IBM

Series/1

GC34-0138-0

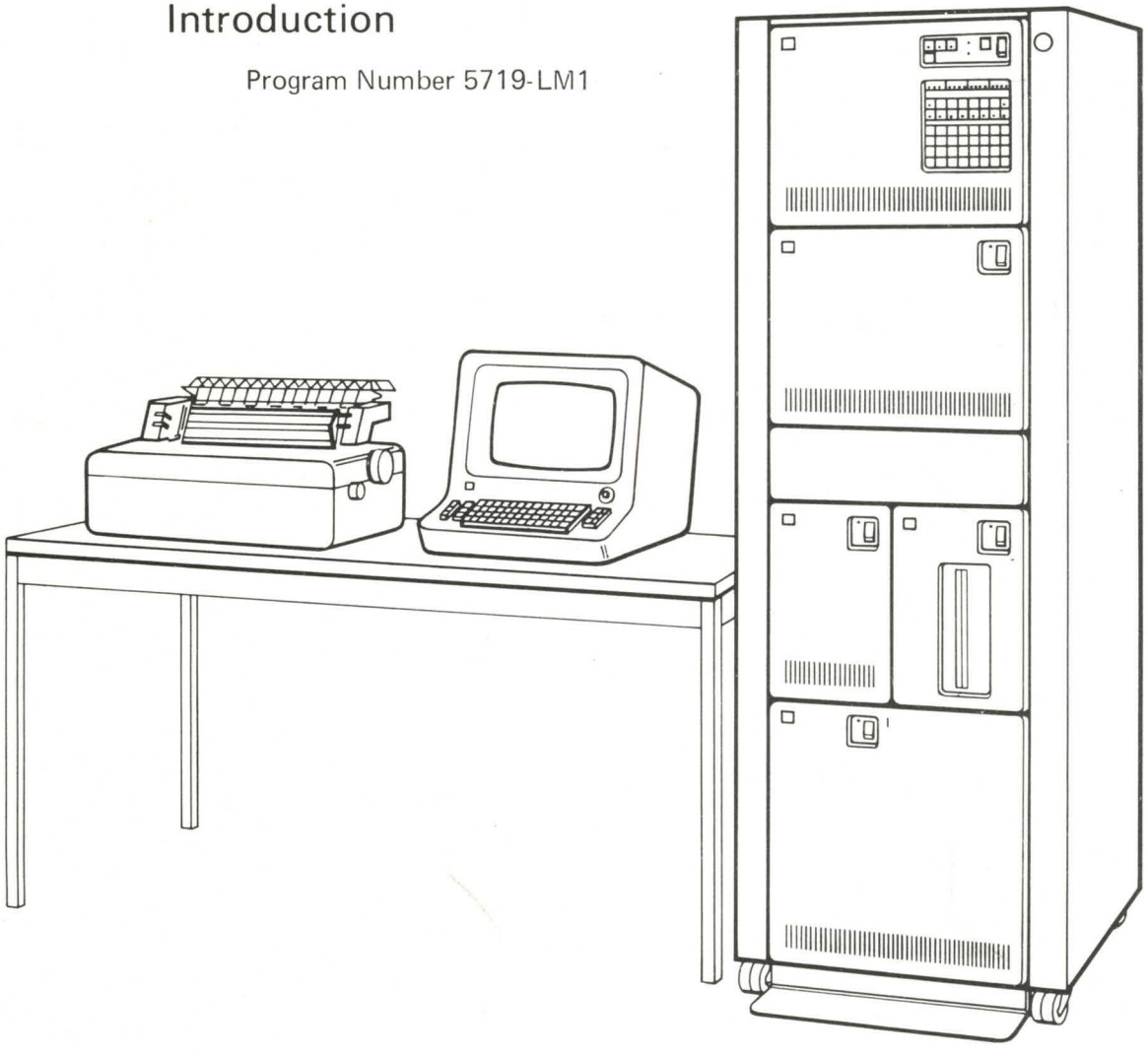
S1-20

PROGRAM
PRODUCT

MFSL INTRODUCTION

IBM Series/1
Mathematical and Functional Subroutine Library
Introduction

Program Number 5719-LM1



IBM

Series/1

GC34-0138-0

S1-20

PROGRAM
PRODUCT

MFSL INTRODUCTION

IBM Series/1
Mathematical and Functional Subroutine Library
Introduction

Program Number 5719-LM1

This publication is for planning purposes only. The information herein is subject to change before the products described become available.

First Edition (February 1977)

This edition applies to the IBM Series/1 Mathematical and Functional Subroutine Library (MFSL) program product (Program Number 5719-LM1). MFSL is compatible with the following program products:

IBM Series/1 FORTRAN IV, Program Number 5719-FO1.
IBM Series/1 FORTRAN IV, Realtime Subroutine Library, Program Number 5719-FO3.
IBM Series/1 Program Preparation Subsystem, Program Number 5719-AS1.
IBM Series/1 Realtime Programming System, Program Number 5719-PC1.

Significant changes or additions to the contents of this publication will be reported in subsequent revisions or Technical Newsletters. Requests for copies of IBM publications should be made to your IBM representative or the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, send your comments to IBM Corporation, Systems Publications, Department 27T, P. O. Box 1328, Boca Raton, Florida 33432. Comments become the property of IBM.

Contents

Preface

This is an introductory manual that describes the Mathematical and Functional Subroutine Library (MFSL) for the IBM Series/1. MFSL comprises mathematical, EBCDIC conversion, and error-checking subroutines for both FORTRAN IV and assembler language users. Library service routines are included for assembler language users only. The topics covered include a general description of MFSL and a description of using MFSL with the application system.

The reader should be familiar with the basic coding techniques of FORTRAN IV or assembler programming; the related language publications are listed below. The section "MFSL and the Application System" is related to the IBM Series/1 Program Preparation Subsystem program product and the IBM Series/1 Realtime Programming System program product. The reader may find it useful to refer to the introductory publications listed below for these program products.

This publication describes the basic mathematical and functional capability of the MFSL program product. Related subroutine support for executive functions and realtime process input/output is provided by the FORTRAN IV Realtime Subroutine Library program product. For information on these subroutines, see the related FORTRAN IV publication listed below.

Related Publications

IBM Series/1 FORTRAN IV: Introduction, GC34-0132.
IBM Series/1 FORTRAN IV: Language Reference, GC34-0133.
IBM Series/1 Program Preparation Subsystem: Macro Assembler User's Guide, SC34-0124.
IBM Series/1 Program Preparation Subsystem: Introduction, GC34-0121.
IBM Series/1 Realtime Programming System: Introduction and Planning Guide, GC34-0102.
IBM Series/1 System Summary, GC34-0035.

The Mathematical and Functional Subroutine Library

The Mathematical and Functional Subroutine Library (MFSL) is a set of subroutines that aids in developing application programs. The MFSL subroutines can be used with the IBM Series/1 FORTRAN IV program product (hereafter referred to as FORTRAN IV) or the macro assembler language provided with the IBM Series/1 Program Preparation Subsystem program product (hereafter referred to as Program Preparation Subsystem). The operating environment required for MFSL, FORTRAN IV, and the assembler is provided by the IBM Series/1 Realtime Programming System program product (hereafter referred to as Realtime Programming System). A user-written operating system that provides the required interfaces can also use the MFSL subroutines.

MFSL is compatible with any Series/1 hardware configuration that includes the primary and secondary storage required for the MFSL subroutines used. The configuration requires floating-point support only if the user application requires REAL or floating-point arithmetic. MFSL functions that operate on integer or fixed-point variables have no internal requirements for floating-point support.

General Description

In the course of developing application programs, many mathematical and other functions are performed over and over again. These functions are often difficult and tedious to implement. For this reason, a set of mathematical and functional subroutines that perform the functions required can save time and effort in developing applications. The functions provided by MFSL are as follows:

Mathematical Subroutines

Arc Tangent, one or two arguments:
Given one argument, an arc tangent subroutine returns the angle that has the argument as its tangent. Given two arguments, an arc tangent subroutine returns the angle that has the quotient of the two arguments as its tangent. All angles are in radians.

Cosine:
A sine-cosine subroutine returns the value of the cosine of the argument. All angles are in radians.

Doubleword Divide:
A division subroutine returns the doubleword value of a doubleword number divided by a doubleword number.

Doubleword Multiply:
A multiplication subroutine returns the doubleword value of a doubleword number multiplied by a doubleword number.

