applicable for unblocked files in the file advance mode or system files in the record advance mode.

Calling Sequence

CALL X:FWRD (lfc,[record], adv,[istatus],[*label])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

record — An INTEGER variable that specifies the type of advance to be performed. This argument value must be a single ASCII character, left justified and blank filled. Options are:

'R'	Record advance.
Default	File advance.

adv — An INTEGER variable that contains the number of records or files to advance. If zero is specified, the service is nullified.

istatus — An INTEGER variable. If specified, istatus is set to a condition code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.17 X:FXLR

The X:FXLR subroutine, in conjunction with X:FXLS, provides disc file gating. A call to X:FXLR releases the exclusive lock and allows other tasks to allocate the associated disc file. Another task is not able to exclusively lock the file until the file is deallocated by this task.

Calling Sequence

CALL X:FXLR (lfc,[istatus],[*label])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

Native	Compatible	
0	0	Normal completion.
32	1	Request denied, file lock not set.
29	5	Request denied, specified LFC not allocated.
30	6	Request denied, specified LFC not assigned to a permanent disc file.
33	N/A	Request denied, resource is not allocated in a sharable mode by this task.
69	69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Considerations

A file lock can be cleared only by the task that set it.

A file lock owned by a task is automatically released when the task terminates.

A file lock is automatically released when the file is deallocated.

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.18 X:FXLS

The X:FXLS subroutine, in conjunction with X:FXLR, provides disc file gating. This subroutine allows the calling task to gain exclusive allocation of a file as though it were an unshared resource. The file must have been previously allocated.

Calling Sequence

```
CALL X:FXLS (lfc,[time],[istatus],[*label])
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

time — An INTEGER variable interpreted as follows:

- +1 — Return immediately with a denial code if the file already has a file lock set.
- 0 — Place the requesting task in a wait state until the task owns the exclusive lock. This is the default value.
- -n — Place the requesting task in a wait state until either the task owns the file lock or the expiration of n timer units, whichever occurs first.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

Native	Compatible	
0	0	Normal completion.
90	1	Request denied, file lock allocated to another task, or is already exclusively locked.
38	4	Request denied, time-out occurred while waiting to become lock owner.
29	5	Request denied, LFC is not allocated.
30	6	Request denied, LFC is not assigned to a permanent disc file.
69	69	File Control Block (FCB) not located.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.19 X:GADRL

The X:GADRL subroutine returns the memory addresses associated with the boundaries of a user's task.

Calling Sequence

CALL X:GADRL (lextn, fdsect, lload, ldsect, fcsect)

lextn — An INTEGER variable that contains the word address of the last location currently available in the task's extended data space (always a map block boundary - one word).

fdsect — An INTEGER variable that contains the word address of the first location of the task's DSECT (always a map block).

lload — An INTEGER variable that contains the word address of the last location in the DSECT actually loaded by the loader.

ldsect — An INTEGER variable that contains the logical word address of the last location available in the task's DSECT (always a map block boundary - one word).

fcsect — An INTEGER variable that contains the word address of the first location of the task's CSECT or COMMON allocation (always a map block boundary).

4.3.20 X:GDSPCE

The X:GDSPCE subroutine dynamically acquires additional map blocks of memory in a task's extended area. Up to 15 blocks may be requested in an 8KW (32/7X) environment if sufficient memory exists. On a 2KW (32/27 or 32/87) environment up to 190 blocks may be requested and on a 2KW (32/67 or 32/97) environment up to 1900 blocks may be requested if sufficient memory exists. The memory will be the same type specified when the task was cataloged, and it is mapped in a logically contiguous manner, with the first requested map block starting at 128K words. The task is suspended until the allocation is successful. X:FDSPCE deallocates the acquired space in reverse order.

Calling Sequence

CALL X:GDSPCE (±nask, nget,[istatus],[*label])

nask — An INTEGER variable that specifies the number of map blocks of extended index data space to append with zeroing (-nask) or without zeroing (+nask) before return.

nget — An INTEGER variable that, upon return, contains the number of map blocks actually received.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Attempted allocation of an excessive number of map blocks.
2	Attempted allocation exceeds physical memory configured.
50	Missing parameter.

label — The statement label to which control is returned if an error exists.

Programming Considerations

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes; or RTxx, where xx is an istatus code of 50.

If the map block starting at 128K words is in use, the next available block is used. If an extended block above 128K words is in use, the map block starting at 128K words is used. (Blocks will be acquired up to and then beyond the extended block.)

4.3.21 X:GESPCE

The X:GESPCE subroutine dynamically expands a task's memory allocation in map block increments starting at the end of its DSECT, up to the top of its logical address space. The additional memory is the same type specified when the task was cataloged. The task is mapped in a logically contiguous manner up to the start of its CSECT, global common, or 128K words, whichever occurs first. The task is suspended until the allocation is successful.

Calling Sequence

```
CALL X:GESPCE (±nask, nget,[istatus],[*label])
```

nask — An INTEGER variable that specifies the number of map blocks of extended execution space to obtain with zeroing (-nask) or without zeroing (+nask) before return.

nget — An INTEGER variable that contains upon return, the number of map blocks actually received.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Excessive DSECT allocation attempted.
2	Attempted allocation exceeds physical memory configured.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus codes.

4.3.22 X:GMSGP

The X:GMSGP subroutine, when called from the message receiver routine of a task that has received a message interrupt, transfers message parameters into the designated receiver buffer and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB).

Calling Sequence

CALL X:GMSGP (prbname,[istatus],[*label])

prbname — An INTEGER array that specifies the PRB. Refer to the MPX-32 Reference Manual, Volume 1, for a description of the contents of the PRB. Note that the parameter receiver buffer address within the PRB must be aligned on a word boundary.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Invalid PRB address.
2	Invalid receiver buffer address.
3	No active send request.
4	Receiver buffer length exceeded.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.23 X:GRUNP

The X:GRUNP subroutine, when called by a task that is executing on behalf of a run request, transfers the run parameters into the designated receiver buffer and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB).

Calling Sequence

CALL X:GRUNP (prbname,[istatus],[*label])

prbname — An INTEGER array that specifies the PRB. Note that the parameter receiver buffer address within the PRB must be aligned on a word boundary.

istatus An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal status.
1	Invalid PRB address.
2	Invalid receiver buffer address.
3	No active send request.
4	Receiver buffer length exceeded.

label The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.24 X:ID

The X:ID subroutine obtains information about an active task, including the calling task, when one of the following parameters is known:

- Task number
- Task load module name
- Owner name
- Pseudonym name of the task

The caller must supply at least one of these parameters; remaining parameters are returned by the subroutine. Repeated calls may be made to the subroutine to find all copies of a multicopied task.

Calling Sequence

```
CALL X:ID (index, taskno, taskname, ownername, pseudo,[istatus],[*label])
```

index An INTEGER variable that must be zero for the initial search. This argument is used by the system subroutine to control the points at which previous searches are discontinued and a search of the dispatch queue is begun. Index must not be a constant because the X:ID subroutine updates the contents of this parameter for retrieval of further entries when repeated calls are made. When all matching tasks are identified, index is returned with value zero.

taskno An INTEGER variable that specifies the task number. It must be set to zero if the task number is unknown.

taskname An INTEGER*8 variable or CHARACTER variable that specifies the task name. This argument must be one to eight ASCII characters, left justified and blank filled. If the taskname is unknown, taskname must contain either a zero or all blanks.

ownername An INTEGER*8 variable or CHARACTER variable that specifies the task's owner name. This argument must be one to eight ASCII characters, left justified and blank filled. If the task owner name is unknown, ownername must contain either a zero or all blanks.

pseudo — An INTEGER*8 variable or CHARACTER variable that specifies pseudonym. This argument must be one to eight ASCII characters, left justified and blank filled. If the pseudonym is unknown, pseudo must contain either a zero or all blanks.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
51	Task not found.
52	All tasks found.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.25 X:INT

The X:INT subroutine issues a break interrupt to any specified task, including the calling task. The specified task is required to have a break receiver routine (refer to X:BRK). This subroutine has the same effect as depressing the break key on the user terminal.

Calling Sequence

CALL X:INT (itask,[istatus],[*label])

itask — An INTEGER*8 that specifies the task name or task number. This argument must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled. A task number of zero specifies the calling task.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
48	Invalid task number.
50	Missing parameter.
53	Invalid receiver.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values; or RTxx, where xx is an istatus code of 50 or 53.

4.3.26 X:RCVR

The X:RCVR subroutine establishes the address of a routine that is to be entered to receive messages sent by other tasks.

Calling Sequence

CALL X:RCVR (*$label_1$,[istatus],[*$label_2$])

$label_1$ — The statement label of the beginning of the task's message receiver routine.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
51	Parameter out of range (invalid receiver address).

$label_2$ — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *$label_2$ are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values; or RTxx, where xx is an istatus code of 50 or 51.

4.3.25A X:MPXEOF

The X:MPXEOF subroutine establishes an optional method of end-of-file checking for formatted, unblocked disc and magnetic tape files. This optional method overrides the default FORTRAN 77+/SRTL method by checking for X'0FE0FE0F' in the first word of a formatted, unblocked record, instead of checking for X'0F' in the first byte of the record. All ENDFILE operations on formatted, unblocked disc and magnetic tape files will write X'0FE0FE0F' in the first word of the current record.

This optional method provides a better method of discrimination between data patterns and EOF indicators, but does not completely eliminate ambiguities. If closer discrimination is necessary, blocked files should be used.

Calling Sequence

```
CALL X:MPXEOF
```

Programming Consideration

For proper use, this subroutine must be called only once, at the beginning of a main program before any I/O is attempted.

4.3.27 X:RSML

The X:RSML subroutine locks a specified resourcemark and, in conjunction with X:RSMU, synchronizes access to a common resource.

Calling Sequence

CALL X:RSML (lockid,[time],[swap],[istatus],[*label])

lockid — An INTEGER variable that specifies the unique numeric resourcemark index (1 through 64). Privileged tasks may use 1 through 64, unprivileged tasks 33 through 64.

time — An INTEGER variable that specifies action to be taken if the lock is currently set and owned by another task. Possible values are:

+1	Immediate denial return.
0	Wait until task is lock owner (default).
-n	Wait until task is lock owner or until n timer units have expired, whichever occurs first.

swap — An INTEGER variable that specifies the swapping mode for the calling task. This argument must be a single ASCII character, left justified and blank filled. Options are:

'P'	While this task is waiting to become lock owner, the swapping mode is to be set to swap this task only if a higher priority task is requesting memory space.
Default	The task is a swap candidate if any task is requesting memory.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Lock index exceeds maximum range.
2	Lock index less than minimum range.
3	Lock owned by another task, and time = +1.
4	Lock owned by another task, time = -n; n timer units have elapsed.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.28 X:RSMU

The X:RSMU subroutine unlocks a resourcemark previously locked by a call to the X:RSML subroutine. If any other tasks are waiting to lock the specified resourcemark, the highest priority waiting task becomes the new lock owner.

Calling Sequence

```
CALL X:RSMU (lockid,[istatus],[*label])
```

lockid — An INTEGER variable that specifies the unique numeric resourcemark index (1 through 64).

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Lock index exceeds maximum range.
2	Lock index less than minimum range.
3	Lock owned by another task.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.29 X:SMSGR

The X:SMSGR subroutine sends up to 768 bytes to a specified destination task. This subroutine may also accept up to 768 bytes as return parameters.

Calling Sequence

```
CALL X:SMSGR (ipsb,[istatus],[*label])
```

ipsb — An INTEGER array that specifies the Parameter Send Block (PSB). Note that the send buffer and the return parameter buffer within the Parameter Receive Block (PRB) must be aligned on a word boundary.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Task not found in Dispatch Queue.
10	Invalid priority.
11	Invalid send buffer address.
12	Invalid return buffer address.
13	Invalid no-wait mode end-action routine address.
14	Memory pool unavailable.
15	Destination task queue depth exceeded.
16	Invalid PSB address.

label — The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.30 X:SRUNR

The X:SRUNR subroutine activates or reactivates the specified destination task with a parameter pass of up to 768 bytes. This subroutine also accepts up to 768 bytes as return parameters.

Calling Sequence

