180
12-30-74
ANCR-1183

Dr - 1101
UC-32

MASTER

**MIMSY
MULTITASK INTERPRETIVE MATHEMATICS
SOFTWARE FOR PDP-15 COMPUTERS**

W.R. Myers

Aerojet Nuclear Company

IDAHO NATIONAL ENGINEERING LABORATORY

Idaho Falls, Idaho — 83401

DATE PUBLISHED—NOVEMBER 1974

PREPARED FOR THE

U. S. ATOMIC ENERGY COMMISSION

IDAHO OPERATIONS OFFICE UNDER CONTRACT AT(10-1)-1375

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency Thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Printed in the United States of America
Available from
National Technical Information Service
U. S. Department of Commerce
5285 Port Royal Road
Springfield, Virginia 22161
Price: Printed Copy \$4.00; Microfiche \$2.25

LEGAL NOTICE

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

ANCR-1183

Mathematics and Computers
TID-4500

MIMSY
MULTITASK INTERPRETIVE MATHEMATICS
SOFTWARE FOR PDP-15 COMPUTERS

W. R. Myers

NOTICE

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

AEROJET NUCLEAR COMPANY

Date Published - November 1974

PREPARED FOR THE U. S. ATOMIC ENERGY COMMISSION

IDAHO OPERATIONS OFFICE

UNDER CONTRACT NO. AT(10-1)-1375

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

216

ABSTRACT

MIMSY is an extensive, completely re-entrant floating point software package for PDP-15 computers equipped with API and EAE. Space requirements are approximately 800 words for the arithmetic operations, 460 additional locations for the input and output controllers, 550 locations for the common functions, and 470 locations for symmetric matrix arithmetic. A 32-word scratch area is required in each application module. The package is interpretive and requires only one-word instructions, except for the "enter" operation which requires two words to establish the user scratch area. Normal PDP-15 addressing conventions are followed, allowing operation in up to 32K words of memory. The interpreter may be easily adapted to a variety of operating systems by modification of the short (bookkeeping) prologue.

TABLE OF CONTENTS

	<u>Page</u>
I. INTRODUCTION	1
II. OPERAND MODES	3
1. FLOATING ACCUMULATOR FORMAT	3
2. DATA	3
3. INTEGER	3
4. FLOATING POINT	4
5. EXTENDED PRECISION	4
6. MATRIX FORMAT	4
III. RE-ENTRANT OPERATION	5
1. INTERNAL OPERATION	5
2. USER TASK SCRATCH AREA	6
3. INTERPRETER ENTRY PROCEDURE	7
IV. INTERPRETIVE MODE INSTRUCTIONS	9
1. ENTER AND LEAVE INSTRUCTIONS	9
2. LOAD AND STORE OPERATIONS	10
3. FLOATING ACCUMULATOR MODIFICATION	10
4. ARITHMETIC INSTRUCTIONS	11
5. PROGRAM CONTROL INSTRUCTIONS	12
6. INPUT AND OUTPUT INSTRUCTIONS	13
7. FUNCTION EVALUATION	14
8. MATRIX OPERATIONS	15
V. INPUT AND OUTPUT	16
1. THE INPUT INTERPRETER	16
2. THE OUTPUT CONTROLLER	17
3. OUTPUT FORMAT	17
VI. FUNCTION EVALUATION	19
1. SQUARE	19
2. SQUARE ROOT	19
3. LOGARITHM	19
4. EXPONENTIAL	20
5. SINE	20
6. COSINE	20
7. ARCTANGENT	20
VII. MATRIX OPERATIONS	22
1. THE MATRIX OPERATIONS	22
2. MATRIX AND VECTOR FORMAT	23
3. SCRATCH AREA REQUIREMENTS	23

TABLE OF CONTENTS (Cont.)

	<u>Page</u>
APPENDICES	
A. MIMSY INSTRUCTION SUMMARY	25
B. EXAMPLE ENTRY PROTOCOL	29
C. PROGRAMMING EXAMPLES	31
1. DOUBLE-PRECISION SIGNED FLOAT	32
2. GENERAL EXPONENTIAL	33
3. LEAST SQUARES FITTING	34

I. INTRODUCTION

MIMSY (Multitask Interpretive Mathematics Software) is an extensive re-enterable floating point arithmetic software package for Digital Equipment Corporation PDP-15 computers equipped with the Extended Arithmetic Element (EAE) and Automatic Priority Interrupt (API). MIMSY implements thirty-five instructions including five immediate operand instructions, seven common functions, three-way compare and branch, inverse divide, and three IØ instructions. Four operand formats are available: 18-bit (positive) data, 18-bit (signed) integers, and two- and three-word floating point formats. The package is interpretive and floating mode coding follows the PDP-15 page-mode addressing conventions allowing indirect and indexed references. Operand mode may be changed in the course of a computation by one of the "operate-class" instructions allowing mixed mode arithmetic; operand fetch and store operations depend on the mode, but all internal operations are performed on extended precision floating-mode operands.

Input and output are handled through the "floating accumulator" utilizing user-written character handling routines. The input interpreter is free-format, allowing input of integer, floating, or fixed-point numbers. The output controller transmits formatted ASCII character strings to the user output routine. The output format may be integer, fixed-point or floating-point (I, F, or E format) and is controlled by one of the interpretive instructions.

The common functions are included as "operate-class" instructions. Square-root utilizes integer arithmetic and executes faster than multiply or divide. The sine, cosine, natural logarithm and exponent functions are evaluated in series expansion, the arctangent by a continued fraction. The series lengths are calculated, using empirical formulas, from the magnitudes of the arguments to minimize execution times. All functions are accurate to at least 34 bits (10 decimal digits).

Four operations on (upper triangular) symmetric matrices have been included in MIMSY for use in least-squares fitting applications. The operations are in-place matrix inversion, vector multiplication by a matrix, matrix scaling, and construction of the scaling vector.

Re-enterability is accomplished by maintaining a 32-word "scratch" area in each user program. The scratch area contains all necessary permanent data, such as the floating accumulator, operand data mode indicator, loop and format control words, and temporary scratch locations needed by the interpreter. The index register is saved in the first word of the scratch area which may be incremented by an interpretive instruction to permit sequential indexed addressing of data. The floating accumulator components, operand mode indicator and format words are available for "normal-mode" initialization or manipulation. If the matrix arithmetic operations are used, the scratch area is increased by twelve words to accommodate pointers to the matrix, required vectors, and an error return address.

MIMSY is intended primarily for use in dedicated applications, not as a general purpose computational tool. In the interest of execution speed and program size, there are no means provided for detection of arithmetic overflow or underflow, and the functions return "reasonable" results for "nonsense" arguments. It is assumed that the programmer will know in advance the range of his variables, and will use appropriate methods to avoid anomalous results.

There are four basic sections: arithmetic, input/output, functions, and matrix operations. Each segment requires the presence of the preceding, but not the following, segment. If the functions are deleted, for example, the matrix operations must also be deleted; the arithmetic and input/output sections, however, would not be affected.