Exponential function:
An exponential subroutine returns the value of e raised to the power of the argument.

Exponentiation:
An exponentiation subroutine returns the value of any base raised to any power.

Hyperbolic tangent:
A hyperbolic tangent subroutine returns the value of the hyperbolic tangent for the argument.

Logarithms, common or natural:
Logarithmic subroutines return the value of the base 10 or base e logarithm of the argument.

Maximum value:
A maximum value subroutine returns the value of the largest argument in a set of arguments.
Minimum value:
A minimum value subroutine returns the value of the smallest argument in a set of arguments.
Modular arithmetic:
A modular arithmetic subroutine returns the remainder from the division of two arguments.
Positive difference:
A positive difference subroutine returns the positive difference between the first argument and the smaller of two arguments.
Sine:
A sine-cosine subroutine returns the value of the sine of the argument. All angles are in radians.
Square root:
A square root subroutine returns the value of the square root of the argument.
Transfer of sign:
A transfer-of-sign subroutine returns the value of the sign of the second argument concatenated to the absolute value of the first argument.

Conversion Subroutines

EBCDIC to floating-point:
A conversion subroutine converts an EBCDIC input into a floating-point number.
EBCDIC to integer:
A conversion subroutine converts an EBCDIC input into an integer value.
Floating-point to EBCDIC:
A conversion subroutine converts a floating-point number into an EBCDIC output.
Integer to EBCDIC:
A conversion subroutine converts an integer value into an EBCDIC output.

Error-Checking Subroutines

Function test:
A function test subroutine determines if an error was detected in any MFSL logarithmic, trigonometric, exponentiation, square root, or conversion subroutine since the last call to function test.
Floating-point divide exception:
A floating-point divide exception subroutine determines if an error in a floating-point divide operation has occurred since the last call to floating-point divide exception.
Floating-point overflow/underflow:
A floating-point overflow/underflow subroutine determines if an overflow or underflow condition has occurred since the last call to floating-point overflow/underflow.

Service Subroutines (Assembler Applications Only)

Applications that consist entirely of assembler-coded programs require these services. FORTRAN IV programs, or assembler programs that run under FORTRAN IV main programs, should not call these subroutines. The FORTRAN IV compiler generates the necessary interfaces without any user action.

Library work area initialization/termination:
An initialization or termination subroutine creates or deletes a work area in storage for user communication (error flagging).

Abnormal termination routine specification:
An abnormal termination routine specification subroutine allows the user to specify a routine to receive control if a program interruption occurs or if an abnormal termination macro instruction (STOPTASK) is issued.

Subroutine Library Contents

MFSL contains four types of subroutines: (1) mathematical functions such as SIN and SQRT, (2) EBCDIC conversion subroutines such as $FCIN (EBCDIC-to-floating-point), (3) error-checking subroutines such as FCTST (function test), and (4) subroutine library services such as $FMYINT (work area initialization).

The library subroutines can be used in either a FORTRAN IV or assembler language program. In FORTRAN IV, calls to the library subroutines are either at the programmer's request through explicit references to subroutine names or in response to the FORTRAN IV exponentiation notation. In assembler language, all MFSL subroutines are invoked through explicit calls to subroutine names. The MFSL subroutines are added to the application task set load module by the application builder.

Mathematical Subroutines

The MFSL mathematical subroutines perform many commonly used mathematical operations to aid the application programmer. The explicitly called mathematical subroutines include logarithmic and exponential functions, trigonometric functions, and the miscellaneous functions of maximum and minimum values, modular arithmetic, positive difference, and transfer of sign. These subroutines are described in Figure 1 through Figure 5. The implicitly called mathematical subroutines provide a general exponentiation capability and an INTEGER*4 (doubleword) multiply and divide capability. These subroutines are described in Figure 6.

General function	*Specific function*	*Entry name(s)*
Logarithmic and exponential subroutines (described in Figure 2)	Exponential	EXP DEXP
	Logarithmic, common and natural	ALOG, ALOG10 DLOG, DLOG10
	Square root	SQRT, DSQRT
Trigonometric subroutines (described in Figure 3)	Arc tangent	ATAN, ATAN2 DATAN, DATAN2
	Cosine and sine	COS, DCOS SIN, DSIN
Hyperbolic function subroutine (described in Figure 4)	Hyperbolic tangent	TANH, DTANH
Miscellaneous subroutines (described in Figure 5)	Maximum and minimum value	MAX0, MIN0 MAX0#, MIN0# AMAX0, AMIN0 AMAX0#, AMIN0# MAX1, MIN1 MAX1#, MIN1# AMAX1, AMIN1 DMAX1, DMIN1
	Modular arithmetic	MOD AMOD, DMOD
	Positive difference	DIM IDIM, IDIM#
	Transfer of sign	SIGN, DSIGN ISIGN, ISIGN#

Figure 1. Explicitly called MFSL subroutines

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1] *and range*[2]
			No.	*Type*[1]	*Range*	
Common and natural logarithm	ALOG	$y=\log_e x$ or $y=\ln x$	1	REAL*4	$x>0$	REAL*4 $y\geq -180.218$ $y\leq 174.673$
	ALOG10	$y=\log_{10}x$	1	REAL*4	$x>0$	REAL*4 $y\geq -78.268$ $y\leq 75.859$
	DLOG	$y=\log_e x$ or $y=\ln x$	1	REAL*8	$x>0$	REAL*8 $y\geq -180.218$ $y\leq 174.673$
	DLOG10	$y=\log_{10}x$	1	REAL*8	$x>0$	REAL*8 $y\geq -78.268$ $y\leq 75.859$
Exponential	EXP	$y=e^x$	1	REAL*4	$x\geq -180.218$ $x\leq 174.673$	REAL*4 $0\leq y\leq \gamma$
	DEXP	$y=e^x$	1	REAL*8	$x\geq -180.218$ $x\leq 174.673$	REAL*8 $0\leq y\leq \gamma$
Square root	SQRT	$y=x^{1/2}$	1	REAL*4	$x\geq 0$	REAL*4 $0\leq y\leq \gamma^{1/2}$
	DSQRT	$y=x^{1/2}$	1	REAL*8	$x\geq 0$	REAL*8 $0\leq y\leq \gamma^{1/2}$

Notes.

[1] Assembler programming: REAL*4 and REAL*8 FORTRAN IV arguments correspond to doubleword and two-doubleword floating-point arguments, respectively.

[2] γ is approximately 7.24×10^{75}; $\gamma^{1/2}$ is approximately 8.51×10^{37}.

Figure 2. Logarithmic and exponential MFSL subroutines

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1] *and range*
			No.	*Type*[1]	*Range*[2]	
Arc tangent	ATAN	y=arctan x	1	REAL*4	Any REAL argument	REAL*4 (in radians) $-\pi/2 \leq y \leq \pi/2$
	ATAN2	y=arctan x_1/x_2	2	REAL*4	Any REAL arguments (except 0/0)	REAL*4 (in radians) $-\pi < y \leq \pi$
	DATAN	y=arctan x	1	REAL*8	Any REAL argument	REAL*8 (in radians) $-\pi/2 \leq y \leq \pi/2$
	DATAN2	y=arctan x_1/x_2	2	REAL*8	Any REAL arguments (except 0/0)	$-\pi < y \leq \pi$
Sine and cosine	SIN	y=sin x	1	REAL*4 (in radians)	$\|x\| < 2^{18}\pi$	REAL*4 $-1 \leq y \leq 1$
	COS	y=cos x	1	REAL*4 (in radians)	$\|x\| < 2^{18}\pi$	REAL*4 $-1 \leq y \leq 1$
	DSIN	y=sin x	1	REAL*8 (in radians)	$\|x\| < 2^{50}\pi$	REAL*8 $-1 \leq y \leq 1$
	DCOS	y=cos x	1	REAL*8 (in radians)	$\|x\| < 2^{50}\pi$	REAL*8 $-1 \leq y \leq 1$

Notes.

[1] Assembler programming: REAL*4 and REAL*8 FORTRAN IV arguments correspond to doubleword and two-doubleword floating-point arguments, respectively.