```
CALL X:SRUNR (ipsb,[istatus],[*label],[itaskno] )
```

ipsb — An INTEGER array that specifies the Parameter Send Block (PSB). Note that the send buffer and the return parameter buffer within the Parameter Receive Block (PRB) must be aligned on a word boundary.

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
2	Load module name not found in System Master Directory (SMD).
3	Load module file is password protected.
4	Invalid load module file format.
5	Dispatch Queue Entry (DQE) unavailable.
6	I/O error on SMD read.
7	I/O error on load module read.
10	Invalid priority.
11	Invalid send buffer address or send quantity exceeds 768 bytes.
12	Invalid return buffer address.
13	Invalid no-wait mode end-action routine address.
14	Memory pool unavailable.
15	Destination task queue depth exceeded.
16	Invalid PSB address.

label — The statement label to which control is returned if an error exists.

itaskno — An INTEGER variable that will receive, upon return, the task number of the task to which the run request was sent.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.31 X:SUAR

The X:SUAR subroutine establishes an address to which control is transferred when an abort occurs during task execution.

Calling Sequence

CALL X:SUAR (*label$_1$, istatus , *label$_2$)

label$_1$ A statement label that receives control when the executing task is aborted for any reason.

istatus An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
1	Specified address outside of the user's allocated area.

label$_2$ A statement label to which control is returned when a denial error exists.

Programming Consideration

All files remain open if transfer occurs.

If istatus and *label$_2$ are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.32 X:SUSP

The X:SUSP subroutine suspends any task, including the calling task, for either a specified number of time units or an indefinite time, as requested. A task resumes after the specified time interval completes. The M:RSUM subroutine resumes a task suspended for an indefinite time interval. Receipt of a message interrupt also resumes a suspended task.

Calling Sequence

CALL X:SUSP (task, time, *label)

task An INTEGER*8 variable that specifies the task name or task number. A task name must be one to eight ASCII characters, left justified and blank filled. A task number must be right justified in the doubleword and zero filled.

time An INTEGER variable interpreted as follows:

0	Suspension for an indefinite period of time.
+n	Positive number of units to elapse before the calling task is resumed. The actual time elapsed can vary by one time unit because of system design.

label The statement label to which control is returned if an error exists.

4.3.33 X:SYNCH

The X:SYNCH subroutine processes task interrupts by placing the task in a synchronous task interrupt mode. X:EAWAIT must be used in conjunction with X:SYNCH. X:SYNCH processes interrupts in the following manner:

- Interrupts that occur after a call to X:SYNCH are queued on a priority basis until X:EAWAIT is called.
- X:EAWAIT either processes interrupts or completes outstanding no-wait operations.
- The wait subroutine is exited immediately if there are no interrupts or outstanding no-wait operations.

Other wait subroutines (X:ANYW, X:ASYNCH, or M:WAIT) may be used with X:SYNCH; however, they only provide a subset of the X:EAWAIT service. If M:WAIT is called subsequent to X:SYNCH, no breaks can occur.

Calling Sequence

```
CALL X:SYNCH
```

Programming Consideration

Break interrupts are still handled asynchronously in synchronous mode. Therefore, use caution to prevent the occurrence of any nonreentrant common routines between the main program and the break receiver (e.g., run-time library routines).

4.3.34 X:TDAY

The X:TDAY subroutine returns the time of day to the caller as computed from the real-time clock interrupt counter. A SYSGEN parameter initializes the counter.

Calling Sequence

```
CALL X:TDAY (itime, idate)
```

itime — An INTEGER variable that receives the time in the following format:

Byte 0	Hours	(0-23)
Byte 1	Minutes	(0-59)
Byte 2	Seconds	(0-59)
Byte 3	Interrupts	

idate — An INTEGER*8 variable that contains, upon return, the date in the following ASCII character format with embedded slant characters /:

```
MM/DD/YY
```

If the date is entered in European format, the following ASCII character format is returned:

```
DDMMMYY
```

4.3.35 X:TSCAN

The X:TSCAN subroutine allows a task activated through the Terminal Subroutines Manager (TSM) to scan the parameters (fields) you pass in the line buffer at execution time. This subroutine returns a number of parameters from the terminal line buffer according to the value specified by the nask argument. If the line buffer contains more parameters than the value specified by nask, only the first nask parameters will be returned, i.e., nget will return a value equal to nask. If the line buffer contains the same or fewer parameters than specified by nask, all of the parameters will be returned, i.e., nget will return a value equal to the number of parameters contained in the line buffer.

Parameters must be separated by delimiting characters. Valid delimiter characters for parameters are blanks, commas, semicolons, equal signs, carriage return, line feeds, dollar signs, exclamation points, percent signs and left or right parentheses. Note that all blanks preceding a parameter or another delimiter are ignored.

Calling Sequence

```
CALL X:TSCAN (nask, nget, ix, text)
```

nask — An INTEGER expression that specifies the number of parameters to be transferred.

nget — An INTEGER variable that contains, upon return, the number of parameters actually transferred.

ix — An INTEGER*1 array that contains, upon return, the indices to delimiter character positions in the text parameter. The dimensions of ix should be greater than or equal to nask +1; ix(1) will always be zero.

text — A CHARACTER variable of sufficient length to contain requested parameters. Parameter text and delimiters are copied into text with extraneous blanks deleted and individual parameter strings truncated in order not to exceed eight bytes.

4.3.36 X:XMEA

The X:XMEA subroutine exits an end action routine associated with a no-wait message send request.

Calling Sequence

```
CALL X:XMEA
```

4.3.37 X:XMSGR

The X:XMSGR subroutine exits the message receiver routine of the calling task. This subroutine must be called after the task has received a message from another task.

Calling Sequence

```
CALL X:XMSGR (rxbname,[istatus],[*label])
```

rxbname An INTEGER array that specifies the Receiver Exit Block (RXB).

istatus An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
51	Parameter out of range.

label The statement label to which control is returned if an error exists.

Programming Consideration

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

4.3.38 X:XNWIO

The X:XNWIO subroutine must be called to exit a no-wait I/O end action routine (both normal and error end action routines). The subroutine returns control to the point of interruption.

Calling Sequence

```
CALL X:XNWIO
```

4.3.39 X:XREA

The X:XREA subroutine exits an end action routine associated with a no-wait run request.

Calling Sequence

```
CALL X:XREA
```

4.3.40 X:XRUNR

The X:XRUNR subroutine exits a task that is executing on behalf of a run request issued from another task.

Calling Sequence

```
CALL X:XRUNR (rxbname,[istatus],[*label])
```

rxbname — An INTEGER array that specifies the Receiver Exit Block (RXB).

istatus — An INTEGER variable. If specified, istatus is set to a completion code upon return from the subroutine. Possible istatus values are:

0	Normal completion.
50	Missing parameter.
51	Parameter out of range.

label — The statement label to which control is returned if an error exists.

Programming Considerations

If istatus and *label are omitted and an error occurs, the current task aborts with an RSxx, where xx is one of the listed istatus values.

The run receiver queue is examined and, if it is not empty, the task is executed again on behalf of the next request. If the queue is empty, the exit options in the RXB are examined. If the option byte is zero, the task will be placed in a wait state, waiting for the next run request to be received. If the option byte is nonzero, the task exits the system. Reexecution of the task transfers control to the instruction following the X:XRUNR call.

ADDR
CARRIAGE
DUMPUSER

4.4 Named Subprograms

The following named subprograms are available in either compatible or native mode.

4.4.1 ADDR

The ADDR function returns a 24 bit pure address for the specified argument. This subroutine is generated inline by FORTRAN 77+ and does not exist in the SRTL.

Calling Sequence

```
i=ADDR (var)
```

i — An INTEGER variable that, upon return, contains the 24 bit pure address of the argument.

var — A variable name, array name, procedure name or statement label for which the address is desired. If var is a statement label, it must be preceded by a dollar sign ($).

4.4.2 CARRIAGE

The CARRIAGE subroutine forces a carriage return/line-feed on the operator's console.

Calling Sequence

```
CALL CARRIAGE
```

4.4.3 DUMPUSER

The DUMPUSER subroutine writes a hexadecimal dump of a task's allocated memory, formatted as hexadecimal side-by-side with ASCII. The program status doubleword and machine registers are output first, followed by the task subroutine area (TSA), then the user program and data. The dump is written to a dynamically allocated SLO file.

Calling Sequence

```
CALL DUMPUSER
```

4.4.4 EOF

The EOF subroutine determines if the previous I/O operation read an end-of-file mark. EOF resets the end-of-file status.

Calling Sequence

```
CALL EOF (lfc, status)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

status — A LOGICAL*1 variable that is set as follows:

.TRUE.	end-of-file was found.
.FALSE.	end-of-file was not found.

Programming Consideration

STATUS and EOF both reset the EOF bit after testing. Therefore, use caution when calling these routines in the same program, because the valid results of one routine may be jeopardized by the other's ability to reset the EOF bit.

4.4.5 EXIT

The EXIT subroutine returns control to MPX-32 and terminates execution of the program, after closing all currently open files

Calling Sequence

```
CALL EXIT
```

4.4.6 LOCF

The LOCF function returns the address of a specified argument.

Calling Sequence

```
iaddress = LOCF (var)
```

iaddress — An INTEGER variable that, upon return, contains the address of the argument.

var — A variable name, array name, subroutine name, function name, or statement label for which the address is desired. If var is a statement label, it must be preceded by a dollar sign ($). For example, IADDRESS = LOCF ($55).

Programming Consideration

Addresses returned by this function include the F and C bits. Bit 12 set indicates bits 13 through 31 contain a byte address. Bit 12 clear indicates bits 30 and 31 are interpreted as listed below. This does not apply to byte variables in extended memory however, because bit 12 is used as an address bit.

30	31	
0	0	- word address
0	1	- left half word address
1	0	- double word address
1	1	- right half word address

4.4.7 MPXSVC

The subroutine MPXSVC allows the user to execute an MPX-32 SVC and has the following interface:

Calling Sequence

```
MPXSVC(isvc,iregin,iregout,icc,istatus)
```

isvc — An INTEGER fullword that contains the SVC number to be executed.

iregin — An INTEGER fullword array whose first eight elements contain the values to be loaded into the machine registers prior to the execution of the SVC.

iregout — An INTEGER array whose first eight elements will receive the values from the machine registers after the SVC has been executed.

icc — An INTEGER variable that receives the condition code bits, right justified and zero filled, after the SVC has been executed.

istatus — An INTEGER variable that receives the status value of this subroutine. Zero indicates success.

Programming Consideration

A system service listed as SVC 1,X'28' would be given in the form X'1028'.

4.4.8 SSWTCH

The SSWTCH subroutine obtains the status of a specified hardware control switch.

Calling Sequence

CALL SSWTCH (iswitch, jstate)

iswitch — An INTEGER variable that specifies the hardware control switch (0 to 12) to be tested.

jstate — An INTEGER or LOGICAL variable that is set as follows:

If the sense switch is on, jstate is set to one if an INTEGER variable or to .TRUE. if a LOGICAL variable.

If the sense switch is off, jstate is set to two if an INTEGER variable or to .FALSE. if a LOGICAL variable.

4.4.9 STATUS

The STATUS subroutine tests the status of any given unit resulting from the latest I/O operation of the unit. This subroutine resets the end-of-file status if found.

Calling Sequence

CALL STATUS (lfc, istatus [,n])

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

istatus — An INTEGER variable set according to the results of the status test, as follows:

1 Not ready.
2 Ready and no previous error.
3 EOF sensed on latest input operation.
4 Parity or lost data error on latest I/O operation.
5 Unit is not open.

n — An INTEGER variable that is set to the data transfer count. When the operation is complete, the data transfer count is in bytes. n is undefined if status = 5.

Programming Consideration

STATUS and EOF both reset the EOF bit after testing. Therefore, use caution when calling these routines in the same program, because the valid results of one routine may be jeopardized by the other's ability to reset the EOF bit.

CHAPTER 5

MPX-32 NATIVE MODE SUBROUTINES

5.1 X_Subroutines

The subroutines described in this section begin with an X_ prefix, which indicates they are available to FORTRAN 77+ users in native mode. All parameters contained within the syntax of a subroutine call are required. Null parameters are not allowed for the subroutines described in this section.

Some X_ subroutines contain the pathname argument. The pathname is a one to three part name that identifies the path to be taken to a volume, directory, or file. Each component of a pathname (volume, directory, file) can be one to sixteen characters. Wherever file names are valid, a complete pathname can be specified, or missing portions of a pathname are assigned.