The approximate sizes of the segments are:

ARITHMETIC	800
INPUT AND OUTPUT	460
FUNCTIONS	550
MATRIX OPERATIONS	470

The total size of the package is approximately 2280 locations plus the size of the system-dependent prologue for entry to interpretive mode.

II. OPERAND MODES

MIMSY operand reference instructions may be used with four distinct types of variables: positive 18-bit data, 18-bit two's-complement (signed) integers, and two- or three-word floating point numbers. The interpretive instruction 'MODE' is used to set the operand reference mode which will be in effect until another 'MODE' instruction is encountered. The ability to handle implicitly positive and signed integers, combined with the use of the immediate-class add, multiply and divide instructions (ADI, MPI, DVI), permits relatively fast and efficient high precision computation on normal experimental data without the need for explicit conversion to and from floating formats. "Mixed-mode" computation is performed simply by switching the operand reference mode as needed in the course of a calculation.

1. FLOATING ACCUMULATOR (FAC) FORMAT

While several distinctively different operand formats may be used, the format for all numbers, regardless of mode, is the same during computation. The interpretive routines perform all operations on two four-word registers, the "floating accumulator" (FAC), and the "floating operand register" (FMQ). The operand mode instruction is used to set the interpreter to make the proper format transformations while moving an operand between memory and the working registers. The FAC and FMQ both have the form:

	S
TWO'S EXPONENT	
0 HIGH MANTISSA	
LOW MANTISSA	

sign in bit 17 (1 if negative).
exponent (two's-complement form).
17-bit high order mantissa.
18-bit low order mantissa.

The sign-word is zero for positive numbers and '1' for negative numbers, the exponent is in 18-bit two's-complement form, the 35-bit mantissa is a normalized, positive, binary fraction. All variables loaded from memory are normalized if necessary as they are picked up by the interpreter. The FAC is normalized after each arithmetic operation.

2. DATA (MODE 0)

Setting the operand reference mode to 'zero' indicates that all following memory operands are one-word (18-bit) positive integers (range $0 \leq X < 2^{18}$). The operands are automatically converted to the normalized fraction format when brought into the working registers. When a result is returned to memory by the 'DAC' instruction, it is rounded to the nearest positive integer; negative numbers are set to zero. If the magnitude of the number exceeds $2^{18}-1$, but is less than 2^{35} , the low-order bits are stored. If the magnitude exceeds 2^{35} , zero is returned.

3. INTEGER (MODE 1)

While the operand reference mode is 'one', all operands will be interpreted as one-word two's-complement integers (range $-2^{17} < X < 2^{17}$).

A result returned to memory by the 'DAC' instruction is rounded to the nearest integer. If the magnitude exceeds $2^{17}-1$, the low-order bits are stored; if the magnitude exceeds 2^{35} , the stored result will be zero.

4. FLOATING POINT (MODE 2)

Memory operands in format 'two' are two-word floating point numbers in the form of 26-bit normalized fraction with one sign bit and an associated 9-bit two's-complement binary exponent. The 26-bit mantissa allows a precision of number representation greater than seven decimal digits; the nine-bit exponent provides a magnitude range from 10^{-76} to 10^{+76} . Floating point operands have the form:

When a result is returned to memory by the 'DAC' instruction the mantissa is truncated to 26 bits (not rounded); the exponent is stored as the low nine bits of the FAC exponent; no tests are made for magnitude overflow or underflow. When the operand is brought into one of the working registers the exponent sign bit (bit '9' of word 1) is extended through bits 0-8 of the FAC or FMQ exponent.

5. EXTENDED PRECISION (MODE 3)

The extended precision floating point format is similar to that of the working registers. Mode '3' operands occupy three consecutive memory words, have a 35-bit mantissa, and an 18-bit two's-complement exponent. The precision is greater than 10 decimal digits, the magnitude range is from 10^{-39000} to 10^{+39000} . Extended precision operands have the form:

6. MATRIX FORMAT

MIMSY includes four operations on symmetric matrices. The matrices are stored in upper triangular form, column by column. The operand mode of matrix entries must be real (2) or extended precision (3).

III. RE-ENTRANT OPERATION

MIMSY operates in a multi-task interrupt-mixing environment; it is re-entrant in the sense that it may be "simultaneously" used by several separate mutually interrupting tasks. Re-entrancy is accomplished using "pure procedure" where practicable, protected procedure where necessary, and by using individual memory areas within each task to provide the pseudo-registers, parameters, and scratch locations required by the interpreter.

1. INTERNAL OPERATION

The major requirement for re-entrant code is that no word of that code shall be modified. MIMSY conforms to this requirement in all but a few procedures (see below), employing separate memory blocks in each task to provide a set of required alterable locations.

The PDP-15 index and limit registers are used to hold the address of the user task scratch area, with the index register as a working register and the limit register as the address constant. The entry procedure (see below) requires that the original values of the limit and index registers be saved in the scratch area, and the "leave" procedures restore those values. MIMSY use of the registers is thus not obvious to the user task.

Hardware indirect-indexed addressing is implemented only in the post-index mode, while the interpreter requires (only) the pre-index mode. Indirect addressing is therefore not used, and operands are accessed by a double-index procedure:

LAC	2,X	/ get operand address.
PAX		/ move to index register.
LAC	0,X	/ get operand.
PLX		/ restore index register.

Memory locations within MIMSY are not modified except to use the EAE multiply and divide instructions which require in-line operands, and during the "enter" and "leave" operations. These procedures are protected by temporarily disabling another entry using the API levels (see entry procedure, below) or the 'DBI' instruction:

LAC	12,X	/ get multiplier.
DBI		/ disable interrupts.
DAC	+.4	/ store operand.
LAC	6,X	/ get multiplicand.
EBI		/ re-enable interrupts.
MUL		/ multiply.
O		/ protected operand.
ANY		/ interrupt may occur here.

MIMSY internal subroutine linkage simulates register linkage by using the accumulator to pass return addresses to the subroutines. Arguments are passed in the MQ register or in a scratch location, and the return addresses are temporarily stored in scratch locations. Subroutine returns are made using indexed jumps:

	LAC	(.+2&7777)	/ return address to AC.
	JMP	SUBR	/ jump to subroutine.
	PLX		/ restore index register.
SUBR	DAC	25,X	/ save return address.
	ANY		/ do something.
	LAC	25,X	/ get return address.
	PAX		/ move to index register.
	JMP	0,X	/ return.

External subroutine linkage is required during execution of the 'READ' and 'WRITE' instructions which employ user-written character-handling subroutines. It is required that each task provide separate routines to link MIMSY to the input and output devices or handlers, and the 'JMS' instruction is used to access those routines. The user routines must preserve or restore the PDP-15 limit register to assure proper operation, and they should not modify any location in the scratch area. As an example, the MIMSY procedure to pass a character to a user output routine is:

LAC	30,X	/ get address of output routine.
PAX		/ move to index register.
LACQ		/ get character.
AND	(177)	/ mask to seven bits.
JMS	0,X	/ call user subroutine.
PLX		/ restore index register.