[2] The following are approximate values:
$2^{18}\pi = 2.62 \times 10^5\pi$
$2^{50}\pi = 1.13 \times 10^{15}\pi$

Figure 3. Trigonometric MFSL subroutines

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1] *and range*
			No.	*Type*[1]	*Range*	
Hyperbolic Tangent	TANH	$y=(e^x-e^{-x})/(e^x+e^{-x})$	1	REAL*4	Any REAL argument	REAL*4 $-1 \leq y \leq 1$
	DTANH	$y=(e^x-e^{-x})/(e^x+e^{-x})$	1	REAL*8	Any REAL argument	REAL*8 $-1 \leq y \leq 1$

Notes.

[1] Assembler programming: REAL*4 and REAL*8 FORTRAN IV arguments correspond to doubleword and two-doubleword floating-point arguments, respectively.

Figure 4. Hyperbolic function MFSL subroutines

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1]
			No.	*Type*[1]	*Range*	
Maximum and minimum values	MAX0	$y=\max(x_1,...,x_n)$	≥2	INTEGER*2 or INTEGER*4 (See Note 2)	Any INTEGER arguments	INTEGER*2 or INTEGER*4 (See Note 2)
	MAX0# (See Note 3)	$y=\max(x_1,...,x_n)$	≥2	INTEGER*2	Any INTEGER arguments	INTEGER*2
	MIN0	$y=\min(x_1,...,x_n)$	≥2	INTEGER*2 or INTEGER*4 (See Note 2)	Any INTEGER arguments	INTEGER*2 or INTEGER*4 (See Note 2)
	MIN0# (See Note 3)	$y=\min(x_1,...,x_n)$	≥2	INTEGER*2	Any INTEGER arguments	INTEGER*2
	AMAX0	$y=\max(x_1,...,x_n)$	≥2	INTEGER*2 or INTEGER*4 (See Note 2)	Any INTEGER arguments	REAL*4
	AMAX0# (See Note 3)	$y=\max(x_1,...,x_n)$	≥2	INTEGER*2	Any INTEGER arguments	REAL*4
	AMIN0	$y=\min(x_1,...,x_n)$	≥2	INTEGER*2 or INTEGER*4 (See Note 2)	Any INTEGER arguments	REAL*4
	AMIN0# (See Note 3)	$y=\min(x_1,...,x_n)$	≥2	INTEGER*2	Any INTEGER arguments	REAL*4

Notes.

(See end of Figure 5.)

Figure 5. Miscellaneous mathematical subroutines (Part 1 of 3)

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1]
			No.	*Type*[1]	*Range*	
Maximum and minimum values (cont.)	MAX1	y=max (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	INTEGER*2 or INTEGER*4 (See Note 2)
	MAX1# (See Note 3)	y=max (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	INTEGER*2
	MIN1	y=min (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	INTEGER*2 or INTEGER*4 (See Note 2)
	MIN1# (See Note 3)	y=min (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	INTEGER*2
	AMAX1	y=max (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	REAL*4
	AMIN1	y=min (x_1,...,x_n)	≥2	REAL*4	Any REAL arguments	REAL*4
	DMAX1	y=max (x_1,...,x_n)	≥2	REAL*8	Any REAL arguments	REAL*8
	DMIN1	y=min (x_1,...,x_n)	≥2	REAL*8	Any REAL arguments	REAL*8

Notes.

(See end of Figure 5.)

Figure 5. Miscellaneous mathematical subroutines (Part 2 of 3)

General function	*Function or entry name*	*Definition*	*Arguments*			*Function value type*[1]
			No.	*Type*[1]	*Range*	
Modular arithmetic	MOD	$y=x_1$ modulo x_2 (See Note 4)	2	INTEGER*2 or INTEGER*4 (See Note 2)	$x_2 \neq 0$ (See Note 5)	INTEGER*2 or INTEGER*4
	MOD# (See Note 3)	$y=x_1$ modulo x_2 (See Note 4)	2	INTEGER*2	$x_2 \neq 0$ (See Note 5)	INTEGER*2
	AMOD	$y=x_1$ modulo x_2 (See Note 4)	2	REAL*4	$x_2 \neq 0$ (See Note 5)	REAL*4
	DMOD	$y=x_1$ modulo x_2 (See Note 4)	2	REAL*8	$x_2 \neq 0$ (See Note 5)	REAL*8
Positive difference	DIM	$y=x_1-\min(x_1,x_2)$	2	REAL*4	Any REAL arguments	REAL*4
	IDIM	$y=x_1-\min(x_1,x_2)$	2	INTEGER*2 or INTEGER*4 (See Note 2)	Any INTEGER arguments	INTEGER*2 or INTEGER*4 (See Note 2)
	IDIM# (See Note 3)	$y=x_1-\min(x_1,x_2)$	2	INTEGER*2	Any INTEGER arguments	INTEGER*2
Transfer of sign	SIGN	$y=\text{sgn}(x_2)\lvert x_1\rvert$ (See Note 6)	2	REAL*4	$x_2 \neq 0$ (See Note 6)	REAL*4
	DSIGN	$y=\text{sgn}(x_2)\lvert x_1\rvert$	2	REAL*8	$x_2 \neq 0$ (See Note 6)	REAL*8
	ISIGN	$y=\text{sgn}(x_2)\lvert x_1\rvert$	2	INTEGER*2 or INTEGER*4 (See Note 2)	$x_2 \neq 0$ (See Note 6)	INTEGER*2 or INTEGER*4
	ISIGN# (See Note 3)	$y=\text{sgn}(x_2)\lvert x_1\rvert$	2	INTEGER*2	$x_2 \neq 0$ (See Note 6)	INTEGER*2

Notes.

1. Assembler programming: REAL*4 and REAL*8 FORTRAN IV arguments correspond to doubleword and two-doubleword floating-point arguments, respectively.
2. In FORTRAN IV, argument and value must be the same type. In assembler language, this entry name is for doublewords only.
3. These entry names are for assembler language programs only.
4. The expression x_1 modulo x_2 is defined as $x_1 - x_2 [x_1/x_2]$ where the brackets indicate that an integer is used. The largest integer whose magnitude does not exceed the value of x_1/x_2 is used. The sign of the integer is the same as the sign of x_1/x_2.
5. If $x_2 = 0$, the modular function is undefined. In this case, a divide exception is recognized, and an interruption occurs.
6. The sgn function is defined as follows: $\text{sgn}(x_2) = 1$ if $x_2 > 0$, and $\text{sgn}(X_2) = -1$ if $x_2 < 0$. If $x_2 = 0$, sgn(x) is not defined. In this case, the transfer-of-sign function does not indicate an error, but its results are unpredictable.

Figure 5. Miscellaneous mathematical subroutines (Part 3 of 3)

General function	*Entry*[1] *name*	*Implicit*[2] *function reference*	*Arguments*		*Function value type*
			No.	*Type*[3]	
Raise an integer to an integer power	IEXP#	m=i**j	2	i=INTEGER*2 j=INTEGER*2	m=INTEGER*2
	IEXP	m=i**j	2	i=INTEGER*4 j=INTEGER*4	m=INTEGER*4
Raise a real number to an integer power	EXPI#	y=x**k	2	x=REAL*4 k=INTEGER*2	y=REAL*4
	EXPI	y=x**k	2	x=REAL*4 k=INTEGER*4	y=REAL*4
	DEXPI#	y=x**k	2	x=REAL*8 k=INTEGER*2	y=REAL*8
	DEXPI	y=x**k	2	x=REAL*8 k=INTEGER*4	y=REAL*8
Raise a real number to a real power	EXPE	y=x**z	2	x=REAL*4 z=REAL*4	y=REAL*4
	DEXPD	y=x**z	2	x=REAL*8 z=REAL*8	y=REAL*8
Divide a doubleword integer by a doubleword integer	IDIV	k=l/m	2	l=INTEGER*4 m=INTEGER*4	k=INTEGER*4
Multiply a doubleword integer by a doubleword integer	IMULT	k=l*m	2	l=INTEGER*4 m=INTEGER*4	k=INTEGER*4

Notes.

[1] Entry names are used in assembler language programs only.

[2] This is only a representation of a FORTRAN IV statement; it is not the only way the subroutine can be called.

[3] Assembler programming: REAL*4 and REAL*8 FORTRAN IV arguments correspond to doubleword and two-doubleword floating-point arguments, respectively.

Figure 6. Exponentiation and INTEGER*4 (doubleword) multiplication and division subroutines

Conversion Subroutines

Input and output data conversions are made easier by the MFSL EBCDIC conversion subroutines. Numerical input data in EBCDIC format can be converted to an internal representation in integer or floating-point format. After computations in integer or floating-point arithmetic, the resulting numerical output can be converted back to an EBCDIC format. These subroutines are described in Figure 7.