Dynamic partitions may be defined in user directories on some versions of the operating system. Although some memory management X_ subroutines contain pathname arguments, only dynamic partitions in the system volume and directory are supported by the SRTL.

Any attempt to mix modes; i.e., compatible with native, will result in an RS99 error with the entry point displayed in an extended abort message.

5.1.1 X_CPART

The X_CPART subroutine creates a resource descriptor for a dynamic memory partition. A dynamic memory partition is named and exists until it is deleted.

Calling Sequence

```
CALL X_CPART (pathname,ownername,projectname,owneraccess,projectaccess,
              otheraccess,shared,size,start,class,istatus)
```

pathname — A CHARACTER expression that specifies the path to the memory partition.

ownername — A CHARACTER expression that specifies the ownername to be associated with the partition being created. A value of all spaces indicates the current ownername of the task is used. When you create a resource definition, specifying an ownername that differs from yours does not change your ownername, it only specifies the ownername associated with the resource being created.

projectname — A CHARACTER expression that specifies the projectname having specific access privileges to the partition. A value of all spaces indicates the projectname associated with the task is used.

owneraccess — A CHARACTER expression that specifies the resource owner's access rights. The access mode (with the exceptions of 'N' and 'Ø') may be specified in any order as a concatenated string (e.g., 'RD' or 'W'//'R'). Duplicates are not allowed.

'R'	Allows partition contents to be read.
'W'	Allows partition write access.
'D'	Allows partition to be deleted.
'N'	Allows no partition access.
'Ø'	Blank allows all access types.

projectaccess — A CHARACTER expression that specifies the access rights of project groups having specific resource access privileges. The access mode (with the exceptions of 'N' and 'Ø') may be specified in any order as a concatenated string (e.g., 'RD' or 'W'//'R'). Duplicates are not allowed.

'R'	Allows partition contents to be read.
'W'	Allows partition entries.
'D'	Allows partition to be deleted.
'N'	Allows no partition access.
'Ø'	Blank allows all access types.

otheraccess — A CHARACTER expression that specifies the access rights of a partition user other than the owner or a specified project group. The access mode (with the exceptions of 'N' and 'ϕ') may be specified in any order as a concatenated string (e.g., 'RD' or 'W'//'R'). Duplicates are not allowed.

'R'	Allows partition contents to be read.
'W'	Allows partition entries.
'D'	Allows partition to be deleted.
'N'	Allows no partition access.
'ϕ'	Blank allows all access types.

shared — A LOGICAL expression that is either true or false. .TRUE. specifies the partition may be shared; .FALSE. specifies the partition can support only one user at a time (exclusive use).

size — An INTEGER expression that specifies the number of protection granules (512 words each) to include in the partition.

start — An INTEGER expression that specifies the starting protection granule in either the nonextended logical address space (1 through 255) or the extended logical address space (256 or greater) at which the memory partition is to be mapped.

class — A CHARACTER expression that specifies the memory class for the memory partition. The values allowed are 'E' (first 128K words), 'H' (high speed), and 'S' (slow). A 'ϕ' (blank) signifies the default (slow).

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

Programming Consideration

Granules in the first several map blocks should not be specified since they are used for the operating system.

5.1.2 X_CREDIR

The X_CREDIR subroutine creates a directory in a volume root directory. You may create entries on the specified volume by gaining access as either the owner of the volume, a member of a defined project group, or an arbitrary user of the volume. You must have add-entry access to volume root directory. In addition, the volume must be mounted.

Calling Sequence

```
CALL X_CREDIR (pathname,ownername,projectname,owneraccess,projectaccess,
               otheraccess,shared,maxsize,start,istatus)
```

pathname — A CHARACTER expression that specifies the path to the new directory. The one to sixteen ASCII character name of the new directory must be the last name in the path. Supplying only a directory name defines the directory in your current volume root directory.

ownername — A CHARACTER expression that specifies the ownername to be associated with the new directory. A value of all spaces indicates the current ownername of the task is used. When you create a resource definition, specifying an ownername different from yours does not change your ownername, it only specifies the ownername associated with the resource being created.

projectname — A CHARACTER expression that specifies the projectname associated with the directory. A value of all spaces indicates the projectname associated with the task is used.

owneraccess — A CHARACTER expression that specifies the directory owner's access rights. The access mode (with the exceptions of 'N' and '$\not\emptyset$') may be specified in any order as a concatenated string (e.g., 'ADR' or 'A'//'D'//'R'). Duplicates are not allowed.

'A'	Allows directory additions.
'E'	Allows directory deletions.
'R'	Allows directory contents to be read.
'T'	Allows directory to be traversed by using pathname.
'D'	Allows directory to be deleted.
'N'	Allows no directory access.
'$\not\emptyset$'	Blank allows all system defaults.

projectaccess — A CHARACTER expression that specifies the access rights of project groups having specific resource access privileges. The access mode (with the exception of 'N' and '$\not\emptyset$') may be specified in any order as a concatenated string (e.g., 'EAD' or 'E'//'A'//'D'). Duplicates are not allowed.

otheraccess — A CHARACTER expression that specifies the access rights of a user other than the owner or a project group associated with a resource. The access mode (with the exceptions of 'N' and 'b') may be specified in any order as a concatenated string (e.g., 'EAD' or 'E'//'A'//'D)'. Duplicates are not allowed.

'A'	Allows directory additions.
'E'	Allows directory deletions.
'R'	Allows directory contents to be read.
'T'	Allows directory to be traversed using pathname.
'D'	Allows directory to be deleted.
'N'	Allows no directory access.
'b'	Blank allows all system defaults.

shared — A LOGICAL expression that is either true or false. .TRUE. indicates the directory may be shared; .FALSE. specifies the directory can support only one user at a time (exclusive use).

maxsize — An INTEGER expression that specifies the maximum number of entries allowed in the directory. MPX-32 creates directories with the total entries rounded up to the number of entries which can fit into the granularity of the disk on which the directory is created.

start — An INTEGER expression that specifies the disc block number where the directory should start. If allocation in the desired space cannot be accomplished, the function is denied. If the value of this argument is zero, the directory may start at any available location.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.3 X_DDIR

The X_DDIR subroutine deletes a directory or a directory entry for a resource defined by the specified pathname.

Calling Sequence

```
CALL X_DDIR (pathname, istatus)
```

pathname — A CHARACTER expression that specifies the path to the target directory or file.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.4 X_DIRECT

The X_DIRECT subroutine changes your working directory. Your access rights in the new directory depend on the protection defined for the directory and whether you own the directory, supply a project group name to gain access rights, or are an arbitrary user of the directory. Once the directory has been changed, the new working directory is in effect until either this subroutine is called again or the program terminates.

Calling Sequence

CALL X_DIRECT (pathname,projectname,projectkey,istatus)

pathname — A CHARACTER expression that specifies the path to the target directory.

projectname — A CHARACTER expression that specifies the projectname associated with the directory. A value of all spaces indicates the projectname associated with the task is used.

projectkey — A CHARACTER expression of one to eight characters that specifies the projectkey.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.5 X_DISMNT

The X_DISMNT subroutine requests that a specified volume or unformatted medium be removed from a device.

Calling Sequence

CALL X_DISMNT (volumeid,istatus)

volumeid — A CHARACTER expression that specifies the volume identification of the volume to be dismounted.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.6 X_DPART

The X_DPART subroutine deletes the memory partition identified by the specified pathname. If the partition is currently allocated, its Allocated Resource Table Entry (ARTE) will be marked for deletion.

Calling Sequence

CALL X_DPART (pathname, istatus)

pathname — A CHARACTER expression that specifies the path to the memory partition.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.7 X_DPXMNT

The X_DPXMNT subroutine incorporates the DATAPOOL partition into a task's extended address area. This subroutine sets a globally defined base address for the DATAPOOL memory partition in the user's task and returns the status of the included memory partition.

Calling Sequence

```
Call X_DPXMNT (istatus)
```

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.8 X_EXCL

The X_EXCL subroutine removes a static or dynamic memory partition from a task's logical address space. A partition thus excluded cannot be referenced until it is again included (refer to X_INCLD).

Calling Sequence

```
CALL X_EXCL (pathname,ownername,istatus)
```

pathname — A CHARACTER expression left justified and blank filled, that specifies the path of the target partition. If just the partition name is specified, then the partition must reside in the system directory.

ownername — A CHARACTER expression of one to eight characters, left justified and blank filled, that specifies the ownername to be associated with the excluded partition. A value of all spaces indicates the absence of an ownername and the system default (i.e., your ownername) is used.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.9 X_EXTEND

The X_EXTEND subroutine obtains more space for an existing file when an End-Of-Medium (EOM) condition is encountered, provided the file was created with manual extendibility. If the request space exceeds the available contiguous space, the request is denied.

Calling Sequence

```
CALL X_EXTEND (pathname,inc,istatus)
     X_XTENDU (unit,inc,istatus)
     X_XTENDL (lfc,inc,istatus)
```

pathname — A CHARACTER expression that specifies the path to the target file.

unit — An INTEGER expression, from 1 through 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 CHARACTER string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 CHARACTER string.

inc — An INTEGER expression that specifies the size (in blocks) requested for the extend function. The number specified is rounded up to the allocation unit of the volume.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.10 X_INCLD

The X_INCLD subroutine allocates a static or dynamic memory partition into your logical address space, providing that space is not currently allocated.

Calling Sequence

```
CALL X_INCLD (pathname,ownername,access,shared,timeout,istatus)
```

pathname — A CHARACTER expression that specifies the path of the target partition.

ownername — A CHARACTER expression that specifies the ownername to be associated with the partition being transferred. A value of all spaces indicates the absence of an ownername, and the system default (i.e, your ownername) is used.

access — A CHARACTER expression indicating the access rights. One of the following may be specified:

'R'	Read only.
'RW'	Read or Write.

shared — A LOGICAL expression that is either true or false. .TRUE. indicates the memory partition may be shared; .FALSE. indicates the memory partition is exclusively requested.

timeout — An INTEGER expression that specifies the desired time to wait for the resource. Possible values are:

+n	Return immediately with defined istatus value.
0	Wait forever.
-n	Number of time units to wait.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.10A X_INCXDP

The X_INCXDP subroutine incorporates a datapool partition into a task's address area. This subroutine sets a globally defined base address for the datapool memory partition in the user's task and returns the status of the included memory partition. Although primarily designed for use with extended datapool partitions, it can also be used with non-extended datapool partitions.

Calling Sequence

CALL X_INCXDP (partition, ownername, access, shared, timeout, istatus)

partition — A CHARACTER expression whose value must be one of the memory partition names DP00L00 through DP00L99 or the standard datapool partition name DATAPOOL.

ownername — A CHARACTER expression that specifies the ownername to be associated with the partition being transferred. A value of all spaces indicates that the current ownername of the task is used.

access — A CHARACTER expression indicating the access rights. One of the following may be specified:

'R'	Read only
'RW'	Read or write

shared — A LOGICAL expression that is either true or false. .True. indicates the memory partition may be shared; .false. indicates the memory partition is exclusively requested.

timeout — An INTEGER expression that specifies the desired time to wait for the resource. Possible values are:

+n	Return immediately with defined istatus value
0	Wait forever
-n	Number of time units to wait

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.11 X_INQ

The X_INQ subroutine obtains information specific to a previously connected allocated resource. A series of pointers return the addresses of the various system data structures that describe the resource. Resources are identified by either a logical file code obtained when the resource was allocated or a memory partition name defined when the partition was created. Interpret the information in the identified structures as the application dictates.

Calling Sequence

```
CALL X_INQ (partition, ownername, array, istatus)
     X_INQU (unit, array, istatus)
     X_INQL (lfc, array, istatus)
```

partition — A CHARACTER expression of one to eight characters, left-justified and blank filled, that specifies the name of a shared memory partition.

ownername — A CHARACTER expression that specifies the ownername associated with the resource. A value of all spaces indicates the absence of an ownername, and the system default (i.e., your ownername) is used.

unit — An INTEGER expression, from 1 through 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 CHARACTER string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 CHARACTER string.

array — An eight-word INTEGER array that has the following structure:

Array element 1

An Allocated Resource Table (ART) address that describes the current resource allocation status in relation to other tasks in the system which may be sharing it or attempting to gain access to it.

Array element 2

A File Assignment Table (FAT) address that contains information pertinent to the access, current use, and status of the resource.