2. USER TASK SCRATCH AREA

To implement re-entrant operation, each user task must reserve a separate 408-word memory area for MIMSY pseudo-registers, control parameters, and scratch usage. The scratch area contains the floating accumulator, operand register, the operand reference mode indicator, the looping and output format control words, and scratch locations needed by the interpreter. The scratch area must be increased to 548 locations if the matrix operations are used; a detailed discussion is presented in section VII. The floating accumulator, operand mode control word, and output format words may be initialized or modified using "normal mode" instructions. The index and limit registers are saved in the scratch area by the entry procedure (see below) for restoration on return to normal mode. The (saved) pseudo-index register is used by the interpreter to determine operand addresses in indexed instructions; it may be modified in interpretive mode by the 'INCR' instruction to employ sequential indexed data reference. The format of the task scratch area is:

<u>word</u>	<u>use</u>
0	index register save.
1	limit register save.
2	interpretive instruction address (offset).
3	operand reference mode.
4-7	floating accumulator (refer to section II).
10-13	floating operand register.
14-17	temporary operand register.
20	loop counter (negative).
21	loop return address (offset).
22	output format integer field width.
23	output format fraction field width.
24-27	internal subroutine linkage.
30	operand address (offset).
31-37	input/output and function scratch.
40-53	matrix arithmetic usage (see section VII).

3. INTERPRETER ENTRY PROCEDURE

The main body of the interpreter is augmented by an entry procedure peculiar to the system in which it operates. The most important functions of the entry procedure are to link the interpreter to the appropriate scratch area and to set the initial value of the interpretive instruction address pointer (floating program counter). If indexed addressing is to be used in interpretive mode, the entry procedure must also store the value (on entry) of the index register in the scratch area. Because the interpreter uses both the index and limit registers, they are restored from the scratch area by the "leave" procedures; both registers should be stored by the entry procedure. Other entry operations may include saving the accumulator and testing for a "float" operation, and testing for other special entry instructions.

Modification of some "common" core memory locations will be necessary in most practical entry procedures; such procedures must be protected from overlapping entries by temporarily disabling interrupts or by operating at an API level adequately high to lock out competing entries.

The entry procedure should store the values, at entry, of the index and limit registers in locations 0 and 1 of the task scratch area.

The entry procedure must set both the index and limit registers to contain the page offset adjusted absolute address of the scratch area. For example, if the task scratch area begins in location 10040, and the interpreter is in page 3, the registers will contain '760040'.

The entry procedure must store the page offset adjusted absolute address, less one, of the first interpretive mode instruction to be executed in location 2 of the scratch area. For example, if the first interpretive mode instruction is in location 20040, and the interpreter is in page 1, word 2 of the scratch area will contain '10037'.

The entry procedure may temporarily save the accumulator and/or the MQ register for special entry operations such as the "float-and-enter" operation.

The requirements are summarized in Appendix B, and an example entry procedure is presented. The example illustrates direct entry (EFM) and implementation of a single-precision signed float-and-enter (FLO) instruction, and entry protection at API level 4 by initiation using the 'CAL' instruction.

IV. INTERPRETIVE MODE INSTRUCTIONS

MIMSY interpretive mode instructions have the same form as the PDP-15 normal mode (hardware implemented) instructions. The memory reference (operand and address reference) class includes all operation codes from 0000₂ through 1110₂. The operate class includes register modification, immediate operand, and function evaluation instructions. The interpreter recognizes direct, indirect, and indexed addressing modes, calculating effective addresses compatible with PDP-15 page (index) mode hardware operation.

MIMSY does not use the auto-increment registers and does not perform the auto-increment operation when these registers are referenced. Although extensive use is made of the index and limit registers, these are restored to their original values on leaving interpretive mode.

The MIMSY instruction set is described below. Detailed discussions of the function evaluation, input/output and matrix operations appear in later sections. Operation codes and execution times are summarized in Appendix A. The instruction mnemonics used below are assigned for convenience in presentation; they are not defined in MIMSY, but are chosen to be compatible with PDP-15 conventions where feasible.

1. ENTER AND LEAVE INSTRUCTIONS

Because entry to interpretive mode is a normal (hardware) mode operation, entry instructions are not defined in MIMSY. The required entry protocol is discussed in section III, and an example is presented in Appendix B. There are two instructions causing a return to normal mode: a "quick" return, and a signed double-precision "fix" operation.

LFM 740000

Leave Floating Mode: The hardware index and limit registers are restored from the user scratch area, the AC and MQ are cleared, and return is made in normal mode at the following instruction. The user FAC is not disturbed.

FIX 740NNN

Fix and Leave: Operation is the same as 'LFM' except that the value in the FAC is converted to a rounded, signed, two's-complement double-precision integer in the AC (high order) and MQ (low order). If the magnitude of the FAC is less than 1/2 or greater than 2^{35} , the AC and MQ are cleared. The FAC is not disturbed.

2. LOAD AND STORE OPERATIONS

MODE 74700N

Set Operand Mode: The operand reference mode (discussed in section II) is set to 0, 1, 2, or 3 as indicated by the low order two bits of the instruction word. The operand mode remains in effect until another 'MODE' instruction is encountered. The content of the FAC is not affected.

LAC 20XXXX

Load FAC from Memory: The variable at the effective address (EA) is loaded into the user floating accumulator according to the current operand mode: EA if mode zero or one, EA and EA+1 if mode two, or EA through EA+2 if mode three. The FAC is then normalized if necessary.

LAW 76NNNN

Load FAC Immediate: The low 13 bits of the instruction word are taken as a two's-complement signed integer and "floated" to the floating accumulator. Constants in the range -4096 to +4095 may be generated directly in the FAC.

DAC 04XXXX

Deposit FAC in Memory: The number in the floating accumulator is moved to the 1, 2, or 3 memory locations specified by the effective address, converted to the appropriate operand format. The floating accumulator is not modified. In 'DATA' mode (mode zero) negative values will be stored as zero. In 'DATA' and 'INTEGER' modes: if the FAC magnitude exceeds 2^{35} , zero will be stored; if the magnitude exceeds the range of the operand format ($2^{18}-1$ or $2^{17}-1$) only the low order bits are stored. No test for valid exponent truncation is made for 'REAL' (mode two) operands.

3. FLOATING ACCUMULATOR MODIFICATION

CLA 750000

Clear Floating Accumulator: The floating accumulator magnitude is set to zero and the sign is set positive, i.e., all four words of the FAC are set to zero.

SSP 742000

Set Sign Positive: The sign of the FAC is forced positive, the magnitude is not affected.

CHS 743000

Change FAC Sign: The sign of the FAC is complemented. If positive, it is made negative, or if negative, it is forced positive. The FAC magnitude is not affected.