To use the conversion subroutines, the user program must establish input and output buffers and manage their contents by using the READ/WRITE facilities of FORTRAN IV or the macro assembler language. The MFSL conversion subroutines always move data between a variable in the user program and an input or output buffer. Each conversion subroutine manages its buffer so that repeated calls to the subroutine will process sequential fields in the buffer. When the buffer is completely processed, the conversion subroutine goes back to the beginning of the buffer for the next conversion following a READ or WRITE

To convert from	*To*	*Call entry name*
EBCDIC (with or without exponent)	Floating-point single precision	$ECIN
	Floating-point double precision	$DCIN
EBCDIC (no exponent)	Fullword integer	$I2CIN
	Doubleword integer	$I4CIN
Floating-point single precision	EBCDIC (with exponent)	$ECOT
	EBCDIC (no exponent)	$FCOT
Floating-point double precision	EBCDIC (with exponent)	$DCOT
	EBCDIC (no exponent)	$FCOTD
Fullword integer	EBCDIC (no exponent)	$I2COT
Doubleword integer	EBCDIC (no exponent)	$I4COT

Figure 7. EBCDIC conversions and MFSL subroutines

I/O operation. The user call to a conversion subroutine requires a parameter list to specify the following:

- The name of the variable used in the user program
- The width (in characters) of the input or output buffer field for each conversion
- The name of the input or output buffer
- The value of a decimal scale factor (if used)
- The default number of decimal places (if used)

Error-checking Subroutines

The MFSL subroutines communicate with the user through flags in the MFSL library work area. There are no MFSL error messages. The typical MFSL procedure for error handling is that the error-detecting subroutine (for example, SQRT detecting a negative argument) sets a flag in the library work area and then continues processing according to a predefined rule (such as taking the square root of the absolute value). To check for errors, the user must either check function arguments before invoking a subroutine or use the error-checking subroutines to validate the results. These subroutines are described in Figure 8.

To check	*Using as interface variables*	*Call assembler name*	*Call FORTRAN IV name*
Logarithmic, trigonometric, exponential, square root, conversion subroutines	Fullwords	FCTST#	FCTST
	Doublewords	FCTST	FCTST
Floating-point divide by zero	Fullwords	DVCHK#	DVCHK
	Doublewords	DVCHK	DVCHK
Floating-point overflow or underflow	Fullwords	OVERFL#	OVERFL
	Doublewords	OVERFL	OVERFL

Figure 8. Error conditions and MFSL error-checking subroutines

In using the error-checking subroutines, the user does not directly access the library work area. The subroutine call requires a parameter list that names an interface variable that is used in the user program. The error-checking subroutine accesses the library work area and sets the interface variable to indicate the error status to the user program. On return from the error-checking subroutine, the user program must test the interface variable to determine whether an error was detected. Each call to an error-checking subroutine resets the associated error indicator bits in the library work area. The values returned by the error-checking subroutines are shown in Figure 9.

Error-checking subroutine	*Returned value*	*Error status indicated*
DVCHK or DVCHK#	1	Division by zero occurred
	2	No division by zero occurred
FCTST or FCTST# (See note below)	1	Function error occurred
	2	No function error occurred
OVERFL or OVERFL#	1	Overflow occurred
	2	No overflow or underflow occurred
	3	Underflow occurred
Note. Two interface variables are required for function test. If the first indicates that an error occurred, the second can be tested to determine the specific error.		

Figure 9. Returned values from error-checking subroutines

Service Subroutines (Called by Assembler Language Programs Only)

The MFSL subroutines require an operating environment that includes the library work area and an interruption-handling facility. This environment is established by the FORTRAN IV compiler for FORTRAN IV programs or a set of programs whose main program is in FORTRAN IV. Assembler language programs that do not execute in a FORTRAN IV environment must perform these service subroutine calls. The library service subroutines are described in Figure 10.

To	*Call entry name*
Initialize library work area	$FMYINT
Delete library work area	$FMYTRM
Specify abnormal termination routine	$FMYSPE

Figure 10. Library services and MFSL subroutines (called by assembler programs only)

The user controls the service subroutines through the calling parameters and interprets their actions through returned parameters or return codes. The user communication with the service subroutines is shown in Figure 11.

MFSL Library Work Area

The MFSL library work area is created and initialized by the service subroutine $FMYINT. The library work area is deleted by $FMYTRM. The initialization subroutine should be called before any other MFSL subroutines are invoked. The termination routine should be called after all MFSL subroutines have been invoked.

Interruption Handling

The MFSL subroutines intercept floating-point exceptions to provide error-handling support. The user can specify a subroutine to receive control when a program check other than floating-point occurs or when an abnormal termination macro instruction (STOPTASK) is issued.

Service subroutines	*Calling parameter list (pointed to by register 0)*	*Return code (RC) in register 0 or parameter list (pointed to by register 0)*
$FMYINT	*Word 1:* Address of a flag field to describe the operating environment and required MFSL functions (see note below). *Word 2:* If dynamic allocation option is not used, the address of a pointer to a storage area for the library work area.	*Word 2:* Address of a pointer to the library work area. **Or** *RC=0:* Successful first initialization (pointer to library work area is in task work stack descriptor). *RC=2:* Successful initialization (pointer to library work area is in task work stack descriptor). *RC=4:* No MFSL functions requested. *RC=6:* Internal error occurred (initialization results unknown).
$FMYTRM	None	*RC=0:* Successful release of library work area. *RC=2:* No library work area existed when $FMYTRM was called. *RC=4:* Reserved. *RC=6:* Internal error occurred (termination results unknown).
$FMYSPE	*Word 1:* Address of a pointer to the routine to get control on abnormal termination. *Word 2:* Address of a pointer to the parameter list passed to the abnormal termination routine.	*Word 3:* Address of a pointer to the abnormal termination routine that existed when $FMYSPE was called (if none existed, this word contains zero). *Word 4:* Address of a pointer to the parameter list that was specified on the last call to $FMYSPE (if none was specified, this word contains zero).

Note. The option flags are as follows:

Bits	*Meaning*
0–3	Reserved
4	Conversion subroutines used
5	Mathematical subroutines used
6–7	Reserved
8	Dynamic allocation allowed
9–14	Reserved
15	Abnormal termination exit allowed

Figure 11. User communication with service subroutines

Configuration Requirements

The system installation must be properly equipped to use MFSL. Both hardware and software requirements are given below.

Hardware Support

1. Processors:
   - 4953
   - 4955
2. Processor Features (optional):
   - Floating-point Feature Number 3920 (4955 Processor only)
3. Primary Storage Considerations:
   For MFSL (See also Appendix A.):
   - The size of user-called subroutines
   - The size of subroutine-called subroutines
   - The size of the error exit routine
   - The size of operating system services

   For User:
   - The size of linkage code for all calls
4. Secondary Storage (Approximate Sizes):
   For MFSL:
   - The size of the MFSL object library, if installed, 38 912 bytes
   - The size of the MFSL load module library, if installed, 139 008 bytes
   - If both libraries are installed, the total, 177 920 bytes

Program Support

MFSL is supported by the Realtime Programming System program product and the Program Preparation Subsystem program product. These programs allow the user to assemble, compile, combine, and execute programs that use the MFSL subroutines.

If floating-point operations are used and the floating-point hardware feature is not installed, then the floating-point emulator option of the Realtime Programming System must be installed. If no functions that require REAL arithmetic are used, MFSL has no requirement for floating-point support in either hardware or software. The functions that require floating-point support are the following:

- Logarithmic and exponential
- Trigonometric
- Hyperbolic
- Exponentiation with REAL variables
- Arithmetic with REAL variables
- EBCDIC conversions with REAL variables

The following topics describe the user actions required to use MFSL. They are presented in the order that represents a typical application.

Installing MFSL

At any time after system generation, the distribution library copies of the MFSL subroutines can be copied from diskette to the system library specified by the user. Distribution libraries are provided for both object and composite modules, so no further assemblies or compilations are required. Both object and composite modules are required to fully support task set preparation. During phase 1 of the application builder, the MFSL object modules can be included in overlay structures. During phase 3 of the application builder, the MFSL composite modules can be included in the task set load module. Installation of the MFSL object and composite module libraries is independent. If there is no requirement for the MFSL object modules (such as overlay programming), the object module library need not be installed.