Array element 3

A Unit Definition Table (UDT) address that specifies the logical and physical characteristics of the peripheral device associated with the resource.

Array element 4

A Device Type Table (DDT) address that contains the ASCII representation of the device type code associated with the device pointed to by array element 3.

Array element 5

A Controller Definition Table (CDT) address that contains I/O processing information for the controller associated with the device described by array element 3.

Array element 6

A Shared Memory Table (SMT) address that describes the physical and logical characteristics of the shared memory region; applicable only to memory partitions.

Array element 7

A File Pointer Table (FPT) address that contains the logical file code associated with the resource.

Array element 8

A Mounted Volume Table (MVT) address that contains information pertinent to the current operating characteristics of the volume on which the resource resides; applicable only to volume resources.

istatus — An INTEGER variable that receives the status after the subroutine has been performed. Appendix B lists istatus values.

Programming Consideration

A parameter description area array element, containing a returned value of zero, implies that the corresponding structure does not apply to the resource for which the inquiry was made. For example, only elements 1, 6, and 8 apply for memory partitions. Therefore, each array element contains a zero for any resource other than a memory partition.

5.1.12 X_LOG

The X_LOG subroutine accepts a pathname and returns resource descriptors and directory entry information for the specified resource. There are two calls for this subroutine. The initial call, X_LOGI, initializes an internal structure that specifies the parameters passed. Depending upon the pathname defined, a specific resource or all the resources within a given directory are logged. If the pathname specifies that all resources within a directory are to be logged, X_LOGS must be used to obtain all entries after the initial call. Each call, including the initial call, X_LOGI, provides the log of one resource.

Calling Sequence

```
CALL X_LOGI (pathname,rdbuffer,length1,dirbuffer,length2,istatus)
CALL X_LOGS (rdbuffer,length1,dirbuffer,length2,istatus)
```

pathname — A CHARACTER expression that specifies the path to the target resource. If this parameter specifies a file name, only that file is logged. If this parameter specifies only the volume and directory parts, all files contained within the specified directory are logged.

rdbuffer — An INTEGER*8 array that returns the resource descriptor for the logged resource. Array rdbuffer must contain at least the number of array elements specified by length1.

length1 — An INTEGER expression that specifies the number of array elements to be returned in rdbuffer. The value of length1 must be 1 through 96.

dirbuffer — An INTEGER array that returns the directory entry. The first four words are the resource name; the remainder contains directory information (e.g., collision counts, etc.). Array dirbuffer must contain at least the number of array elements specified by length2.

length2 — An INTEGER expression that specifies the number of array elements to be returned in dirbuffer. The value of length2 must be 0 through 16.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. A value of -1 indicates all resources specified have been logged. Appendix B lists istatus values.

5.1.13 X_MDESC

The X_MDESC subroutine reads the resource descriptor into memory to modify it. This subroutine locks the descriptor so that any other attempt to access it will not be allowed until the descriptor is written to the volume. Therefore, X_WDESC must be called to unlock the descriptor. X_MDESC may be called only by the owner of the resource.

Calling Sequence

```
CALL X_MDESC (pathname,buffer,istatus)
     X_MDESCU (unit,buffer,istatus)
     X_MDESCL (lfc,buffer,istatus)
```

pathname — A CHARACTER expression specifying the path to the target resource.

unit — An INTEGER expression, from 1 through 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

buffer — A 96 doubleword integer array that returns the resource descriptor upon completion of this subroutine.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.14 X_MOUNT

The X_MOUNT subroutine requests that a volume or unformatted medium be mounted on a device. The volume can be declared as either shared or exclusive. If not currently mounted, a mount message appears on the console.

Calling Sequence

```
CALL X_MOUNT (volumeid,device,public,nomsg,shared,timeout,buffer,istatus)
```

volumeid — A CHARACTER expression that specifies the name of the volume to be mounted.

device — A CHARACTER expression that specifies the two character device mnemonic for the mount operation. In addition, the channel and subaddress can be specified.

public — A LOGICAL expression that is either true or false. .TRUE. specifies that the volume is to be mounted as public; .FALSE. specifies that the volume is to be mounted as nonpublic.

nomsg — A LOGICAL expression that is either true or false. .TRUE. specifies that no mount message is to be displayed; .FALSE. specifies that a mount message is to be displayed.

shared — A LOGICAL expression that is either true or false. .TRUE. indicates the volume may be shared; .FALSE. specifies that the volume is exclusively used.

timeout — An INTEGER expression that specifies the desired time to spend waiting for an available disc to mount the volume. Possible values are:

>0	Return immediately
=0	Wait forever
<0	Number of time units to wait

buffer — An INTEGER variable that is present for compatibility with previous releases. No information is returned in buffer.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.15 X_PERM

The X_PERM subroutine changes the status of a file allocated to the calling program from temporary to permanent. The temporary file must have been previously created and allocated with all of its file attributes; this subroutine does not establish attributes for a file.

Calling Sequence

```
CALL X_PERML (lfc,pathname,istatus)
     X_PERMU (unit,pathname,istatus)
```

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

unit — An INTEGER expression, from 1 to 999, that specify the logical file code of the resource.

pathname — A CHARACTER expression that specifies the pathname of the target file.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.16 X_PROJCT

The X_PROJCT subroutine changes a project group to gain specific access rights of the project group to resources.

Calling Sequence

```
CALL X_PROJCT (projectname,projectkey,istatus)
```

projectname — A CHARACTER expression that specifies the projectname associated with the directory. A value of all blanks indicates the projectname associated with the task is used and no change will occur.

projectkey — A CHARACTER expression of one to eight characters that specifies the projectkey. A value of all blanks indicates the absence of a projectkey.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.17 X_RDESC

The X_RDESC subroutine reads the resource descriptor of a file, directory, or memory partition, thus providing information about a resource.

Calling Sequence

```
CALL X_RDESC (pathname,buffer,istatus)
     X_RDESCU (unit,buffer,istatus)
     X_RDESCL (lfc,buffer,istatus)
```

pathname — A CHARACTER expression specifying the path to the target file, or memory partition.

unit — An INTEGER expression, from 1 to 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

buffer — A 96 doubleword integer array that returns the resource descriptor upon completion of this subroutine.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.18 X_RECON

The X_RECON subroutine reconstructs a pathname for an assigned volume resource.

Calling Sequence

```
CALL X_RECONU (unit,pathname,istatus)
CALL X_RECONL (lfc,pathname,istatus)
```

unit — An INTEGER expression, from 1 to 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an INTEGER constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

pathname — A CHARACTER buffer that will contain the converted pathname on return from the subroutine.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.19 X_REPLC

The X_REPLC subroutine assigns the resource descriptor of a permanent file to a temporary file. The original space occupied by the permanent file is released and the logical file code associated with the temporary file is deallocated. The permanent file may be open at the time X_REPLC is called. However, the file must not be shared.

Calling Sequence

```
CALL X_REPLCU (unit,pathname,istatus)
CALL X_REPLCL (lfc,pathname,istatus)
```

unit — An INTEGER expression, from 1 to 999, that specifies the logical file code of the resource. This is the unit number assigned to the temporary file.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string. This is the LFC assigned to the temporary file.

pathname — A CHARACTER expression that specifies the pathname of the permanent file to be replaced.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.20 X_RID

The X_RID subroutine returns the first eight words of the resource descriptor for a specified resource.

Calling Sequence

```
CALL X_RID (pathname,rid,istatus)
```

pathname — A CHARACTER expression that specifies the path to the target resource.

rid — An eight-word INTEGER array that receives the first eight words of the resource descriptor.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.21 X_RNAME

The X_RNAME subroutine renames existing files. The user must supply both the original pathname and the new pathname.

Calling Sequence

```
CALL X_RNAME (pathname,npathname,istatus)
```

pathname — A CHARACTER expression that specifies the path to the target file.

npathname — A CHARACTER expression that specifies the new pathname of the target resource.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

Programming Consideration

Files cannot be renamed across volumes.

5.1.22 X_TRUNC

The X_TRUNC subroutine deallocates all unused segments currently allocated to an extended disc file. The file can be either a temporary or permanent file; if a permanent file, it can be opened or closed when this subroutine is called. You must specify either 'write', 'update', or 'append' access when the file is created.

Calling Sequence

```
CALL  X_TRUNC  (pathname,istatus)
      X_TRUNCU (unit,istatus)
      X_TRUNCL (lfc,istatus)
```

pathname — A CHARACTER expression that specifies the path to the target file.

unit — An INTEGER expression, from 1 to 999, that specifies the logical file code of the resource.

lfc — An INTEGER or CHARACTER expression or variable that specifies the logical file code. If the LFC is an INTEGER, this argument must be an integer constant in the range of 1-999 or a 1-3 character string left justified and blank filled. If the LFC is a CHARACTER, this argument must be a 1-3 character string.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.1.23 X_WDESC

The X_WDESC subroutine writes the resource descriptor (modified by the X_MDESC subroutine) to the volume and unlocks the description to allow access.

Calling Sequence

```
CALL X_WDESC (buffer,istatus)
```

buffer — A 96 INTEGER*8 array that contains the resource descriptor which will be written to the volume.

istatus — An INTEGER variable that returns the status after the subroutine has been performed. Appendix B lists istatus values.

5.2 Device Type Codes

The following are the device type codes used for FORTRAN 77+ callable MPX-32 subroutines.

Device Type Code (Hex)	Decimal Equivalent	Two-Character Device Mnemonic	Device Description
00	00	CT	Operator console (not assignable)
01	01	DC	Any disc unit
02	02	DM	Any moving-head disc
03	03	DF	Any fixed-head disc
04	04	MT	Any magnetic tape unit
05	05	M9	Any nine-track magnetic tape unit
06	06	M7	Any seven-track magnetic tape unit
07	07	CD	Any card reader or card reader/punch
08	08	CR	Any card reader
09	09	CP	Any card punch
0A	10	LP	Any line printer
0B	11	PT	Any paper tape reader/punch
0C	12	TY	Any teletypewriter (other than console)
0D	13	CT	Operator console (assignable)
0E	14	FL	Floppy disc
0F	15	NU	Null device
10	16	CA	Communications adapter (binary synchronous/asynchronous)
11	17	U0	For user-defined application
12	18	U1	For user-defined application
13	19	U2	For user-defined application
14	20	U3	For user-defined application
15	21	U4	For user-defined application
16	22	U5	For user-defined application
17	23	U6	For user-defined application
18	24	U7	For user-defined application
19	25	U8	For user-defined application
1A	26	U9	For user-defined application
1B	27	LF	Lineprinter/floppy controller (used only with SYSGEN)

CHAPTER 6

SUPPORT FOR OTHER STANDARDS

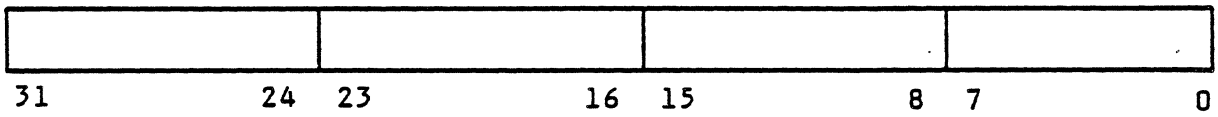
6.1 Support Subroutines and Functions

This chapter presents subroutines and functions that provide the features of the following FORTRAN 77+ related standards: ISA-S61.1, ISA-S61.2, and MIL-STD-1753.

6.2 Bit Field Manipulation

The following subprograms constitute support for the ANSI/ISA S61.1-1976 and MIL-STD-1753 bit manipulation capability.

The functions allow the programmer to view INTEGER data as ordered sets of bits (a_{31},a_{30},..., a_0). Note that bits are numbered from RIGHT to LEFT. The rightmost bit position is zero.



6.2.1 Logical Operations

In the following functions, j and m are INTEGER expressions. Operations are performed on all corresponding bits of the operands. The numeric storage unit is a 32-bit word.