SGN 744000

Convert to Sign: The magnitude of the FAC is forced to unity, the sign is unchanged. Positive values, including zero, are converted to +1, negative values to -1.

4. ARITHMETIC INSTRUCTIONS

ADD 30XXXX

Add Memory Operand to FAC: The memory operand at EA, or EA and EA+1, or EA through EA+2, according to the current operand reference mode, is added to the number in the floating accumulator. The sum is normalized and left in the FAC. No tests for exponent overflow or underflow are made.

ADI 752NNN

Add Immediate Operand to FAC: The low nine bits of the instruction word are taken as a two's-complement signed integer and "floated" to the operand register, then added to the number in the FAC. The sum is normalized and left in the FAC. Immediate operands may have values from -256 through +255. If the operand is zero, only the FAC normalization operation is performed, providing a direct "double-precision float" procedure.

SUB 34XXXX

Subtract Memory Operand from FAC: The memory operand is subtracted from the number in the floating accumulator. The difference is normalized and left in the floating accumulator. No tests for overflow or underflow are made.

MPY 40XXXX

Multiply FAC by Memory Operand: The number in the floating accumulator is multiplied by the variable at the effective address. The normalized product is left in the FAC. No tests are made for overflow or underflow.

MPI 751NNN

Multiply by Immediate Operand: The low nine bits of the instruction word, a two's-complement signed integer in the range -256 through +255, multiply the number in the FAC. The normalized product is left in the FAC.

DVD 44XXXX

Divide FAC by Memory Operand: The number in the floating accumulator is divided by the operand specified by the effective address. The normalized quotient is left in the floating accumulator. If either the dividend or the divisor is zero, no divide occurs, and the FAC is set to zero.

DVI 753NNN

Divide by Immediate Operand: The low nine bits of the instruction word, a two's-complement signed integer in the range -256 through +255, divide the number in the floating accumulator. The normalized quotient is left in the FAC. If either the FAC or the operand is zero, the result is forced to zero.

RDV 24XXXX

Divide Memory Operand by FAC: The memory operand at the effective address is divided by the number in the FAC. The normalized quotient is left in the floating accumulator; the operand is not modified. If either the dividend or the divisor is zero, the result is forced to zero.

5. PROGRAM CONTROL INSTRUCTIONS

JMP 60XXXX

Jump to Location: Program control is transferred to the location specified by the effective address. Interpretive mode remains in effect. The floating accumulator is not modified.

JMS 10XXXX

Jump to Subroutine: The address of the 'JMS' instruction, incremented by one, is stored in the location specified by the effective address (EA); program control is transferred to EA+1. Interpretive mode remains in effect. The floating accumulator is not modified.

CAS 54XXXX

Compare FAC to Storage: The number in the floating accumulator is algebraically compared to the one-, two-, or three-word operand at the effective address. Program control is transferred to the first, second, or third location following the 'CAS' instruction depending on whether the FAC is less than, equal to, or greater than the operand. The floating accumulator is not modified.

Example:

LAC*	A,X	
CAS*	B,X	
JMP	LESS	/ 'A' < 'B'
JMP	EQUAL	/ 'A' = 'B'
DAC*	B,X	/ 'A' > 'B'

CAI 745NNN

Compare FAC Immediate: The number in the floating accumulator is compared to the value of the (low nine bits of the instruction word) signed immediate operand. Program control is transferred to the first, second, or third location following the 'CAI' instruction in the same way as for the 'CAS' instruction. The floating accumulator is not modified. If the immediate operand is zero (745000), the instruction is a negative-zero-positive test.

BEG 70NNNN

Begin a Program Loop: The loop counter is loaded with the two's complement of a number specified by the address field of the instruction. If the address is not modified, the loop count is taken directly from the low twelve bits. If the instruction is indirect and/or indexed, the (one-word) value at the effective address is the loop count. The address field may not be empty (all zeroes). The FAC is not modified.

END 700000

End of Program Loop: The loop counter set by the 'BEG' instruction is incremented. If it becomes zero, program control is transferred, in interpretive mode, to the instruction following 'END'. If the loop counter is not incremented to zero, program control is transferred to the location immediately following the last encountered 'BEG' instruction. 'END' must have an empty address field; if the entire instruction address field is not zero, it is a 'BEG' instruction.

'BEG, ..., END' loops may not be nested; MIMSY contains only one loop counter. Several 'END' instructions, however, may be associated with any 'BEG' instruction.

INCR 00XXXX

Increment Word at Address: The single-word value at the effective address is incremented by the current operand "size", i.e., by 1, 1, 2 or 3 for operand reference modes 0, 1, 2 or 3, respectively. 'INCR' is used for operand pointer control in program loops. If the address is the first word of the user scratch area, the "index register" is modified.

6. INPUT AND OUTPUT INSTRUCTIONS

The input and output operations and instructions are explained in detail in section V; summary explanations are presented here.

FMT 64XXXX

Set Output Format: The low twelve bits of the effective address are loaded to the output format control words in the user scratch area. Bits 12-17 specify the width of the "fraction" field, including the decimal point, and bits 7-11 specify the width of the "integer" field preceding the decimal point (if present). Bit 6 specifies the 'E' format (scientific floating point) option.

WRITE 50XXXX

Write Number from FAC: The value of the floating accumulator is converted to an ASCII character string according to the current output format and transmitted to a user character-handling subroutine at the effective address, one character at a time. The user routine must not modify, or must restore, the hardware limit register. 'WRITE' does not modify the floating accumulator.

READ 14XXXX

Read Number to FAC: A free-format number is read to the floating accumulator by requesting ASCII characters from a user subroutine at the effective address. Characters are requested, one at a time, until a terminator (refer to section V) is encountered. The user routine must not modify the hardware limit register.

7. FUNCTION EVALUATION

The function evaluation operations are summarized below; detailed discussion is presented in section VI.

SQR 746000

Square Root: The square root of the absolute value of the floating accumulator is evaluated and left in the FAC.

SQA 741000

Square of FAC: The number in the floating accumulator multiplies itself.

EXP 754000

Natural Exponential of FAC: The natural exponential, e^x , of the number in the floating accumulator is evaluated. If the value is beyond the capacity of the FAC, the result is forced to zero.

LOG 754001

Natural Logarithm: The natural logarithm of the absolute value of the floating accumulator is evaluated. If the FAC contains zero, the result is forced to zero.

COS 754002

Cosine of the FAC: The cosine of the value in the floating accumulator, in radians, is evaluated. Precision decreases for large values.

SIN 754003

Sine of the FAC: The sine of the value in the FAC, in radians, is evaluated. Precision decreases for large values.

ATN 754004

Arctangent of the FAC: The principle value arctangent, in radians, in the range $\pm \pi/2$, is evaluated.

8. MATRIX OPERATIONS

The MIMSY operations on upper-triangular symmetric matrices are summarized below; a more detailed discussion is presented in section VII.