Sharing MFSL Modules

MFSL uses reentrant programming techniques to allow different user programs to share the subroutines. The MFSL subroutines can be resident or transient in any user partition. The subroutines can be shared by programs in one or more partitions.

Referencing MFSL from Assembler Programs

The assembler user references all MFSL subroutines through the CALL macro instruction using the PARM= parameter list to pass arguments. Usually, the subroutines return the function value in a register. In some cases, the value is returned in a storage location whose address is passed as an input argument. An example of the assembler interface is

```
          CALL   SIN,PARM=(X)
          . . .
X         DC     E'0'
```

which finds the sine of the argument X. X is the name of a single-precision floating-point field that is defined in the caller's program. The result is returned in floating-point register 0 following the call.

Referencing MFSL from FORTRAN IV Programs

The FORTRAN IV user references some MFSL subroutines as FORTRAN IV FUNCTION subprograms, some subroutines by CALL statements, and other subroutines implicitly in FORTRAN IV statements.

The mathematical subroutines supported by FORTRAN IV as FUNCTION subprograms are typically coded as

```
A = SIN(X)
```

where SIN is the name of the FUNCTION subprogram.

The FORTRAN IV user can invoke other subroutines through the CALL statement, such as

```
CALL FCTST(I,J)
```

where the function test subroutine is invoked with I and J as parameters.

Implicit references to the MFSL subroutines are made in two ways. The first way is through the FORTRAN IV exponentiation notation, such as

```
Y = X**2.5
```

where a subroutine is needed to raise X to the 2.5 power. The second way in which an implicit reference is made is through the compiler selection of INTEGER*2 or INTEGER*4 subroutines according to the CMPAT or NOCMPAT compiler options. (See the publication *FORTRAN IV: Language Reference*, GC34-0133, for further information on the compiler option.)

Building Applications with MFSL

After MSFL is installed in the system, the MFSL modules can be combined with user-written programs to build the application. When the automatic call facility of the application builder is used to reference MFSL, the user need only make source code references to the subroutines. In the compile-load-go process, the application builder automatically includes the MFSL modules in the application task set load module.

Executing the Application

During execution there are no user actions or interfaces with the MFSL subroutines. The MFSL functions interact with the application programs and the library work area.

Managing Storage

The following discussion of storage management topics is provided to show storage-critical installations how they can use the flexibility of MFSL to minimize the MFSL storage requirement. The design of typical applications will not require the use of these storage optimization techniques.

Storage Optimization Techniques

For any given system generation and application design, there is an optimum way to install the MFSL subroutines to minimize their storage requirements. The Series/1 program products that allow the user to prepare efficient software configurations for MFSL applications are the following:

- The shared task set capability of the Realtime Programming System program product
- The application builder facility of the Program Preparation Subsystem program product

The Realtime Programming System allows the user to take advantage of the reentrant property of the MFSL subroutines. When included in the shared task set, an MFSL subroutine can be executed by different programs in different partitions at the same time. A single in-storage copy of the subroutine in the shared task set eliminates the need to allocate storage for the subroutine in any other partition.

The application builder allows a user task set to minimize its MFSL storage requirement in three ways: First, the application builder allows the task set to include only the subset of MFSL subroutines that it actually needs. Secondly, the

application builder allows a group of main programs that executes in one task set to include MFSL subroutines in a shared subroutine area so that a single in-storage copy of each subroutine can be executed by any main program or subprogram in the partition. Finally, the application builder allows the MFSL subroutines to be used in overlay structures.

The basic MFSL installation questions are these:

- Which MFSL subroutines should go into the shared task set?
- Which MFSL subroutines should go into a shared subroutine area?

The answers to these questions must be based on an analysis of unique system and application characteristics. The following topics describe the problems and some methods for their solution.

Using a Shared Task Set for MFSL

Different applications can be independently prepared for each partition. If this approach is taken, the application builder will combine the programs for each task set, and MFSL subroutines that are referenced will be included in each task set. This approach will result in a software configuration such as the one shown on the "problem" side of Figure 12. In this case, a four-partition system has been generated. Partition 0 contains the Realtime Programming System, and partitions 1–3 are for user applications. Because each of the applications uses the MFSL subroutines, MFSL is built into each of the user partitions. This approach uses storage inefficiently because many of the same MFSL modules are included three times.

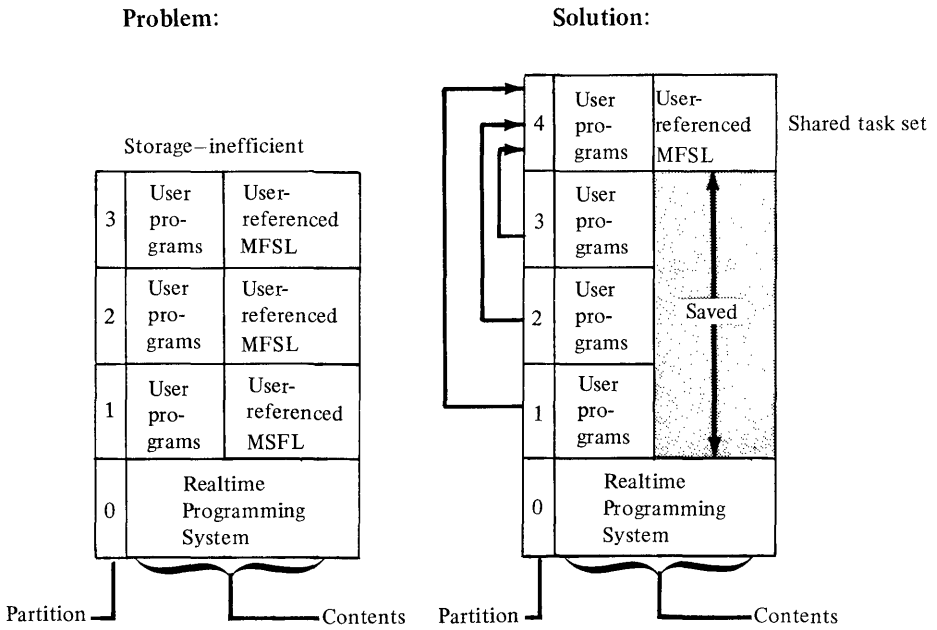


Figure 12. Using a shared task set for MFSL

On the "solution" side of Figure 12, the system has been redesigned to include partition 4 as a shared task set. The MFSL subroutines included in the shared task set are now referenced from programs that execute in the three original user partitions. Now, there is only one copy of each MFSL subroutine in storage. The cross-partition references are made possible by using the application builder to combine each user task set in partitions 1–3 with the shared task set in partition 4.

The specific MFSL subroutines placed in the shared task set can be selected in two ways: One method is simple and flexible, but it does not ensure an absolute minimum storage requirement. The other method requires more analysis, but it does eliminate all redundancy.

The simple way is to include all MFSL modules in the shared task set. Then, any application that uses MFSL can be combined with the shared task set to resolve its external references to MFSL subroutines. This method is appropriate if the application usage of MFSL is unknown or unpredictable for a variable set of applications.

The other method of selecting MFSL subroutines for the shared task set is shown in Figure 13. Here, the subroutine requirements of each partition are known in advance. The subroutines used can be tabulated for all partitions, and only those that are used by more than one partition need be placed in the shared task set. The other subroutines, which are used only in one partition, can be included with that partition. This method is appropriate for a dedicated set of applications where efficiency is more important than flexibility.