6.2.1.1 IAND (Bit-wise Logical AND Function)

Calling Sequence

iresult = IAND (j, m)

where the result of IAND (j, m) is:

$$\sum_{k=0}^{31} 2^k * (j_k * m_k)$$

6.2.1.2 IEOR (Bit-wise Exclusive OR Function)

Calling Sequence

iresult = IEOR (j, m)

where the result of IEOR (j, m) is:

$$\sum_{k=0}^{31} 2^k * (2 - (j_k + m_k)) * (j_k + m_k)$$

6.2.1.3 IOR (Bit-wise Logical OR Function)

Calling Sequence

iresult = IOR (j, m)

where the result of IOR (j, m) is:

$$\sum_{k=0}^{31} 2^k * ((j_k + m_k) - (j_k * m_k))$$

6.2.1.4 NOT (Bit-wise Logical Complement Function)

Calling Sequence

iresult = NOT (j)

where the result of NOT (j) is:

$$\sum_{k=0}^{31} 2^k * (1-j_k)$$

6.2.2 Shift Operations

The shift operations provided are logical shift and circular shift; these operations are implemented as INTEGER functions. The arguments may be INTEGER constants, INTEGER variables, INTEGER array elements, or INTEGER expressions. The arguments ivalue and ishftcnt are defined as follows:

ivalue — The value (binary pattern) to be shifted.

ishftcnt — The shift count is specified as follows:

ishftcnt > 0	Left shift.
ishftcnt = 0	No shift.
ishftcnt < 0	Right shift.

If the absolute value of the shift count is greater than the number of bits in a numeric storage unit, the result is undefined. The shift functions do not alter the values of any of their arguments.

6.2.2.1 ISHFT

Calling Sequence

iresult = ISHFT (ivalue, ishftcnt)

All 32 bits representing the argument ivalue are shifted ishftcnt places. If ishftcnt > 0, shift to the left; if ishftcnt < 0, shift to the right. Bits shifted out, from either the left or right end, are lost. Zeros are shifted in from the opposite end.

6.2.2.2 ISHFTC

Calling Sequence

iresult = ISHFTC (ivalue, ishftcnt, isub)

The rightmost isub bits of the ivalue argument are shifted circularly ishftcnt places (i.e., the bits shifted out from one end of the subfield isub are shifted into the opposite end). The shifted bits are combined with the leftmost 32-isub unshifted bits from ivalue, and the resulting value is returned by this function. No bits are lost. The absolute value of the argument ishftcnt must be less than or equal to isub. The argument isub must be greater than or equal to one and less than or equal to 32.

6.2.3 Bit Processing

Individual bits of numeric storage units can be tested and changed with the following routines for bit processing. The functions have two arguments, ivalue and ibit, which are INTEGER expressions.

ivalue Binary pattern.

ibit Position (rightmost bit is bit 0).

6.2.3.1 BTEST

The BTEST LOGICAL function tests a specified bit of an INTEGER.

Calling Sequence

iresult = BTEST (ivalue, ibit)

The bit specified by ibit of argument ivalue is tested. If it is one, the value of the function is .TRUE.; if it is zero, the value of the function is .FALSE..

6.2.3.2 IBSET

The IBSET INTEGER function returns the value of a specified INTEGER with a specified bit set.

Calling Sequence

iresult = IBSET (ivalue, ibit)

The value of the IBSET function is equal to the value of ivalue, with the ibit set to one.

6.2.3.3 IBCLR

The IBCLR INTEGER function returns the value of a specified INTEGER with a specified bit reset.

Calling Sequence

iresult = !BCLR (ivalue, ibit)

The result of the IBCLR function is equal to the value of ivalue, with the ibit set to zero.

6.2.4 Bit Subfields

Bit subfields are referenced by specifying both a bit position and a length. Bit positions within a numeric storage unit are numbered from RIGHT to LEFT. The rightmost bit position is zero. Bit fields may not extend from one 32-bit word into another 32-bit word and the length of a field must be greater than zero.

6.2.4.1 IBITS

The IBITS function extracts a specified bit subfield from a specified argument. The result field is right justified and the remaining bits in the returned value are set to zero.

Calling Sequence

iresult = IBITS (ivalue, isrcpos, len)

ivalue, isrcpos, len INTEGER expressions.

A bit field is extracted from the value of ivalue, starting from bit position isrcpos and extending left for len bits. The value of isrcpos+len must be less than or equal to 32.

6.2.4.2 MVBITS

The, MVBITS subroutine moves a bit subfield from a specified value to another bit subfield in a specified destination variable or array element.

Calling Sequence

CALL MVBITS (ivalue, isrcpos, len, iresult, idestpos)

ivalue, isrcpos, len, idestpos INTEGER expressions.

iresult An INTEGER variable or array element.

Len bits from positions isrcpos through isrcpos+len-1 of argument ivalue are moved to positions idestpos through idestpos+len-1 of argument iresult. The portion of argument iresult not affected by the movement of bits remains unchanged. Arguments ivalue and iresult may be the same numeric storage unit. The values of isrcpos+len and idestpos+len must be less than or equal to 32.

6.3 Time and Date

6.3.1 TIME (Obtain Time-of-Day)

The TIME subroutine allows a program to determine the current time of day.

Calling Sequence

```
CALL TIME (itime)
```

itime An INTEGER array that contains, upon return, the absolute time of day.

itime(1)	Hours (0-23)
itime(2)	Minutes (0-59)
itime(3)	Seconds (0-59)

6.3.2 DATE (Obtain Date)

The DATE subroutine allows the calling program to determine the current calendar date.

Calling Sequence

```
CALL DATE (idate)
```

idate An INTEGER array that contains, upon return, the date.

idate(1)	A.D. year since zero
idate(2)	Month (1-12)
idate(3)	Day (1-31)

6.4 Task Control Calls

Task control calls control the operation of programs within the system. Through these external procedures, the execution of programs are started, stopped, or delayed.

6.4.1 START

The START subroutine executes the designated load module after the expiration of the specified delay. The load module must be in the system directory.

Calling Sequence

```
CALL START (loadmod, idelay, iunits, istatus)
```

loadmod — An INTEGER array that specifies the name of the program to be executed. The array must contain a left-justified, blank-filled ASCII doubleword.

idelay — An INTEGER expression that specifies the minimum length of time, in units specified by iunits, to delay before executing the program. If the value of idelay is zero or negative, the requested program is run as soon as possible.

iunits — An INTEGER expression that specifies the units of time as follows:

0	Basic counts of the system's real-time clock.
1	Milliseconds.
2	Seconds.
3	Minutes.

istatus — An INTEGER variable or INTEGER array element, set upon return to the calling program, that indicates the disposition of the request as follows:

1	Request accepted.
2 or greater	Request rejected.

6.4.2 TRNON

The TRNON subroutine executes the designated load module at a specified time of day. The load module must be located in the system directory.

Calling Sequence

```
CALL TRNON (loadmod, itime, istatus)
```

loadmod — An INTEGER *8 variable that specifies the load module to be executed. This argument must be one to eight ASCII characters, left-justified, and blank-filled.

itime — An INTEGER array that contains the absolute time of day at which the specified program is to be executed as follows:

itime(1)	Hours (0 to 23)
itime(2)	Minutes (0 to 59)
itime(3)	Seconds (0 to 59)

istatus — An INTEGER variable or INTEGER array element, set on return to the calling program, that indicates the disposition of the request as follows:

1	Request accepted.
2 or greater	Request rejected.

6.4.3 WAIT

The WAIT subroutine delays the continuation of the calling program for a specified length of time.

Calling Sequence

```
CALL WAIT (idelay, iunits, istatus)
```

idelay — An INTEGER expression that specifies the length of time, in units specified by iunits, to delay before returning to the calling procedure. If the value of idelay is zero or negative, no delay occurs. If the number of time units specified is less than one system time unit, the task is not suspended and a status of two is returned.

iunits — An INTEGER expression that specifies units of time as follows:

0	Basic counts of the system's real-time clock.
1	Milliseconds.
2	Seconds.
3	Minutes.

istatus — An INTEGER variable or INTEGER array element that is set on return to the calling program, indicating the disposition of the request as follows:

1	Request accepted.
2 or greater	Delay as specified has not occurred.

6.5 File Access

The following subroutines allow the user to create, open, close, and delete files, as defined by the ISA standard. Because of the unique restrictions placed on files under the ISA standard, the file must be created and manipulated internally by the task, using only the following ISA-related file access routines.

6.5.1 CFILW

The CFILW creates a permanent file that remains defined to the system until it is deleted. Files established by this subroutine have no privacy attribute to restrict a concurrent program from accessing the files. The contents of the newly created file is all zeros.

Calling Sequence

CALL CFILW (pathname, ibytes, irecords, istatus)

pathname — A CHARACTER expression or CHARACTER variable that specifies the full pathname of the file. For programs run in compatible mode, this argument is an INTEGER*8 variable that contains the one to eight ASCII character, left justified and blank filled name of the file.

ibytes — An INTEGER expression that specifies the number of bytes per record in the file.

irecords — An INTEGER expression that specifies the number of records in the file.

istatus — An INTEGER variable that is set on return to the calling program. The argument indicates the disposition of the request as follows:

Native	Compatible	
1	1	File created.
2	2	File already exists.
36	5	Disc space unavailable.
26	10	File name contains invalid characters.
29	N/A	Directory not found.

6.5.2 DFILW

The DFILW subroutine deletes a specified permanent file. Any file created by CFILW can be deleted by DFILW. There is no protection against deleting a file that is currently open to another program.

Calling Sequence

CALL DFILW (pathname, istatus)

pathname A CHARACTER expression or CHARACTER variable that specifies the full pathname of the file. For programs run in compatible mode, this argument is an INTEGER*8 variable that contains the one to eight ASCII character, left justified and blank filled name of the file.

istatus An INTEGER variable that is set on return to the calling program. This argument indicates the disposition of the request as follows:

1 File deleted.
2 A file of the specified name does not exist.

6.5.3 OPENW

The OPENW subroutine associates the unit (logical file code) specified by the calling program with the named file and defines the desired access privilege of that program to the file. A file can be opened only if it exists.

Calling Sequence

```
CALL OPENW (iunit, pathname, iuse, istatus)
```

iunit — An INTEGER expression that specifies the unit number of the file.

pathname — A CHARACTER expression or CHARACTER variable that specifies the full path name of the file for programs run in native mode. For programs run in compatible mode, this argument is an INTEGER*8 variable that contains the one to eight ASCII character, left justified and blank filled name of the file.

iuse — An INTEGER expression that declares the program's intended file use as follows:

1	Read only	The current program can read but not write; other concurrent programs can read and write.
2	Shared	The calling program can read or write; other concurrent programs can also read or write.
3	Protected read	The calling program can read only; other concurrent programs can read only.
4	Exclusive all	The calling program can read or write; all other concurrent programs cannot read or write.

istatus — An INTEGER variable that is set on return to the calling program. The argument indicates the disposition of the request as follows:

1	File opened.
11	File specified does not exist or was not created in this run.

6.5.4 CLOSEW

The CLOSEW subroutine ends the calling program's association of the specified unit number with a named file.

Calling Sequence

```
CALL CLOSEW (iunit, istatus)
```

iunit — An INTEGER expression that specifies the unit number of the file.

istatus — An INTEGER variable that is set on return to the calling program. The argument indicates the disposition of the request as follows:

1	File closed.
2	File not closed.

6.6 Unformatted Random I/O

6.6.1 RDRW

The RDRW subroutine allows the transfer of one data record from a file. The file is treated as a direct access file for selection of the record. The calling program must open the file with the OPENW subroutine and must currently have read access privileges.

Calling Sequence

CALL RDRW (iunit, irecord, idata, imaxtran, istatus)

iunit — An INTEGER expression that specifies the unit number of the file.

irecord — An INTEGER expression that specifies the record number to be read. (The record number must be positive.)

idata — An INTEGER variable, array element, or array name with a byte address that designates the first variable into which information is to be placed.

imaxtran — An INTEGER expression that specifies the maximum number of bytes that can be transferred.

istatus — An INTEGER variable that is set on return to the calling program. The argument indicates the disposition of the request as follows:

1	Data transfer completed successfully.
2	Data transfer failed.
3	Data buffer (idata) is not a byte address; data transfer may have failed.
67	Illegal random access.

6.6.2 WRTRW

The WRTRW subroutine writes unformatted direct access information into files that have previously been opened by the calling program. The file is treated as a direct access file for selection of the record. The calling program must have opened the file with the OPENW subroutine and must currently have write access privileges.