SCLVCR 755000

Build Scaling Vector: The square roots of the diagonal elements of the matrix are written to the scaling vector for use by the 'NORMLZ' operation. If a diagonal element is zero, the corresponding scale vector element is set to one.

NORMLZ 755001

Normalize (Scale) the Matrix: The elements of the matrix are scaled, to produce values of one in all of the diagonal elements, by the scaling vector. Prior use of 'SCLVCR' is assumed. The object is to form a "well-conditioned" matrix for inversion.

MLTPLY 755002

Multiply Vector by Matrix: The 'B' vector is multiplied, from the left, by the matrix to produce the 'P' vector.

INVERT 755003

Invert Matrix: The (upper triangular) symmetric matrix is inverted in place. If singular, control is transferred, in normal mode, to the location specified in word 47₈ of the user scratch area.

V. INPUT AND OUTPUT

MIMSY includes a free-format input interpreter which reads ASCII character strings to numeric values in the floating accumulator, and a variable-format output controller to convert numeric data to ASCII representation. Output data may be presented in integer, fixed-point, or floating-point formats. The input and output controllers receive and transmit data, one character at a time, through user-written character-handling subroutines. The entry point to the user input (output) routine is indicated by the effective address of the input (output) instruction. To preserve MIMSY re-entrancy, there must be separate user routines for each task, and the routines must preserve the contents of the PDP-15 limit register.

1. THE INPUT INTERPRETER

The 'READ' instruction (14XXXX) causes the input interpreter to request a series of characters, until a numeric terminator is encountered, by calling (JMS) the user-written subroutine at the instruction effective address, and to construct the appropriate numeric value in the floating accumulator. The user subroutine must return one seven- or eight-level ASCII character to the interpreter, in the accumulator, each time it is entered. The input interpreter masks all characters to the seven-bit representation.

The input character string has the form: "SXXX.YYYESZZT", where "S" is a sign (+ or -), "." is the decimal point, "E" is the character representing decimal exponentiation, and "T" is a terminator; "X", "Y" and "Z" are decimal digits. Any of the above fields is optional, and input may have any of the forms:

1., 1, 2.5, .3, -2.3, -1E10, -1E-10, 3.14E6

The only limitation is that the mantissa (XXXYYY) should not contain more than ten significant digits, i.e., it must be smaller in magnitude than 2^{36} .

For MIMSY numeric input, there are four classes of characters: numeric characters, characters which are always ignored, conditionally significant characters, and terminators. The numeric characters are:

0-9	60-71	decimal digits
E	105	exponent indicator (10)
.	56	decimal point

The characters always ignored are:

null	0
rub-out	177
line feed	12

The conditional characters are:

blank	40	
tab	11	
+	53	plus sign
-	55	minus sign

Blank, tab, and the plus sign (+) are ignored if they precede numeric characters; following numeric characters they act as terminators. The minus sign (-) indicates negation when preceding numeric input, and termination following numeric input. Any character not described above is a terminator.

If line erasure and character rub-out operations, typically initiated by 'CNTRL-U' and 'RUBOUT' characters, are needed, they must be implemented in the user character input routine or in the monitor operating system.

2. THE OUTPUT CONTROLLER

The 'WRITE' instruction (50XXXX) causes the output controller to transmit a series of characters, representing the value in the floating accumulator, to the user-written character-handling subroutine at the instruction effective address. For each character required to satisfy the current output format (see below), the controller calls (JMS) the subroutine with the (seven-level) ASCII character in the accumulator.

To preserve re-entrancy, there must be a separate character output subroutine for each user task; the subroutines must preserve the contents of the limit register.

The floating accumulator is restored to its original value at completion of the output procedure.

3. OUTPUT FORMAT

Numeric output may be presented in any of three formats: integer (I), fixed-point (F), or floating-point (E). The format is specified by the low twelve bits of the effective address of the 'FORMAT' instruction (64XXXX). The low six bits (12-17) specify the number of characters in the "fractional" part of the character string, the decimal point and all following digits. Bits 7-11 specify the number of spaces to be allowed for the "integer" part of the representation, the number of spaces preceding (if present) the decimal point. If bit 6 is set, floating-point (E) format is specified, and four additional characters, following the "fraction" field, are generated.

The specified format is not dependent on either the value of the number to be represented or the current operand reference mode; the format indicates only the required external alphanumeric representation. However, if the value of the number is too large to be represented in the currently specified format: if the format is integer or fixed-point, the transmitted character field consists of one blank followed by enough asterisks (*) to fill the field; if the format is floating-point and the value exceeds 10^{100} , the digits in the 'E' field will be invalid.

The output controller rounds all values in the last displayed digit. Because MIMSY arithmetic accuracy is limited to 10 to 11 decimal digits, the rounding may not be apparent if more than ten significant digits are required in the format.

INTEGER and FIXED-POINT formats are basically the same. Since bits 12-17 specify the width of the "fraction" field, including the decimal point, if these bits are zero, the format is integer. If the value in bits 12-17 of the format control word is one, the decimal point is the last transmitted character. If the value is two or more, one or more digits will be transmitted following the decimal point. The total field width for these formats is the sum of the value in 12-17 and the value in bits 7-11. The width of the "integer" part of the field should be large enough for all expected digits, the sign (which will be transmitted as a blank for positive numbers), and the desired number of leading blanks.

FLOATING-POINT format is specified by setting bit 6 of the format control word. Numbers are represented with one digit preceding the "fraction" field (if present), and a following 'E' field to indicate a multiplicative power of ten. The 'E' field consists of the character 'E', a sign, and a two-digit decimal number. The "integer" field width must be large enough (two spaces) for the leading digit and a sign, but serves primarily to specify the number of leading blanks. The "fraction" field is as described above, but, in floating-point format, it determines the precision of the number representation. The total field width is the sum of the "integer" field width and the "fraction" field width plus four.

Example: consider the number 13.265 in several formats:

<u>format word</u>	<u>external appearance</u>
0600	13
0601	13.
0602	13.3
0603	13.27
4603	1.33E+01
4604	1.327E+01

VI. FUNCTION EVALUATION

MIMSY functional instructions include square, square root, natural logarithm and exponential, and sine, cosine, and arctangent. The functions operate on the floating accumulator, returning the appropriate value to the floating accumulator. The accuracy is 34 to 35 bits (10 decimal digits) independent of the current operand reference mode. Where series expansions are used, the number of terms is variable to achieve the required accuracy in the minimum time.

A brief description of each algorithm is given below. The execution times are averages (see Appendix A) and may not be representative of any specific application.

1. SQUARE (SQA) 741000 152 microseconds

The value of the floating accumulator is copied to the operand register, and the multiply routine is entered to produce the square of the value.

2. SQUARE ROOT (SQR) 746000 180 microseconds

The square root of the absolute value of the number in the floating accumulator is evaluated by reducing the argument to a number between 1/2 and 2, then performing a Newton iteration. The reduction is made by removing the even component of the binary exponent which is divided by two for later addition to the exponent of the intermediate result. A first approximation of .928862 is made, and the Newton iteration:

$$R_{i+1} = (R_i + \text{ARG}/R_i)/2 ,$$

is performed three times on the high mantissa, giving 17 bits of precision. A fourth iteration on the entire mantissa extends the precision to at least 34 bits. The exponent is then augmented by half the value of the original exponent to complete the evaluation. The result is normalized in the floating accumulator.