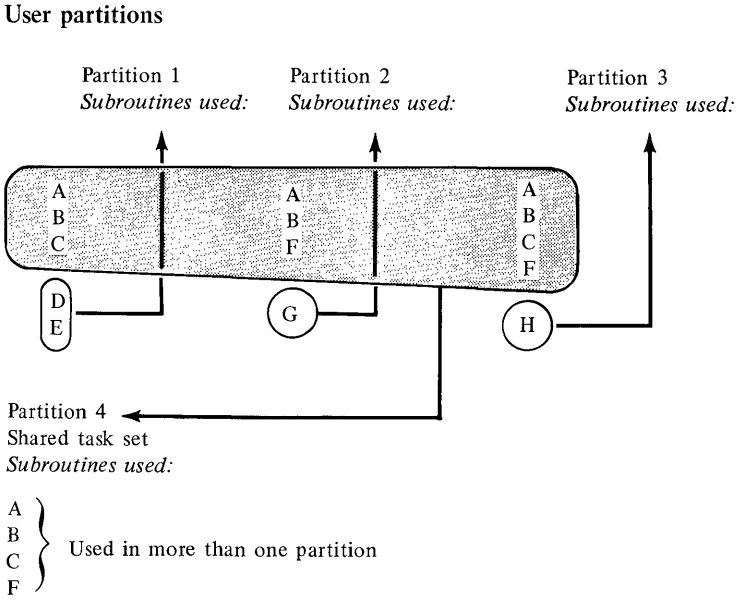


Figure 13. Selecting subroutines for the shared task set

Using a Subset of MFSL

The organization of MFSL, which provides each function as a separate object or composite module, provides a great advantage to the user who needs only a few MFSL functions. In the process of building applications, the application builder can construct overlays by including the specified MFSL object modules during phase 1. The application builder can also resolve external references from user-written programs to MFSL subroutines by bringing the appropriate MFSL composite modules into a task set load module during phase 3. Only the MFSL modules that are used (plus supporting data areas) are made part of the task set load module. The result is that each user task set requires storage only for a subset of the total MFSL. The user task set does not require storage for unused subroutines. Figure 14 illustrates the problem that would exist if the entire subroutine library had to be in a task set that used only a subset of the MFSL functions. The storage estimating information in Appendix A shows how to approximate the actual storage that a task set will need for MFSL, based on the functions used.

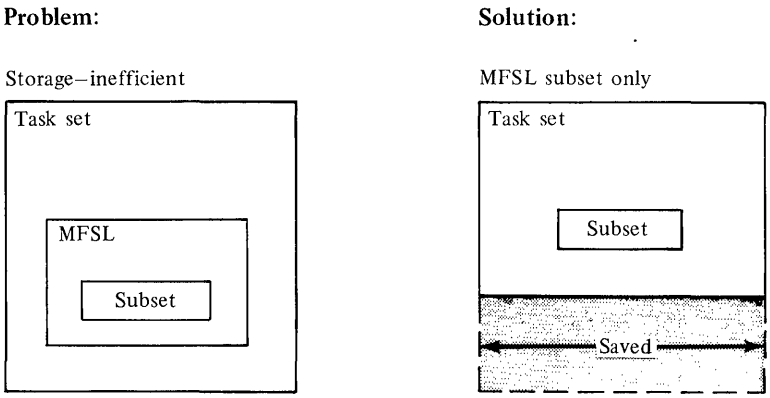


Figure 14. Using an MFSL subset in a task set

Including MFSL Subroutines in a Shared Subroutine Area

When an application contains many programs, it is often convenient to develop and test the application in logical groups of programs. As the programs that make up a functional unit of the application are completed, they can be combined to form task set load modules that can be executed and tested independently. When all of the functional units are completed, their composite modules can be combined to form the task set load module for the application. This step-by-step process allows testing during development and makes it easier to produce the complete set of programs for the application.

When identical MFSL modules are used throughout a large application, they should be combined into a shared subroutine area to conserve storage space. This means that the MFSL library should be specified as input to the application builder only during the phase 3 process when the composite modules are combined to build the task set load module.

The "problem" side of Figure 15 shows the task set load module structure that would result if MFSL were specified as input to the application builder during separate phase 1 combinations of the three main programs shown. Because two of the main programs had references to MFSL, the MFSL modules were included twice. This task set structure does not take advantage of a shared subroutine area.

The "solution" side of Figure 15 shows the same task set structure modified to use a shared subroutine area. When each of the main programs was combined during phase 1 execution of the application builder, MFSL was not specified as input. The external references to MFSL modules were left unresolved in the composite modules. Finally, during Phase 3 execution, MFSL was specified as input to the application builder. At this time, the MFSL modules were included. However, only one copy of each MFSL subroutine was brought into the task set load module. Now, external references to MFSL from any program in the task set can be resolved to a single in-storage copy of each MFSL subroutine. The storage allocated to MFSL is in a shared subroutine area rather than in the composite modules of each main program.

The process of selecting MFSL modules for a shared subroutine area is similar to the process of selecting MFSL subroutines for the shared task set. Figure 16 shows the tabular method of selecting commonly used subroutines. This approach produces a shared subroutine area of minimum size. However, the total task set size would be the same if all of the referenced subroutines were placed in the shared subroutine area. Therefore, on a task set basis, the simple approach of placing all MFSL modules in the shared subroutine area requires the minimum amount of storage for MFSL.

Problem: **Solution:**

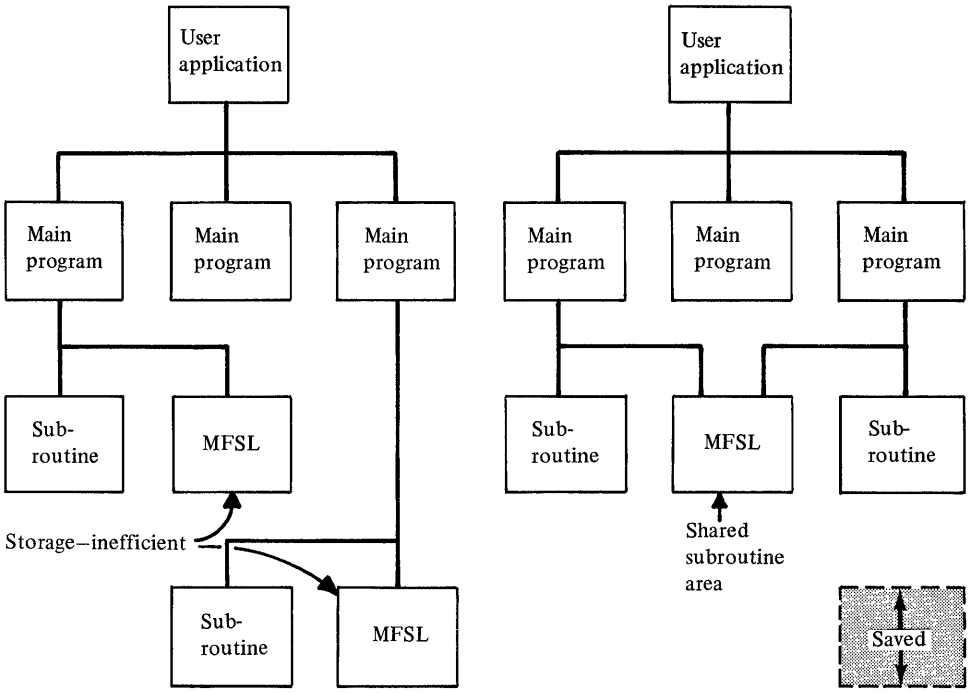


Figure 15. Using a shared subroutine area for MFSL

Within task set

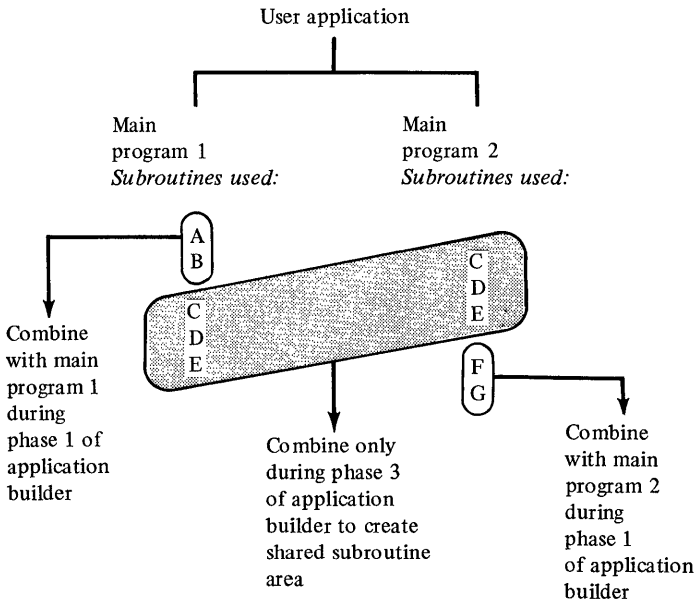


Figure 16. Selecting subroutines for a shared subroutine area

Appendix A. Storage Estimating

The following information can be used to determine the additional storage required by application programs to support MFSL functions. The storage requirements analysis must consider these factors:

- Basic MFSL storage requirement
- MFSL functional storage requirement
- Common MFSL subroutines

The module sizes and all resulting storage requirements given in this appendix are approximate.