Calling Sequence

CALL WRTRW (iunit, irecord, idata, imaxtran, istatus)

iunit — An INTEGER expression that specifies the unit number of the file.

irecord — An INTEGER expression that specifies the record number to be written. (The record number must be positive.)

idata — An INTEGER variable, array element, or array name with a byte address that designates the first variable from which information is to be obtained.

imaxtran — An INTEGER expression that specifies the maximum number of bytes that can be transferred.

istatus — An INTEGER variable that is set on return to the calling program. The argument indicates the disposition of the request as follows:

1	Data transfer completed successfully.
2	Data transfer failed.
3	Data buffer (idata) is not a byte address; data transfer may have failed.
67	Illegal random access.

CHAPTER 7

SUBROUTINE AND FUNCTION CALLING CONVENTIONS

7.1 Calling FORTRAN Subroutines from Assembly Language Programs

The Gould CSD FORTRAN 77+ compiler generates three different types of calling protocols for CALL statements. The code generated is determined by the number of parameters. For an assembler program to call a FORTRAN program, it must use the same calling protocol as the FORTRAN program.

Function and subroutine calling conventions are the same, except the function value is returned in one or more registers. Refer to the FORTRAN 77+ Release 4.X Reference Manual for specifics.

7.1.1 Calling a FORTRAN Subroutine with No Parameters

To call a FORTRAN subroutine with no parameters:

1. Declare the subroutine name external (Assembly statement EXT) .
2. Branch and link to the subroutine (Assembly operation BL) .

The FORTRAN subroutine returns control to the assembler routine at the word following the branch and link.

7.1.2 Calling a FORTRAN Subroutine with One Parameter

To call a FORTRAN subroutine with one parameter:

1. Declare the subroutine name external (Assembly statement EXT).
2. Load register 1 with the parameter address.
3. Branch and link to the FORTRAN subroutine (Assembly operation BL).

The FORTRAN subroutine returns control to the assembler routine at the word following the branch and link.

Programming Consideration

The parameter storage location in the assembler routine should start on a boundary compatible with the type of FORTRAN variable associated with the parameter.

7.1.3 Calling a FORTRAN Subroutine with Two or More Parameters

To call a FORTRAN subroutine with more than one parameter:

1. Declare the subroutine name external (Assembly statement EXT).
2. Set up a parameter area immediately after the branch and link (see below).
3. Branch and link to the FORTRAN subroutine (Assembly operation BL).

Parameter Area:

```
BL          FORTRAN subroutine name
DATAW nW    n is number of parameters; w is 4 bytes per parameter word
ACx         Address of first parameter
ACx         Address of second parameter
 .
 .
 .
ACx         Address of last parameter
```

The calling parameter types in the first byte of the word address in the calling list are:

Type Code	Type
0	INTEGER*1
1	INTEGER*2
2	INTEGER*4
3	INTEGER*8
4	REAL*4
5	REAL*8
6	COMPLEX*8
7	COMPLEX*16
8	Undefined
9	LOGICAL*1
A	LOGICAL*4
B	CHARACTER

Bit addressing is not implemented.

If set, bit 8 indicates an array address. It will be followed by an additional parameter word pointing to the location containing the number of elements in the array being passed (valid for call to A.TF and A.TU only).

If set, bit 12, with an address of absolute zero, indicates a null or nonexistent argument.

For CHARACTER type arguments, two parameters per argument are passed. The first parameter word is an address pointing to the starting address of the string argument. The next parameter word is an address pointing to the location containing the number of characters (bytes) in the argument.

ACx must be coded as either ACB, ACH, ACW, or ACD for parameters that are byte, halfword, word, or doubleword, respectively.

The FORTRAN subroutine returns control to the main or calling program at the first word after the parameter area.

7.2 Example: Assembler Routine Calling FORTRAN Subroutine

The following is an example of an assembler routine calling a FORTRAN subroutine with two parameters:

```
$JOB EXAMPLE          PROGRAMMER, KEY
$OPTION 2 5
$FORT77
        SUBROUTINE PRINTER(WORD, VALUE)
        INTEGER WORD                    !WORD TO BE PRINTED
        INTEGER VALUE                   !INDEX OF WORD
*** WRITE THE INDEX AND THE WORD
        WRITE('LO',10) VALUE,WORD
10      FORMAT(1X,I1,' ',A4)
        END
$OPTION 2 5
$ASSEMBLE
        PROGRAM CALLER
        M.REQS
***THIS PROGRAM LOOPS TILL COUNTER MATCHES CERTAIN VALUE
***THE FORTRAN PROGRAM IS CALLED TO PRINT THE VALUES
          DEF     CALLER        ALLOWS EXTERNAL REFERENCE
          EXT     PRINTER       ALLOWS REFERENCE
CALLER    EQU     $             DEMONSTRATE SOME FLEXIBILITY IN FORMAT
          ZR      R1            BIAS REG.
          ZR      R3            VALUE OF ITERATIONS IN BYTES COUNT
          LNW     R2,COUNT      GET NUMBER OF ITERATIONS
LOOP      LW      R5,DATA,R1
          STW     R5,DATUM
          STW     R3,VALUE
          STF     R0,SAVEREGS   JUST IN CASE SUBROUTINE CHANGES
CALLING   BL      PRINTER       TRANSFER TO THE PRINTER
          DATAW   2W            NUMBER OF PARMS IS 2
          ACW     DATUM         TELL SUBROUTINE THAT DATUM IS WORD
          ACW     VALUE         TELL SUBROUTINE THAT VALUE IS WORD
          LF      R0,SAVEREGS   CONTROL RETURNS HERE
          ADI     R3,1          INC PRINT VALUE
          ADI     R1,1W         INC
          BIW     2,LOOP        LOOP AGAIN
**TO GET HERE WE FELL OUT OF LOOP
          SVC     1,X'55'       EXIT TO MPX
***DATA SECTION
COUNT     DATAW   10W
DATA      DATAW   C'WRD0',C'WRD1',C'WRD2',C'WRD3',C'WRD4'
          DATAW   C'WRD5',C'WRD6',C'WRD7',C'WRD8',C'WRD9'
VALUE     RES     1W
DATUM     RES     1W
SAVEREGS          RES           1F
          END     CALLER
```

```
$CATALOG
AS LO TO SLO
BUILD TEMP
PROGRAM PRINTER
PROGRAM CALLER
$EOJ
$$
```

CHAPTER 8

RUN-TIME I/O TABLES AND BUFFERS

8.1 L.BB4

The SRTL module L.BB4 contains I/O buffers for File Control Blocks (FCB) and Allocation Table Entries. The value N.ENTRYS determines the number of 38W File Control Blocks and 22W allocation table entries the user's task will get.

By modifying the source of L.BB4, increasing or decreasing the value N.ENTRYS, the user can tailor the overall size of the I/O library included in tasks cataloged with this library. The default for N.ENTRYS is set at 30.

Once the user modifies the source for L.BB4, the module must be reassembled and placed into the user's run-time library.

The following is reprinted from the source of L.BB4 for use as a reference:

Run-time static memory pool constant definitions.

The parameter 'N.ENTRYS' may be changed to allow additional or fewer entries in the FCB and allocation tables. 'N.ENTRYS' restricts the total number of files that can be either dynamically allocated and/or open at any given time.

```
N.ENTRYS        EQU     30                      Total number of FCB and allocation
                                                table entries
FCBSIZE         EQU     38W

ALCENTSZ        EQU     22W

File Control Block Table

                BOUND   1D
F.CB            EQU     $                       FCB table beginning address
                REZ     FCBSIZE*N.ENTRYS        FCB table reserved area
F.CL            EQU     $                       FCB table ending address

Allocation Table

A.LCTBL         EQU     $                       Allocation table beginning address
                REZ     ALCENTSZ*N.ENTRYS       Allocation table reserved area
A.LCTEND        EQU     $                       Allocation table ending address
```

8.2 L.DIOBUF

The SRTL module L.DIOBUF contains blocking buffers for Direct Access I/O. The value N.BUFFERS determines the number of 192W Blocking Buffers the user's task will get.

By modifying the source of L.DIOBUF, increasing or decreasing the value N.BUFFERS, the user can tailor the overall size of the I/O library included in tasks cataloged with this library. The default for N.BUFFERS is 6.

Once the user modifies the source for L.DIOBUF, the module must be reassembled and placed into the user's runtime library.

The following is reprinted from the source of L.DIOBUF for use as a reference:

```
        PROGRAM  L.DIOBUF
        LIST     NODATA

        Definitions

        DEF      B.MAP     I/O BUFFER FREE LIST BIT MAP
        DEF      B.NM      SAVE AREA FOR # BLOCKING BUFFERS
        DEF      B.SIZ     SIZE OF ONE BUFFER
        DEF      B:BL      BLOCKING BUFFER END ADDRESS
        DEF      B:BN      BLOCKING BUFFER BEGIN ADDRESS AND
                           LENGTH OF BLOCKING BUFFERS AREA
```

The parameter 'N.BUFFERS' may be changed to allow additional or fewer blocking buffers used in Direct Access I/O. 'N.BUFFERS' restricts the total number of blocking buffers that can be used by a single FORTRAN task at any given time for direct access I/O from a minimum of 1 to a maximum of 32.

```
N.BUFFERS    EQU   6

Blocking Buffers

B:BN         DATAW    192W*N.BUFFERS      Beginning and length of block buff area
             REZ      192W*N.BUFFERS-1W
B:BL         EQU      $-1W                End of blocking buffer area
B.SIZ        DATAW    192W                Size of one buffer
B.NM         DATAW    0                   Number of blocking buffers
B.MAP        DATAW    0                   Buffer free list bit map
             END
```

APPENDIX A

ASCII CODE SET

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
00	0		12-0-9-8-1	0000 0000	NUL
01	1		12-9-1	0000 0001	SOH
02	2		12-9-2	0000 0010	STX
03	3		12-9-3	0000 0011	ETX
04	4		9-7	0000 0100	EOT
05	5		0-9-8-5	0000 0101	ENQ
06	6		0-9-8-6	0000 0110	ACK
07	7		0-9-8-7	0000 0111	BEL
08	8		11-9-6	0000 1000	BS
09	9		12-9-5	0000 1001	HT
0A	10		0-9-5	0000 1010	LF
0B	11		12-9-8-3	0000 1011	VT
0C	12		12-9-8-4	0000 1100	FF
0D	13		12-9-8-5	0000 1101	CR
0E	14		12-9-8-6	0000 1110	SO
0F	15		12-9-8-7	0000 1111	SI
10	16		12-11-9-8-1	0001 0000	DLE
11	17		11-9-1	0001 0001	C1
12	18		11-9-2	0001 0010	DC2
13	19		11-9-3	0001 0011	DC3
14	20		9-8-4	0001 0100	DC4
15	21		9-8-5	0001 0101	NAK
16	22		9-2	0001 0110	SYN
17	23		0-9-6	0001 0111	ETB
18	24		11-9-8	0001 1000	CAN
19	25		11-9-8-1	0001 1001	EM
1A	26		9-8-7	0001 1010	SUB
1B	27		0-9-7	0001 1011	ESC
1C	28		11-9-8-4	0001 1100	FS
1D	29		11-9-8-5	0001 1101	GS
1E	30		11-9-8-6	0001 1110	RS
1F	31		11-9-8-7	0001 1111	US

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
20	32			0010 0000	SP
21	33	!	12-8-7	0010 0001	Exclamation point
22	34	"	8-7	0010 0010	Quotation marks
23	35	#	8-3	0010 0011	Number sign
24	36	$	11-8-3	0010 0100	Dollar sign
25	37	%	0-8-4	0010 0101	Percent
26	38	&	12	0010 0110	Ampersand
27	39	'	8-5	0010 0111	Apostrophe
28	40	(	12-8-5	0010 1000	Opening parenthesis
29	41	)	11-8-5	0010 1001	Closing parenthesis
2A	42	*	11-8-4	0010 1010	Asterisk
2B	43	+	12-8-6	0010 1011	Plus
2C	44	,	0-8-3	0010 1100	Comma
2D	45	-	11	0010 1101	Hyphen
2E	46	.	12-8-3	0010 1110	Period
2F	47	/	0-1	0010 1111	Slant
30	48	0	0	0011 0000	Zero
31	49	1	1	0011 0001	One
32	50	2	2	0011 0010	Two
33	51	3	3	0011 0011	Three
34	52	4	4	0011 0100	Four
35	53	5	5	0011 0101	Five
36	54	6	6	0011 0110	Six
37	55	7	7	0011 0111	Seven
38	56	8	8	0011 1000	Eight
39	57	9	9	0011 1001	Nine
3A	58	:	8-2	0011 1010	Colon
3B	59	;	11-8-6	0011 1011	Semicolon
3C	60	<	12-8-4	0011 1100	Less than
3D	61	=	8-6	0011 1101	Equals
3E	62	>	0-8-6	0011 1110	Greater than
3F	63	?	0-8-7	0011 1111	Question mark
40	64	@	8-4	0100 0000	Commercial at
41	65	A	12-1	0100 0001	Uppercase A
42	66	B	12-2	0100 0010	Uppercase B
43	67	C	12-3	0100 0011	Uppercase C
44	68	D	12-4	0100 0100	Uppercase D
45	69	E	12-5	0100 0101	Uppercase E
46	70	F	12-6	0100 0110	Uppercase F
47	71	G	12-7	0100 0111	Uppercase G