3. LOGARITHM (LOG) 754001 3.67 milliseconds

The natural logarithm of the absolute value of the number in the floating accumulator is evaluated and left in the floating accumulator. If the argument is zero, the result is forced to zero. The function is evaluated by reducing the argument, X, by:

$$Z = (\sqrt{2} \cdot X - 2^I) / (\sqrt{2} \cdot X + 2^I) ,$$

where I is the original binary exponent of X, then evaluating:

$$\log(X) = \frac{2I-1}{2} \cdot \log(2) + 2 \cdot Z \left(1 + \frac{Z^2}{3} \left(1 + \frac{3Z^2}{5} \left(1 + \dots \frac{7Z^2}{9} \left(1 + \frac{9Z^2}{11} \right) \right) \right) \right) .$$

The first term ignored is $Z^{13}/13$ which is, at most, of the order 2^{-36} .

4. EXPONENTIAL (EXP) 754000 3.42 milliseconds

The natural exponent of the number in the floating accumulator is evaluated by conversion to a two's exponent:

$$e^x = 2^{a \cdot x}; \quad a = \log_2(e)$$

The argument for the two's exponent is then "fixed" with rounding to produce integral and fractional exponentials:

$$2^{a \cdot x} = 2^{I+f} = 2^I \cdot 2^f = 2^I e^{f/a}$$

The integer argument is saved for later adjustment of the result and the fractional part is evaluated with the series:

$$2^f = e^{f/a} = e^Z = 1 + Z + \frac{Z^2}{2} + \frac{Z^3}{3} + \dots + \frac{Z^N}{N} + \dots$$

The number of terms used in the series is determined by the value of the binary exponent of 'Z' (K), which is negative nonzero:
 $N = 49/(4-K)$; or $N \geq 2$.

5. SINE (SIN) 754003 3.87 milliseconds

The sine function evaluation requires the angular value in the floating accumulator to be in radians. The sine is evaluated by reducing the argument, by truncation, to the first quadrant, saving the original sign and quadrant for later sign and magnitude adjustment. The truncation process can cause a loss of precision for large arguments. The basic computation is performed by evaluating the series:

$$\sin(X) = X(1 - X^2(1 - \frac{X^2}{6}(1 - \frac{X^2}{20}(1 - \dots - \frac{X^2}{N(N+1)}(1 - \dots)))$$

where 'N' is determined by the binary exponent of the truncated value 'X'; N is 16 for positive, 14 for zero, and 12 for negative values of the exponent.

6. COSINE (COS) 754002 3.97 milliseconds

The cosine of the value in the floating accumulator, assumed in radian measure, is evaluated using the 'SINE' function:

$$\cos(X) = \sin(X + \pi/2)$$

7. ARCTANGENT (ATN) 754004 3.41 milliseconds

The principle value arctangent of the number in the floating accumulator, in radians in the range $\pm\pi/2$, is evaluated. If the argument is greater than one, the value is calculated as

$\pi/2 - \text{atn}(1/X)$. If the argument (or the inverse) is then greater than $\sqrt{2}-1$, the further reduction:

$$\text{atn}(X) \leftarrow \pi/8 + \text{atn}((X - \tan(\pi/8)) / (1 + X \cdot \tan(\pi/8))) ,$$

is made. The primary computation consists in evaluating the (terminated) continued fraction:

$$\text{atn}(X) = X / (1 + X^2 / (3 + 4X^2 / (5 + \dots / ((2N-1) + N^2 X^2 / (2N+1)))) ,$$

where 'N' is calculated from the binary exponent of the reduced argument by $N = (K+11)/2+2$.

VII. MATRIX OPERATIONS

Problems involving real symmetric matrices are frequently encountered in laboratory computer applications. MIMSY implements four operations, including normalization, inversion, and vector multiplications, to simplify the solution of these problems. The operations are applicable to problems of the type: $A_{ij}x_j = b_i$, with solution: $x_i = (A^{-1})_{ij}b_j$, where $A_{ij} = A_{ji}$. The matrix inversion algorithm is essentially that of Rutishauser* for the in-place inversion of real symmetric upper-triangular-stored matrices.

1. THE MATRIX OPERATIONS

The MIMSY matrix operations are described below; the details of matrix and vector storage and format, and augmentation of the user task scratch area are described following.

SCALING VECTOR CONSTRUCTION

The instruction 'SCLVCR' (755000) causes the scaling vector (S) to be computed from the diagonal elements of the matrix (A) for use in the matrix normalization operation. If a matrix diagonal element is zero, the corresponding scale vector entry is set to one:

$$S_i \leftarrow 1/\sqrt{A_{ii}} \quad .$$

MATRIX NORMALIZATION

It is often desirable to assure a well-conditioned matrix before attempting the inversion. The instruction 'NORMLZ' (755001) multiplies the matrix (A) on the left and the right by the scaling vector (S), as a diagonal matrix, to scale all of the diagonal elements to unit magnitude:

$$A_{ij} \leftarrow S_i A_{ij} S_j \quad .$$

MATRIX INVERSION

The instruction 'INVERT' (755003) causes the matrix (A) to be inverted in place. If the matrix is singular, control is transferred, in normal mode, to the location specified in word 47₈ of the user scratch area. The inversion operation will modify the contents of the P, Q, and R vectors; the B and S vectors are not disturbed:

$$A_{ij} \leftarrow (A^{-1})_{ij} \quad .$$

* Rutishauser, Communications of the ACM, algorithm 150, vol. 6, no. 2.

VECTOR MULTIPLICATION

The instruction 'MLTPLY' (755002) causes the column vector B to be multiplied on the left by the matrix; the product appears in the vector P:

$$P_i \leftarrow A_{ij} B_j$$

2. MATRIX AND VECTOR FORMAT

The elements of the matrix (A) and four of the five vectors (B, P, Q and S) must be in either floating-point (mode 2) or extended precision (mode 3) operand format. If the operand reference mode is not two or three, any matrix operation will be aborted through the "singular matrix" return. The scratch vector R requires only one word for each entry. If the problem has dimension "N", and the operand mode is "M", the number of words required for matrix and vector storage is:

<u>quantity</u>	<u>words required</u>
A matrix	$M*N*(N+1)/2$
B vector	$M*N$
P vector	$M*N$
Q vector	$M*N$
S vector	$M*N$
R vector	N

The matrix is stored in upper triangular form, column by column:

<u>Matrix</u>	<u>Equivalent vector</u>
A ₁₁ A ₁₂ A ₁₃ A ₁₄ →	
A ₂₂ A ₂₃ A ₂₄	A ₁₁ A ₁₂ A ₂₂ A ₁₃ A ₂₃ A ₃₃ A ₁₄ →
A ₃₃ A ₃₄	
A ₄₄	
↓	