Basic MFSL Storage Requirement

Any application that uses MFSL must include the following:

- Space in the partition dynamic area (free storage) for a library work area of 32 bytes
- Space for the MFSL error exit routine ($FMYABN) of 136 bytes

The total basic MFSL storage requirement is 168 bytes.

MFSL Functional Storage Requirements

When an application program uses an MFSL function, the user interface is either a function name or the FORTRAN IV exponentiation notation. The name or notation used in the application program provides a link to the MFSL subroutines.

In assembler language programs, most MFSL functions require an interface module plus one or more dependent subroutine modules to perform the function. The interface modules are not required by FORTRAN IV programs. For example, Figure 17 shows the module dependencies that result from references to the modular arithmetic and hyperbolic tangent functions. The same module dependencies result for references from the FORTRAN IV program or the assembler program, except for the interface modules.

The MFSL storage requirement for the functions shown in Figure 17 includes the following:

- The interface modules for the assembler program, MOD and TANH: total 8 bytes
- The modular arithmetic function $FMEMOD and its module dependencies for dividing and multiplying integers, $FMDDDD and $FMDDMD: total 424 bytes
- The single-precision hyperbolic tangent function, $FMETNH, and its module dependency for the single-precision exponential function, $FMEEXP: total 432 bytes.

The total for all of the above functions is 864 bytes. This storage requirement is in addition to the basic MFSL requirement of 168 bytes for the library work area and the error exit routine. Thus, the total MFSL storage requirement for the user application is 1032 bytes.

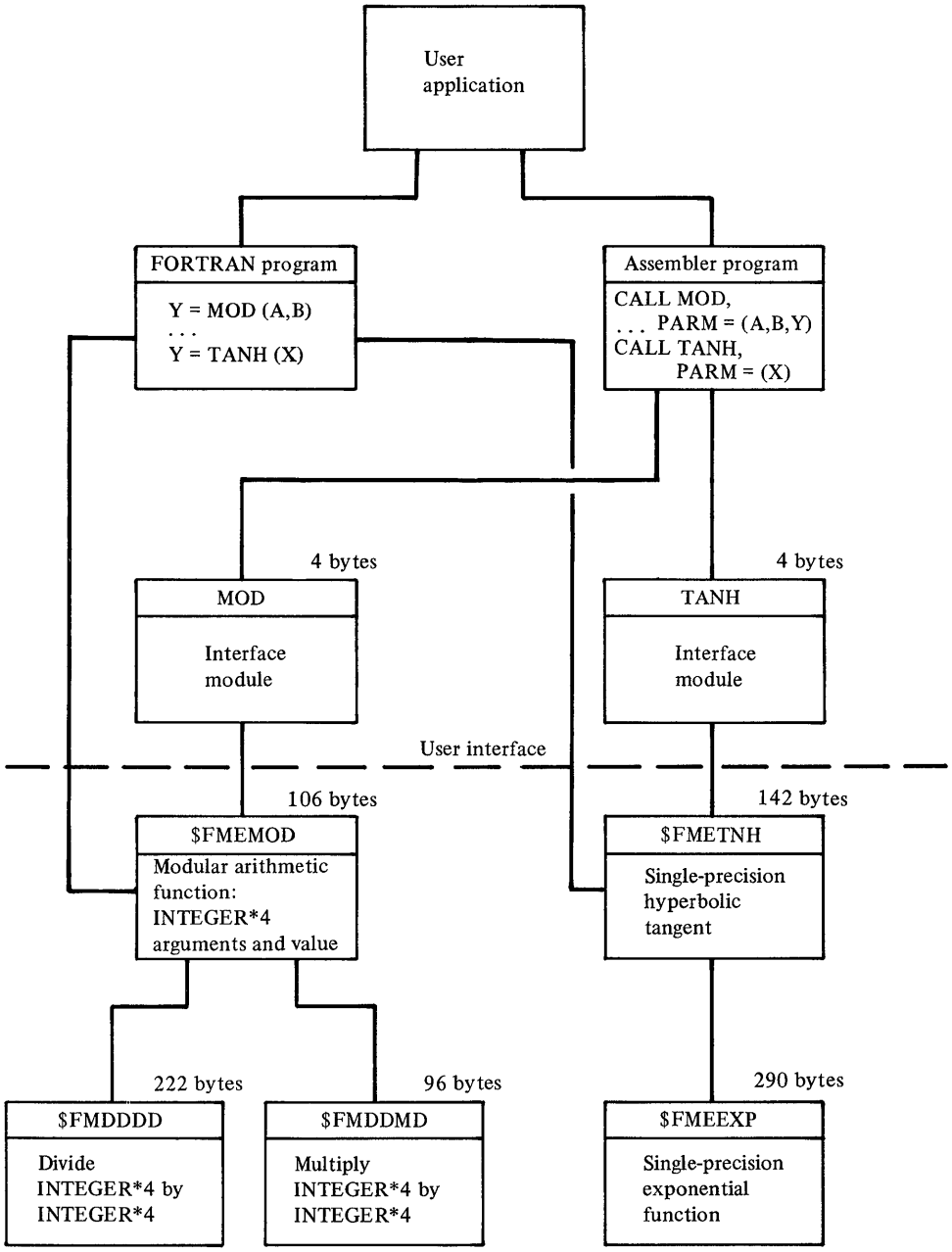


Figure 17. Typical MFSL module dependencies

Common MFSL Subroutines

The above example using the data in Figure 17 illustrates an important point to consider when estimating storage requirements: Only one copy of each required MFSL module need be included in the task set load module. If the functional storage requirements for the FORTRAN IV program and the assembler program in Figure 17 were calculated independently and the results added together to obtain the application storage requirements, then the basic MFSL modules would be counted twice. The functional storage requirement would appear to be almost twice as large as it really is.

Common subroutines can also reduce the apparent functional storage requirement for a single program. This case is illustrated by the sine and cosine functions. In Figure 18, an assembler program is shown referencing both the sine and cosine functions. Because the sine-cosine subroutine $FMESNC is common

to both functions, the total functional storage requirement is only 206 bytes. The requirement for sine or cosine alone is 202 bytes. The only module requirement for the additional function is the interface module. In FORTRAN IV, there is no additional module requirement; the only requirement is 198 bytes for $FMESNC.

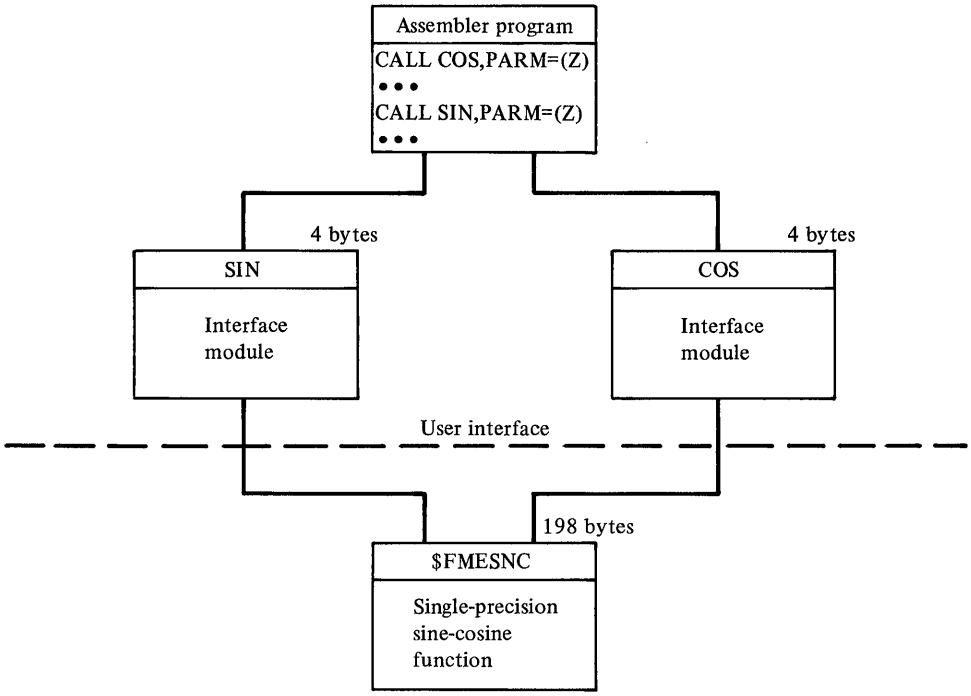


Figure 18. Common subroutine usage

The basic tool for MFSL storage estimating is the MFSL functional storage requirements table given in Figure 19. This table gives the total storage required for each MFSL function. Module dependencies are shown, including the size of each module required.