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
48	72	H	12-8	0100 1000	Uppercase H
49	73	I	12-9	0100 1001	Uppercase I
4A	74	J	11-1	0100 1010	Uppercase J
4B	75	K	11-2	0100 1011	Uppercase K
4C	76	L	11-3	0100 1100	Uppercase L
4D	77	M	11-4	0100 1101	Uppercase M
4E	78	N	11-5	0100 1110	Uppercase N
4F	79	O	11-6	0100 1111	Uppercase O
50	80	P	11-7	0101 0000	Uppercase P
51	81	Q	11-8	0101 0001	Uppercase Q
52	82	R	11-9	0101 0010	Uppercase R
53	83	S	0-2	0101 0011	Uppercase S
54	84	T	0-3	0101 0100	Uppercase T
55	85	U	0-4	0101 0101	Uppercase U
56	86	V	0-5	0101 0110	Uppercase V
57	87	W	0-6	0101 0111	Uppercase W
58	88	X	0-7	0101 1000	Uppercase X
59	89	Y	0-8	0101 1001	Uppercase Y
5A	90	Z	0-9	0101 1010	Uppercase Z
5B	91	[	12-8-2	0101 1011	Opening bracket
5C	92	\	0-8-2	0101 1100	Reverse slant
5D	93	]	11-8-2	0101 1101	Closing bracket
5E	94	^	11-8-7	0101 1110	Circumflex
5F	95	_	0-8-5	0101 1111	Underline
60	96	`	8-1	0110 0000	Accent grave
61	97	a	12-0-1	0110 0001	Lowercase a
62	98	b	12-0-2	0110 0010	Lowercase b
63	99	c	12-0-3	0110 0011	Lowercase c
64	100	d	12-0-4	0110 0100	Lowercase d
65	101	e	12-0-5	0110 0101	Lowercase e
66	102	f	12-0-6	0110 0110	Lowercase f
67	103	g	12-0-7	0110 0111	Lowercase g
68	104	h	12-0-8	0110 1000	Lowercase h
69	105	i	12-0-9	0110 1001	Lowercase i
6A	106	j	12-11-1	0110 1010	Lowercase j
6B	107	k	12-11-2	0110 1011	Lowercase k
6C	108	l	12-11-3	0110 1100	Lowercase l
6D	109	m	12-11-4	0110 1101	Lowercase m
6E	110	n	12-11-5	0110 1110	Lowercase n
6F	111	o	12-11-6	0110 1111	Lowercase o

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
70	112	p	12-11-7	0111 0000	Lowercase p
71	113	q	12-11-8	0111 0001	Lowercase q
72	114	r	12-11-9	0111 0010	Lowercase r
73	115	s	11-0-2	0111 0011	Lowercase s
74	116	t	11-0-3	0111 0100	Lowercase t
75	117	u	11-0-4	0111 0101	Lowercase u
76	118	v	11-0-5	0111 0110	Lowercase v
77	119	w	11-0-6	0111 0111	Lowercase w
78	120	x	11-0-7	0111 1000	Lowercase x
79	121	y	11-0-8	0111 1001	Lowercase y
7A	122	z	11-0-9	0111 1010	Lowercase z
7B	123	{	12-0	0111 1011	Opening brace
7C	124	|	12-11	0111 1100	Vertical Line
7D	125	}	11-0	0111 1101	Closing brace
7E	126	~	11-0-1	0111 1110	Tilde
7F	127		12-9-7	0111 1111	DEL
80	128		11-0-9-8-1	1000 0000	
81	129		0-9-1	1000 0001	
82	130		0-9-2	1000 0010	
83	131		0-9-3	1000 0011	
84	132		0-9-4	1000 0100	
85	133		11-9-5	1000 0101	
86	134		12-9-6	1000 0110	
87	135		11-9-7	1000 0111	
88	136		0-9-8	1000 1000	
89	137		0-9-8-1	1000 1001	
8A	138		0-9-8-2	1000 1010	
8B	139		0-9-8-3	1000 1011	
8C	140		0-9-8-4	1000 1100	
8D	141		12-9-8-1	1000 1101	
8E	142		12-9-8-2	1000 1110	
8F	143		11-9-8-3	1000 1111	
90	144		12-11-0-9-8-1	1001 0000	
91	145		9-1	1001 0001	
92	146		11-9-8-2	1001 0010	
93	147		9-3	1001 0011	
94	148		9-4	1001 0100	
95	149		9-5	1001 0101	
96	150		9-6	1001 0110	
97	151		12-9-8	1001 0111	

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
98	152		9-8	1001 1000	
99	153		9-8-1	1001 1001	
9A	154		9-8-2	1001 1010	
9B	155		9-8-3	1001 1011	
9C	156		12-9-4	1001 1100	
9D	157		11-9-4	1001 1101	
9E	158		9-8-6	1001 1110	
9F	159		11-0-9-1	1001 1111	
A0	160		12-0-9-1	1010 0000	
A1	161		12-0-9-2	1010 0001	
A2	162		12-0-9-3	1010 0010	
A3	163		12-0-9-4	1010 0011	
A4	164		12-0-9-5	1010 0100	
A5	165		12-0-9-6	1010 0101	
A6	166		12-0-9-7	1010 0110	
A7	167		12-0-9-8	1010 0111	
A8	168		12-8-1	1010 1000	
A9	169		12-11-9-1	1010 1001	
AA	170		12-11-9-2	1010 1010	
AB	171		12-11-9-3	1010 1011	
AC	172		12-11-9-4	1010 1100	
AD	173		12-11-9-5	1010 1101	
AE	174		12-11-9-6	1010 1110	
AF	175		12-11-9-7	1010 1111	
B0	176		12-11-9-8	1011 0000	
B1	177		11-8-1	1011 0001	
B2	178		11-0-9-2	1011 0010	
B3	179		11-0-9-3	1011 0011	
B4	180		11-0-9-4	1011 0100	
B5	181		11-0-9-5	1011 0101	
B6	182		11-0-9-6	1011 0110	
B7	183		11-0-9-7	1011 0111	
B8	184		11-0-9-8	1011 1000	
B9	185		0-8-1	1011 1001	
BA	186		12-11-0	1011 1010	
BB	187		12-11-0-9-1	1011 1011	
BC	188		12-11-0-9-2	1011 1100	
BD	189		12-11-0-9-3	1011 1101	
BE	190		12-11-0-9-4	1011 1110	
BF	191		12-11-0-9-5	1011 1111	

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
C0	192		12-11-0-9-6	1100 0000	
C1	193		12-11-0-9-7	1100 0001	
C2	194		12-11-0-9-8	1100 0010	
C3	195		12-0-8-1	1100 0011	
C4	196		12-0-8-2	1100 0100	
C5	197		12-0-8-3	1100 0101	
C6	198		12-0-8-4	1100 0110	
C7	199		12-0-8-5	1100 0111	
C8	200		12-0-8-6	1100 1000	
C9	201		12-0-8-7	1100 1001	
CA	202		12-11-8-1	1100 1010	
CB	203		12-11-8-2	1100 1011	
CC	204		12-11-8-3	1100 1100	
CD	205		12-11-8-4	1100 1101	
CE	206		12-11-8-5	1100 1110	
CF	207		12-11-8-6	1100 1111	
D0	208		12-11-8-7	1101 0000	
D1	209		11-0-8-1	1101 0001	
D2	210		11-0-8-2	1101 0010	
D3	211		11-0-8-3	1101 0011	
D4	212		11-0-8-4	1101 0100	
D5	213		11-0-8-5	1101 0101	
D6	214		11-0-8-6	1101 0110	
D7	215		11-0-8-7	1101 0111	
D8	216		12-11-0-8-1	1101 1000	
D9	217		12-11-0-1	1101 1001	
DA	218		12-11-0-2	1101 1010	
DB	219		12-11-0-3	1101 1011	
DC	220		12-11-0-4	1101 1100	
DD	221		12-11-0-5	1101 1101	
DE	222		12-11-0-6	1101 1110	
DF	223		12-11-0-7	1101 1111	
E0	224		12-11-0-8	1110 0000	
E1	225		12-11-0-9	1110 0001	
E2	226		12-11-0-8-2	1110 0010	
E3	227		12-11-0-8-3	1110 0011	
E4	228		12-11-0-8-4	1110 0100	
E5	229		12-11-0-8-5	1110 0101	
E6	230		12-11-0-8-6	1110 0110	
E7	231		12-11-0-8-7	1110 0111	

Hexadecimal	Decimal	ASCII Graphic	Card Code (IBM029)	Internal Binary	ASCII Name
E8	232		12-0-9-8-2	1110 1000	
E9	233		12-0-9-8-3	1110 1001	
EA	234		12-0-9-8-4	1110 1010	
EB	235		12-0-9-8-5	1110 1011	
EC	236		12-0-9-8-6	1110 1100	
ED	237		12-0-9-8-7	1110 1101	
EE	238		12-11-9-8-2	1110 1110	
EF	239		12-11-9-8-3	1110 1111	
F0	240		12-11-9-8-4	1111 0000	
F1	241		12-11-9-8-5	1111 0001	
F2	242		12-11-9-8-6	1111 0010	
F3	243		12-11-9-8-7	1111 0011	
F4	244		11-0-9-8-2	1111 0100	
F5	245		11-0-9-8-3	1111 0101	
F6	246		11-0-9-8-4	1111 0110	
F7	247		11-0-9-8-5	1111 0111	
F8	248		11-0-9-8-6	1111 1000	
F9	249		11-0-9-8-7	1111 1001	
FA	250		12-11-0-9-8-2	1111 1010	
FB	251		12-11-0-9-8-3	1111 1011	
FC	252		12-11-0-9-8-4	1111 1100	
FD	253		12-11-0-9-8-5	1111 1101	
FE	254		12-11-0-9-8-6	1111 1110	
FF	255		12-11-0-9-8-7	1111 1111	

APPENDIX B

DIAGNOSTICS AND ERROR STATUS

B.1 Execution-time Diagnostics

RS-Error Number	Cause of Error
RS01	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS02	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS03	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS04	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS05	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS06	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS07	I/O error while reading load module.
RS08	No free MIDL space.
RS09	Insufficient memory.
RS10	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS11	Invalid send buffer address or quantity exceeds 768 bytes.
RS12	Invalid return buffer address.
RS13	Invalid no-wait mode end-action routine address.
RS14	Memory pool unavailable.
RS15	Destination task queue depth exceeded.
RS16	Invalid PSB address.
RS22	Missing file control block (FCB).

RS Error Number	Cause of Error
RS29	Request denied, LFC not allocated.
RS30	Request denied, specified LFC not assigned to a permanent disc file.
RS32	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS33	Error occurred in the routine named in the extended abort code. See the named service for specific reason.
RS38	Request denied, time out occurred while waiting to become lock owner.
RS47	Invalid time interval request.
RS48	Invalid task number.
RS49	Invalid run request.
RS50	Missing parameter.
RS53	Invalid receiver.
RS60	Invalid address specified.
RS65	Invalid delete request.
RS66	Invalid abort request.
RS67	Invalid resource mark request.
RS68	Taskname/tasknumber not found.
RS69	File control block (FCB) not located.
RS70	Allocation error (appears only if IOSTAT and $n parameters have been omitted).
RS90	Request denied, file lock allocated or exclusively locked.
RS99	An attempt was made to mix calls between SRTL libraries.

RT Error Number	Cause of Error
RT01	Unformatted read I/O error.
RT02	Formatted read I/O error.
RT03	Unformatted write I/O error.