3. SCRATCH AREA REQUIREMENTS

The user task scratch area must be augmented from 408 to 548 locations to provide pointers to the matrix and vectors, the matrix dimension, an error return address, and additional scratch words:

wordcontents

40	address of matrix
41	address of vector B
42	address of vector P
43	address of vector Q
44	address of vector S
45	address of vector R
46	dimension of matrix
47	address of error return
50-53	scratch

APPENDIX A

MIMSY INSTRUCTION SUMMARY

The MIMSY instructions are summarized below in the format:

mnemonic	octal code typical usage	execution time description
----------	-----------------------------	-------------------------------

The mnemonics are not defined in the MIMSY program package, but are used for continuity and clarity in the presentations; they conform to PDP-15 conventional mnemonics where feasible. The octal operation codes are the numeric instructions recognized by the interpreter. Typical usage is shown for each instruction, and indicates only one of (often) several valid representations; memory reference instructions may be shown both indirect and indexed in addressing mode, but both modifiers are optional. The descriptions are brief; the full descriptions are given in section IV.

The execution times are given in microseconds. They are measured for 10,000 operations each on random numbers in the range ± 256 for all operations except SIN, COS, and ATN, where the range is ± 2 . Times for the matrix operations were measured using random matrix elements in the broader range. The range makes no significant difference in the measured times, except for the arctangent evaluation. All times were measured on a standard PDP-15/20 using extended precision operands.

ENTER AND LEAVE

EFM	(refer to IV-1) EFM SCR	(refer to 'LFM', below) Enter Floating Mode (Pointer to Scratch Area)
FLO	(refer to IV-1) FLO SCR	(refer to 'FIX', below) Float and Enter (Pointer to Scratch Area)
LFM	740000 LFM	100 (including example EFM) Leave Floating Mode
FIX	740NNN FIX	175 (including example FLO) Fix and Leave

LOAD AND STORE

MODE	74700N MODE 3	33 Set Operand Mode
LAC	20XXXX LAC* A,X	107 Load FAC from Memory
LAW	76NNNN LAW 12	55 Load FAC Immediate
DAC	04XXXX DAC* A,X	93 Deposit FAC in Memory

FAC MODIFICATION

CLA	750000 CLA	35 Clear Floating Accumulator
SSP	742000 SSP	28 Set Sign Positive
CHS	743000 CHS	30 Change FAC Sign
SGN	744000 SGN	38 Set Magnitude to One

ARITHMETIC

ADD	30XXXX ADD* A,X	173 Add Memory to FAC
SUB	34XXXX SUB* A,X	180 Subtract Memory from FAC
ADI	752NNN ADI 144	162 Add Immediate to FAC
MPY	40XXXX MPY* A,X	207 Multiply FAC by Memory
MPI	751NNN MPI -5	197 Multiply Immediate
DVD	44XXXX DVD* A,X	205 Divide FAC by Memory
DVI	753NNN DVI 74	195 Divide by Immediate
RDV	24XXXX RDV* A,X	217 Divide Memory by FAC

PROGRAM CONTROL

JMP	60XXXX JMP* L	43 Jump to Location
JMS	10XXXX JMS S	53 Jump to Subroutine
CAS	54XXXX CAS* A,X	115 Compare FAC to Memory
CAI	745NNN CAI 0	103 Compare Immediate
BEG	70NNNN BEG* N	57 Begin Loop
END	700000 END	47 End Loop
INCR	00XXXX INCR L	55 Add Mode to Word

INPUT AND OUTPUT

FMT	64XXXX FMT 1207	60 Set Output Format
READ	14XXXX READ UIR	(indeterminate) Read Input Data to FAC
WRITE	50XXXX WRITE UOR	(indeterminate) Write Data from FAC

FUNCTIONS

SQR	746000 SQR	180 Square Root of FAC
SQA	741000 SQA	152 Square of FAC
EXP	754000 EXP	3420 Natural Exponent
LOG	754001 LOG	3667 Natural Logarithm
COS	754002 COS	3968 Cosine of FAC (radians)

SIN	754003	3867
	SIN	Sine of FAC (radians)
ATN	754004	3407
	ATN	Principle Value Arctangent

MATRIX ARITHMETIC

Times shown are for 6X6 matrices.

SCLVCR	755000	2007
	SCLVCR	Build Scaling Vector
NORMLZ	755001	10428
	NORMLZ	Normalize Matrix
MLTPLY	755002	15405
	MLTPLY	Multiply 'B' Vector by Matrix
INVERT	755003	74040
	INVERT	Invert Matrix

Matrix Operation times for random symmetric matrices of dimensions 3, 6, 9 and 12, in microseconds, are:

<u>Operation</u>	<u>Dimension</u>			
	<u>3</u>	<u>6</u>	<u>9</u>	<u>12</u>
SCLVCR	1025	2007	2985	3980
NORMLZ	3130	10428	21950	37820
MLTPLY	3830	15405	34623	61550
INVERT	12830	74040	218830	480380

APPENDIX B

EXAMPLE FOR ENTRY PROTOCOL

MIMSY entry protocol requires:

- a. the contents, at entry, of the index register must be saved in location #0 of the user scratch area.
- b. the contents, at entry, of the limit register must be saved in location #1 of the user scratch area.
- c. the address of the user scratch area, adjusted for page offset, must be saved in both the index and limit registers.
- d. the interpretive-mode instruction address location, decremented by one and adjusted for page offset, must be stored in location #2 of the scratch area.
- e. if a single-word "float" operation is required, the contents of the AC, at entry, should be saved in the MQ.
- f. because the above operations will require some use of memory locations, they must be protected against overlapping interpretive mode entry.

The following example is used in a system in which entry to interpretive mode is allowed from API software levels (4 through 8). Protection is implemented by using the 'CAL' instruction to execute the prologue at API level #4. A 'DBK' instruction is executed after all necessary use of memory locations is finished, and the interpreter operates at the user API level (#4 excepted).

The indirect 'CAL' is used for direct entry to avoid the time penalty of a typical 'CAL' processor. The "signed single-precision float" instruction is implemented by setting flag bits in the 'CAL' instruction.

```

/   EFM=020000
/   FLO=020007
      .LOC      10000
/
/   BEGIN PROLOGUE
FME      0      / entry point from indirect 'CAL'.
      LMQ      / save accumulator for 'FLOAT'.
/
      PXA      / temporarily save index register
      DAC      .ADR / in a scratch location.
/
HMSK     770000 / get user scratch area address
      TAD*     FME / (with offset) in index register.
      PAX
/
      LAC      .ADR / pick up original XR value and
      DAC      0,X / save in scratch word #0.
      PLA      / get limit register and save
      DAC      1,X / in scratch word #1.
      PXL      / back up scratch address in LR.
/
      LAC      0 / get call address (+1) to establish
      DBK      / floating mode program counter
      AND      (77777) / (with offset) in scratch area
      TAD      HMSK / word #2, and release entry
      DAC      2,X / protection with 'DBK'.
/
      AAC      -1 / pick up 'CAL' instruction word
      PAX      / and test low bits for 'FLOAT'
      LAC      0,X / instruction.
      AND      (7)
      SZA
      JMP      .FLOD+2 / go to 'FLOAT' entry.
/
/   END OF PROLOGUE
/
.BAK     PLX      / normal entry.
      ISZ      2,X / increment FPC.
.EIM     LAC      2,X / pick up FPC.
      PAX      / move to XR.
      LAC      0,X / pick up instruction.
      PLX      / restore XR.
      .
      .
      .