The table should be used in the following way:

1. Identify all MFSL functions used by your application. (FORTRAN IV programs should include their module requirements for exponentiation by using the entry names given in Figure 6 for assembler programs.)
2. Look up each function you use in the "Function or entry name" column. Note the modules required, and cumulatively total the number of bytes given in the "Maximum storage required" column. (The last column is the sum of the individual modules required. Estimates for FORTRAN IV programs should subtract the 4 bytes used by the interface module.)
3. As each function is totaled, check the modules required to make sure that none are counted twice. Whenever a common subroutine is encountered, subtract its size from the "Maximum storage required" entry before adding the functional storage requirement to the cumulative total.

When the storage required for all the functions is totaled in this way, the result is your application's MFSL functional storage requirement. Add this to the basic MFSL requirement to find the total MFSL storage requirement for your application.

Function or entry name	*Modules required (assembler only†)*	*Module size (bytes)*	*Maximum storage required (bytes)*
$DCIN	$DCIN† $FFDCIN $FFCHRC $FFFTEN	4 880 92 412	1388
$DCOT	$DCOT† $FFDCOT $FFFTEN	4 1146 412	1562
$ECIN	$ECIN† $FFDCIN $FFCHRC $FFFTEN	4 880 92 412	1388
$ECOT	$ECOT† $FFDCOT $FFFTEN	4 1146 412	1562
$FCOT	$FCOT† $FFDCOT $FFFTEN	4 1146 412	1562
$FCOTD	$FCOTD† $FFDCOT $FFFTEN	4 1146 412	1562
$FMYINT	$FMYINT $FMYABN (Weak Reference)	336 136	472
$FMYSPE	$FMYSPE	18	18
$FMYTRM	$FMYTRM	190	190
$I2CIN	$I2CIN† $FFI2CIN $FFCHRC	4 146 92	242
$I2COT	$I2COT† $FFI2COT	4 156	160
$I4CIN	$I4CIN† $FFI4CIN $FFCHRC	4 158 92	254
$I4COT	$I4COT† $FFI4COT	4 164	168
ALOG	ALOG† $FMELOG	4 232	236
ALOG10	ALOG10† $FMELOG	4 232	236
AMAX0	AMAX0† $FMEAMX0	4 56	60
AMAX0#	AMAX0#† $FMNAMX0	4 56	60

Figure 19. MFSL functional storage requirements table (Part 1 of 4)

Function or entry name	*Modules required (assembler only†)*	*Module size (bytes)*	*Maximum storage required (bytes)*
AMAX1	AMAX1† $FMEAMX1	4 46	50
AMIN0	AMIN0† $FMEAMN0	4 56	60
AMIN0#	AMIN0#† $FMNAMN0	4 56	60
AMIN1	AMIN1† $FMEAMN1	4 46	50
AMOD	AMOD† $FMEAMOD	4 36	40
ATAN	ATAN† $FMEATN	4 264	268
ATAN2	ATAN2† $FMEATN	4 264	268
COS	COS† $FMESNC	4 198	202
DATAN	DATAN† $FMLATN	4 378	382
DATAN2	DATAN2† $FMLATN	4 378	382
DCOS	DCOS† $FMLSNC	4 332	336
DEXP	DEXP† $FMLEXP	4 390	394
DEXPD	DEXPD† $FMLLPL	4 62	66
DEXPI	DEXPI† $FMLLPD	4 94	98
DEXPI#	DEXPI#† $FMLLPF	4 78	82
DIM	DIM† $FMEDIM	4 36	40
DLOG	DLOG† $FMLLOG	4 302	306
DLOG10	DLOG10† $FMLLOG	4 302	306
DMAX1	DMAX1† $FMLMAX1	4 46	50
DMIN1	DMIN1† $FMLMIN1	4 46	50

Figure 19. MFSL functional storage requirements table (Part 2 of 4)

Function or entry name	*Modules required (assembler only†)*	*Module size (bytes)*	*Maximum storage required (bytes)*
DMOD	DMOD† $FMLDMOD	4 38	42
DSIGN	DSIGN† $FMLSGN	4 32	36
DSIN	DSIN† $FMLSNC	4 332	336
DSQRT	DSQRT† $FMLSQR	4 116	120
DTANH	DTANH† $FMLTNH $FMLEXP	4 188 390	582
DVCHK	DVCHK† $FMDDCK	4 18	22
DVCHK#	DVCHK#† $FMEDCK	4 16	20
EXP	EXP† $FMEEXP	4 290	294
EXPE	EXPE† $FMEEPE $FMEEXP $FMELOG	4 62 290 232	588
EXPI	EXPI† $FMEEPD	4 94	98
EXPI#	EXPI#† $FMEEPF	4 78	82
FCTST	FCTST† $FMDFNT	4 40	44
FCTST#	FCTST#† $FMFFNT	4 32	36
IDIM	IDIM† $FMLIDIM	4 86	90
IDIM#	IDIM#† $FMNIDIM	4 40	44
IDIV	IDIV† $FMDDDD	4 222	226
IEXP	IEXP† $FMDDPD	4 186	190
IEXP#	IEXP#† $FMFFPF	4 36	40
IMULT	IMULT† $FMDDMD	4 96	100

Figure 19. MFSL functional storage requirements table (Part 3 of 4)

Function or entry name	*Modules required (assembler only†)*	*Module size (bytes)*	*Maximum storage required (bytes)*
ISIGN	ISIGN† $FMDSGN	4 40	44
ISIGN#	ISIGN#† $FMFSGN	4 22	26
MAX0	MAX0† $FMEMAX0	4 94	98
MAX0#	MAX0#† $FMNMAX0	4 46	50
MAX1	MAX1† $FMEMAX1	4 102	106
MAX1#	MAX1#† $FMNMAX1	4 56	60
MIN0	MIN0† $FMEMIN0	4 94	98
MIN0#	MIN0#† $FMNMIN0	4 46	50
MIN1	MIN1† $FMEMIN1	4 102	106
MIN1#	MIN1#† $FMNMIN1	4 56	60
MOD	MOD† $FMEMOD $FMDDDD $FMDDMD	4 106 222 96	428
MOD#	MOD#† $FMNMOD	4 36	40
OVERFL	OVERFL† $FMDOFL	4 18	22
OVERFL#	OVERFL#† $FMFOFL	4 16	20
SIGN	SIGN† $FMESGN	4 32	36
SIN	SIN† $FMESNC	4 198	202
SQRT	SQRT† $FMESQR	4 130	134
TANH	TANH† $FMETNH $FMEEXP	4 142 290	436

Figure 19. MFSL functional storage requirements table (Part 4 of 4)

Index

Mathematical and Functional Subroutine
Library: Introduction
GC34-0138-0

READER'S
COMMENT
FORM

YOUR COMMENTS, PLEASE . . .

Your comments assist us in improving the usefulness of our publications; they are an important part of the input used in preparing updates to the publications. All comments and suggestions become the property of IBM.

Please do not use this form for technical questions about the system or for requests for additional publications; this only delays the response. Instead, direct your inquiries or requests to your IBM representative or to the IBM branch office serving your locality.

Corrections or clarifications needed:

Page Comment

Cut or Fold Along Line

What is your occupation?_____
Number of latest Technical Newsletter (if any) concerning this publication: _____
Please indicate your name and address in the space below if you wish a reply.

_____

_____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A.
(Elsewhere, an IBM office or representative will be happy to forward your comments.)

GC34-0138-0

Your comments, please . . .

This manual is part of a library that serves as a reference source for IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold

Fold

First Class
Permit 40
Armonk
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

IBM Corporation
Systems Publications, Dept 27T
P.O. Box 1328
Boca Raton, Florida 33432

Fold

Fold

IBM

International Business Machines Corporation
General Systems Division
5775D Glenridge Drive N.E.
P.O. Box 2150, Atlanta, Georgia 30301
(U.S.A. only)

Cut Along Line

IBM Series/1 Mathematical and Functional Subroutine Library: Introduction Printed in U.S.A. GC34-0138-0

International Business Machines Corporation

General Systems Division
5775D Glenridge Drive N.E.
P. O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

IBM Series/1 Mathematical and Functional Subroutine Library: Introduction Printed in U.S.A. GC34-0138-0

GC34-0138-0