RT Error Number	Cause of Error
RT04	Formatted write I/O error.
RT05	Reference made to nonexistent device type or address.
RT06	Unit out of range (0-999).
RT07	No left parenthesis in format.
RT08	Transfer index out of range (option 7 or M:ERRFLG can be used to avoid an abort.
RT09	Format error.
RT10	The I/O transfer requirements for data buffer are incompatible with amount of available data.
RT11	Format parenthesis level in excess of two.
RT12	Invalid descriptor in format table.
RT13	Argument list exceeds logical read record.
RT14	Incorrect descriptor in format.
RT15	Integer descriptor, but non-integer argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT16	Hexadecimal descriptor, but non-hexadecimal argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT17	D,E,F,G, descriptor, not real or complex argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT18	Logical descriptor, but non-logical argument (option 7 or M:ERRFLG can be used to avoid an abort).
RT19	Attempt to read past EOF/EOM.
RT20	Attempt to write past EOF/EOM.
RT21	Attempt to read past EOF/EOM.
RT22	Attempt to write past EOF/EOM.
RT23	Attempt to backspace following EOF/EOM.
RT24	Rewind after EOF/EOM.
RT25	Formatted record read.
RT26	Unformatted record read.

RT Error Number	Cause of Error
RT27	Doubleword integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT28	Byte integer input with negative sign (option 7 or M:ERRFLG can be used to avoid an abort).
RT29	Byte integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT30	Halfword integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT31	Fullword integer overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT32	Illegal character in D,E,F,G, input (option 7 or M:ERRFLG can be used to avoid an abort).
RT33	Underflow in floating conversion (option 7 or M:ERRFLG can be used to avoid an abort).
RT34	Overflow in floating conversion (option 7 or M:ERRFLG can be used to avoid an abort).
RT35	Argument list overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT36	Argument list overflow (option 7 or M:ERRFLG can be used to avoid an abort).
RT37	Not enough arguments were passed.
RT40	Attempt to free busy IOCH/IOCB entry.
RT41	Attempt to link busy IOCH/IOCB entry.
RT42	IOCH/IOCB table overflow.
RT43	ADI wait I/O returned before I/O termination.
RT44	Status parameter not linked to ADI device prior to I/O request.
RT46	ADI table address not on halfword boundary.
RT50	Missing parameter.
RT51	Parameter out of range.
RT52	End of search list reached.
RT53	No unit connection.

RT Error Number	Cause of Error
RT55	Error found in math library routine.
RT60	Illegal random access.
RT61	List-directed I/O (input) encountered, character string split between two records.
RT62	Internal file read/write past EOF/EOM with no END option specified.
RT63	Block number exceeds maximum block number in file.
RT64	Record overflow.
RT65	Record length exceeds maximum allowable.
RT66	Record length not specified for random access or specified for sequential file.
RT67	Implicit open not allowed for random access I/O.
RT68	Reference to sequential operation on a file opened for direct access.
RT69	Error(s) encountered on open.
RT70	File must be unblocked and opened for random access for BUFFERIN/BUFFEROUT random I/O.
RT74	Attempt to delete a file that does not exist or does not have delete access.
RT80	Subscript error (i.e., subscript not a decimal number, illegal punctuation, excessive subscripts, or subscript out of range.)
RT81	NAMELIST identifier error (i.e., column 1 non-blank, ampersand character not present, name does not immediately follow ampersand character, or non-blank following name).
RT82	Symbolic name error (no equal sign after variable/array name).
RT83	Data item error (i.e., excessive values for symbol or expected to find symbol).
RT84	Illegal value (i.e., illegal punctuation, missing comma, zero Hollerith count, or illegal character in value).
RT85	Attempt to read past EOF/EOM.
RT86	Attempt to write past EOF/EOM.
RT87	Symbolic name not defined in NAMELIST statement.

RT Error Number	Cause of Error
RT88	Repeat count error.
RT89	Symbol name exceeds eight characters.
RT90	Invalid read/write operation.
RT91	End-of-file status return pursuant to random access record.
RT92	Random access partition number out of range (i.e., partition number not between 1 and 95, inclusive).
RT93	Random access record number out of range (i.e., record number not between 1 - 65,535, inclusive).
RT94	Random access transfer length (write/read) or record size definition (define) out-of-range (i.e., transfer record length not between 1 and 65,535 bytes inclusive).
RT95	Invalid random access argument list length.
RT96	FCB table overflow (31) or maximum number of entries allowed in the allocation table (30) has been exceeded.
RT97	Diagnostic output messages exceed 100 lines. To allow more diagnostic messages, statically assign the "DO" file (e.g., $ASSIGN2 DO=SLO,500).
RT98	Denial return when attempting to allocate file for diagnostic output message.
RT99	Insufficient blocking buffer space. (Each unit assignment to a system file requires one blocking buffer unless one file is assigned to another, i.e., $AS LFC TO LFC).

The RT prefix can be replaced by a W. or T., which indicates whether the error is warning or terminal, respectively.

The following errors are considered minor in the sense that either option 7 or M:ERRFLG may be used to avoid an abort.

RT08

RT15

RT16

RT17

RT18

RT27

RT28

RT29

RT30

RT31

RT32

RT33

RT34

RT35

RT36

B.2 Error Status Values for X_ Subroutines

The following istatus values may apply to these subroutines:

X_DISMNT	X_EXCL	X_INCXDP
X_MOUNT	X_INCLD	
X_INQ	X_DPXMNT	

0	Request accepted.
1	Resource does not exist (invalid pathname or memory partition definition).
2	Specified access mode not allowed.
3	File Printer Table/File Assignment Table (FPT/FAT) space not available.
4	Blocking buffer space not available.
5	Shared Memory Table (SMT) entry not found for partition.
6	Volume Assignment Table (VAT) space not available.
7	Static assignment to dynamic common.
8	Unrecoverable input/output error to volume.
9	Invalid usage specification.
10	Dynamic partition definition exceeds memory limitations.
11	Invalid Resource Requirement Summary (RRS) entry.
12	Logical file code logically equated to an unassigned LFC.

13 Assigned device not in system.
14 Resource already allocated by requesting task.
15 System General Output (SGO) or System Control (SYC) assignment not allowed by real-time task.
16 Common memory conflicts with task's address space.
17 Duplicate LFC assignment attempted.
18 Invalid device specification.
19 Invalid resource id.
20 Volume not assigned to this task or volume is public.
21 J.MOUNT run request failed.
22 Resource marked for deletion.
23 Assigned device is marked off-line.
24 Segment definition allocation by unprivileged task.
25 Random access not allowed for this access mode.
26 User attempting to open System Control (SYC) file in a write mode.
27 Resource already opened by this task in different access mode.
28 Invalid access specification at open.
29 Logical file code not assigned.
30 Invalid allocation index.
31 Close request issued for unopened resource.
32 Attempt to release exclusive resource lock not owned by this task or a synchronous lock not set.
33 Attempt to release exclusive resource lock on resource allocated for exclusive use.
35 Attempt to exclude memory partition not mapped into requesting task's address space.
36 Attempt to include dynamic partition in memory-only environment.
37 Invalid J.MOUNT request.
38 Timeout occurred while waiting for resource to become available.
50 Resource is exclusively locked by another task.
51 Shareable resource allocated by another task in incompatible access mode.
52 Volume space not available.
53 Assigned device not available.
54 Unable to allocate resource for specified use.
55 Allocated Resource Table (ART) space is not available.
57 Volume not available for mount with requested use.
58 Shared Memory Table (SMT) space not available.
59 Mounted Volume Table (MVT) space not available.

The following istatus values may apply to these subroutines:

X_CREDIR	X_MDESC	X_RNAME
X_DDIR	X_PERM	X_TRUNC
X_DIRECT	X_RDESC	X_WDESC
X_EXTEND	X_RECON	X_CPART
X_PROJCT	X_REPLC	X_DPART
X_LOG	X_RID	

0 Operation successful.
1 Pathname invalid.
2 Pathname consists of volume only.
3 Volume not mounted.
4 Directory does not exist.
5 Directory name in use.
6 Directory creation not allowed at specified level.

7	Resource does not exist.
8	Resource name in use.
9	Resource descriptor unavailable.
10	Directory entry unavailable.
11	Required file space unavailable.
12	Unrecoverable input/output error while reading DMAP.
13	Unrecoverable input/output error while writing DMAP.
14	Unrecoverable input/output error while reading resource descriptor.
15	Unrecoverable input/output error while writing resource descriptor.
16	Unrecoverable input/output error while reading SMAP.
17	Unrecoverable input/output error while writing SMAP.
18	Unrecoverable input/output error while reading directory.
19	Unrecoverable input/output error while writing directory.
20	Projectgroup name or key invalid.
21	Reserved.
22	File Control Block (FCB) destroyed.
23	Parameter address error.
24	Resource descriptor not currently allocated.
25	Pathname block overflow.
26	File space not currently allocated.
27	Change defaults not allowed.
28	Resource cannot be accessed in required mode.
29	Operation not allowed on this resource type.
30	Required parameter not specified.
31	File extension denied; segment definition area full.
32	File extension denied; file would exceed maximum size allowed.
33	Input/output occurred when resource was zeroed.
34	Replacement file cannot be allocated.
35	Invalid directory entry.
36	Directory and file not on same volume.
37	Reserved.
38	Replacement file not exclusively allocated to you.
39	Out of system space.
40	Cannot allocate File Assignment Table/File Pointer Table (FAT/FPT) when creating a temporary file.
41	Deallocate error in zeroing file.
42	Resource descriptor destroyed.
43	Invalid resource specification.
44	Error from Resource Management Module (H.REMM).
45	Attempt to modify more than one resource descriptor at the same time; or an attempt to rewrite before modifying resource descriptor.
46	Resource descriptor is locked by another CPU (dual port only).
47	Directory contains active entries and cannot be deleted.
48	A resource descriptor's link count is zero.
49	Attempting to delete a permanent resource without specifying a pathname or pathname block vector.
50	Resource descriptor contains unexpected resource descriptor type.

Gould S.E.L. Users Group . . .

The purpose of the Gould S.E.L. Users Group is to help create better User/User and User/Gould S.E.L. Communications.

There is no fee to join the Users Group. Simply complete the Membership Application on the reverse side and mail to the Users Group Administrator. You will automatically receive Users Group Newsletters, Referral Guide and other pertinent Users Group Activity information.

Fold and Staple for Mailing

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 947 FT. LAUDERDALE, FLORIDA 33310

POSTAGE WILL BE PAID BY ADDRESSEE

Gould Inc., S.E.L. Computer Systems Division
Attn: Users Group Administrator
6901 W. Sunrise Blvd., P.O. Box 9148
Ft. Lauderdale, FL 33310-9148

Fold and Staple for Mailing

GOULD
Electronics

Gould Inc., S.E.L. Computer Systems Division
6901 W. Sunrise Blvd., P.O. Box 9148,
Fort Lauderdale, FL 33310-9148
Phone (305) 587-2900

GOULD
Electronics

Users Group Membership Application

USER ORGANIZATION: _____

REPRESENTATIVE(S): _____

_____

_____

ADDRESS: _____

_____

TELEX NUMBER: _____ PHONE NUMBER: _____

NUMBER AND TYPE OF GOULD S.E.L. COMPUTERS: _____

_____

_____

_____

APPLICATIONS (Please Indicate)

1. EDP
   - A. Inventory Control
   - B. Engineering & Production Data Control
   - C. Large Machine Off-Load
   - D. Remote Batch Terminal
   - E. Other

2. Communications
   - A. Telephone System Monitoring
   - B. Front End Processors
   - C. Message Switching
   - D. Other

3. Design & Drafting
   - A. Electrical
   - B. Mechanical
   - C. Architectural
   - D. Cartography
   - E. Image Processing
   - F. Other

4. Industrial Automation
   - A. Continuous Process Control Op.
   - B. Production Scheduling & Control
   - C. Process Planning
   - D. Numerical Control
   - E. Other

5. Laboratory and Computational
   - A. Seismic
   - B. Scientific Calculation
   - C. Experiment Monitoring
   - D. Mathematical Modeling
   - E. Signal Processing
   - F. Other

6. Energy Monitoring & Control
   - A. Power Generation
   - B. Power Distribution
   - C. Environmental Control
   - D. Meter Monitoring
   - E. Other

7. Simulation
   - A. Flight Simulators
   - B. Power Plant Simulators
   - C. Electronic Warfare
   - D. Other

8. Other

Please return to:

Users Group Administrator

Date: _____