```

APPENDIX C

PROGRAMMING EXAMPLES

Because MIMSY usage is so like normal PDP-15 assembly language coding, it is difficult to provide nontrivial examples that are both useful and reasonably brief. The differences in using MIMSY are involved in the transfers between "normal" and "interpretive" modes, the use of a (separate) scratch area for each (simultaneously executing) task, and the proper usage of the several operand modes.

Three examples are presented below. The double-precision signed float subroutine illustrates both the specification of a task scratch area, and a procedure that is required in many real-time applications. The general exponentiation procedure illustrates the use of the compare instruction and the function evaluation operations. The third example illustrates the use of the MIMSY loop instructions, the matrix operations, and the augmented task scratch area.

1. DOUBLE-PRECISION SIGNED FLOAT

Since an 'ADD (0)' instruction will force a normalization of the FAC, it may be used to implement a double-precision "float" operation. The example is a subroutine to float a 36-bit two's-complement signed integer in the long register (AC high, MQ low) to the floating accumulator and return in interpretive mode.

DPSF	0		/ enter double signed float.
	DZM	ACS	/ set FAC positive.
	SMA CLL		/ test negative.
	JMP	POSI	/ no, store positive.
	CMA CLL		/ complement high order and
	DAC	ACH	/ store in high mantissa.
	ISZ	ACS	/ set FAC negative.
	LACQ		/ get low order, negate to
	CLL CIA		/ store in low mantissa.
	SZL CLL		/ if negation carry,
	ISZ	ACH	/ increment high mantissa.
	JMP	CONT	
POSI	DAC	ACH	/ store positive high order.
	LACQ		/ get low order and
CONT	DAC	ACL	/ store in low mantissa.
	LAC	(43)	/ set binary point to bottom
	DAC	ACX	/ of low mantissa.
	EFM		/ enter interpreter,
	SCR		/ using area 'SCR'.
	ADI	0	/ add zero to normalize.
	JMP*	DPSF	/ return in interpretive mode.
	.		
	.		
SCR	.BLOCK	3	/ task scratch area.
MOD	3		/ operand reference mode.
ACS	0		/ FAC sign.
ACX	0		/ FAC exponent.
ACH	0		/ FAC high mantissa.
ACL	0		/ FAC low mantissa.
	.BLOCK	30	/ more scratch area.
	.		
	.		

2. GENERAL EXPONENTIAL

A general exponential operation is not included in MIMSY, but is easily coded as a subroutine or an in-line procedure. General exponentiation is the process of raising any (positive non-zero) base, B, to any power, P; the algorithm is:

$$|B|^P = \exp(P * \log(|B|))$$

If the base were allowed to have negative values, the result would, in general, be complex; if B is zero, a special test is required. The MIMSY procedure for general exponentiation is:

LAC	B	/ get the base.
CAI	0	/ compare to zero.
JMP	.+2	/ log takes absolute value.
JMP	.+4	/ done if equal to zero.
LOG		/ get logarithm.
MPY	P	/ multiply by power.
EXP		/ get exponent.
ANY		/ done.

Integer powers and roots, B^N and $B^{1/N}$, are readily calculated by replacing the 'MPY P' instruction with 'MPI N' or 'DVI N'.

3. LEAST-SQUARES FITTING

To illustrate the use of the looping instructions and matrix operations, this example solves for the coefficients of a straight line fitting a set of points (x,y) such that the sum of the squares of the deviations is minimized:

$$\sum_{i=1}^N (a+bx_i-y_i)^2 \rightarrow 0$$

The solution for the "best" values for (a,b) is:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum 1 & \sum x \\ \sum x & \sum x^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum y \\ \sum xy \end{pmatrix}$$

The operand reference mode and matrix dimension are set in "normal" mode, and interpretive mode is entered using a 'FLOAT' instruction. The value of A(1,1) is just the number of points to be fit. The floating accumulator is cleared and stored in A(1,2)=A(2,1), A(2,2), B(1) and B(2) to initialize the sums. The 'BEG - END' loop is executed to perform the summations; the 'INCR' instructions increment the address pointers by two, corresponding to the operand reference mode. The matrix operations invert the matrix and perform the vector multiplication, leaving the solution in the 'P' vector: a=P(1), b=P(2). The scratch area is shown augmented from 408 to 548 locations to implement the matrix arithmetic.

	LAC	(2)	
	DAC	DIMEN	/ set dimension to two.
	DAC	OPMOD	/ set operand mode to two.
	LAC	NMPNT	/ number of (x,y) pairs.
	FLOAT		/ float number of points,
	SCR		/ use scratch area 'SCR'.
	DAC	A11	/ set A(1,1) to sum(1)=N.
	CLA		
	DAC	A12	/ initialize sums to zero.
	DAC	A22	
	DAC	B1	
	DAC	B2	
	BEG*	NMPNT	/ start loop on all pairs.
	LAC*	XPTR	
	ADD	A12	
	DAC	A12	/ A(1,2) = sum(x).
	LAC*	XPTR	
	SQA		
	ADD	A22	
	DAC	A22	/ A(2,2) = sum(x*x).
	LAC*	YPTR	
	ADD	B1	
	DAC	B1	/ B(1) = sum(y).
	LAC*	XPTR	
	MPY*	YPTR	
	ADD	B2	
	DAC	B2	/ B(2) = sum(x*y).
	INCR	XPTR	/ increment x pointer.
	INCR	YPTR	/ increment y pointer.
	END		/ mark end of loop.
	SCLVCR		/ build scale vector in 'S'.
	NORMLZ		/ scale the matrix.
	INVERT		/ invert the matrix.
	NORMLZ		/ rescale matrix.
	MLTPY		/ P(I) = A(I,J)*B(J).
	.		/ done.
	.		
SCR	.BLOCK	3	/ scratch area.
OPMOD	.BLOCK	35	/ operand mode.
	A11		/ matrix origin address.
	B1		/ pointer to 'B' vector.
	P1		/ pointer to 'P' vector.
	Q		/ pointer to 'Q' vector.
	S		/ pointer to 'S' vector.
	R		/ pointer to 'R' vector.
DIMEN	2		/ dimension of matrix.
	SING		/ pointer to error code.
	.BLOCK	4	/ more scratch.