214
5/4/72
AMCR-1039

DR 2125

UC-32

MASTER

FP9

FLOATING POINT ARITHMETIC SOFTWARE FOR
DEC PDP-9 AND PDP-15 COMPUTERS

W. R. MYERS

Aerojet Nuclear Company

NATIONAL REACTOR TESTING STATION

Idaho Falls, Idaho — 83401

DATE PUBLISHED - MARCH 1972

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 22151
Price: Printed Copy \$3.00; Microfiche \$0.95

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.

FP9

FLOATING POINT ARITHMETIC SOFTWARE FOR
DEC PDP-9 AND PDP-15 COMPUTERS

by

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 — March 1972

PREPARED FOR THE U. S. ATOMIC ENERGY COMMISSION
IDAHO OPERATIONS OFFICE
UNDER CONTRACT NO. AT(10-1)-1375

ACKNOWLEDGMENTS

The author is grateful to the many people who have used FP9 during its evolution for their advice and criticism, particularly to C. W. Richardson for suggesting the implementation of the integer operand modes, and D. D. Metcalf for the square root algorithm and an improvement of the divide algorithm.

ABSTRACT

FP9 is a small fast interpretive arithmetic software package for Digital Equipment Corporation PDP-9 and PDP-15 computers, equipped with the extended arithmetic option (EAE), with up to 32,768 words of core memory. Versions are available for operation in either bank or page addressing modes. The routines have been modified during several years' use to provide an effective compromise among size, speed, and number of instructions. The arithmetic section requires less than 700₁₀ words of memory to implement twenty-five floating point operations including float, fix, three-way compare, inverse divide, square, square root, and immediate add, multiply and divide, as well as the usual arithmetic operations. The full package, requiring less than 1300₁₀ locations, includes relatively fast subroutines for the trigonometric and natural functions, and monitor- or user-oriented free-format input and variable-format output routines.

TABLE OF CONTENTS

	<u>Page</u>
ACKNOWLEDGMENTS	ii
ABSTRACT	iii
I. INTRODUCTION	1
II. OPERAND MODES	3
1. Floating Accumulator (FAC) Format	3
2. Data	3
3. Integer	4
4. Floating Point	4
5. Extended Precision	4
III. INITIATING AND TERMINATING INTERPRETIVE MODE	5
IV. INTERPRETIVE MODE INSTRUCTIONS	7
V. FUNCTION SUBROUTINES	19
1. Arctangent	19
2. Sine and Cosine	19
3. Natural Logarithm	20
4. Exponential	20
5. Square Root	21
6. Extended Functions	21
VI. INPUT AND OUTPUT	25
1. The Input Controller	25
2. The Output Controller	29
VII. PROGRAMMING EXAMPLES	33
1. A Faster Exponential	34
2. Spectrum Conversion	36
3. A Sorting Algorithm	38
4. An Input/Output Example	43
APPENDICES	
A. INSTRUCTION SUMMARY	A-1
B. INSTRUCTION EXECUTION TIMES	B-1
C. FP9 PROGRAM SOURCE LISTING	C-1

I. INTRODUCTION

FP9 is a system of arithmetic and logical functions which provides the assembly language programmer the capability of coding floating point computations in a direct, natural manner. The package will operate in a Digital Equipment Corporation PDP-9 or PDP-15 computer equipped with the extended arithmetic element (EAE). Extensive use is made of the EAE instruction set to obtain a fast, small, yet versatile floating arithmetic capability. Versions are provided for operation with either bank or page mode addressing; the page mode version correctly handles indexed (hardware) as well as indirect addressing; neither version disturbs the index or limit registers, or uses the auto-increment registers.

The arithmetic-logical segment, which can be used independently of the rest of the package, requires less than 700₁₀ words of memory to implement four number formats and over twenty-five instructions. In addition to memory-reference load, store, add, subtract, multiply and divide, the instruction set includes inverse divide, float, fix, jump, jump to subroutine, two three-way compares, square, square root, a full complement of immediate-class instructions: load, add, multiply, divide, and several "operate-class" instructions.

Other segments include small relatively fast subroutines for exponential, logarithm, sine, cosine and arctangent. Two versions of the input and output controllers are provided: one version communicates through 'IOPS' ASCII buffers and is completely compatible with the KM9/15 system; the second version employs external user written character handling routines.

FP9 is intended primarily for use in dedicated applications, and, though it is well suited, not as a general purpose computational tool. In the interest of execution speed and program size, then, there are no means provided for detection of arithmetic overflow or underflow, and the functions return "reasonable" results for "nonsense" arguments (e.g., $\log(|X|)$, $\sqrt{|X|}$). 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 only two exceptions: the "deposit floating accumulator" instruction (DAC) will store 'zero' rather than a negative number in a DATA operand (operand mode zero), and the output controller will send a full field of 'periods' to the output routine if a number is too large for the field width specified in the format indicator.

Floating point (interpretive mode) coding is written in exactly the same way as the coding for normal hardware-implemented operations. The FP9 control routine picks up single-word instructions, separates them into operation codes and operand designators, and performs the required operation. Thus, the operation "add X to the floating accumulator" is written "ADD X". Twenty to thirty machine cycles (approximately 10% of a typical instruction time) are required for this interpretive operation; this cost in time is offset by the saving of approximately 40% in application program size compared to the faster two-word operation sequence (JMS* (ADD), .DSA X).

FP9 consists of several program segments which may be included as desired for each application. "F9" or "F15" and "F1" must be included in each assembly. The segments are:

<u>SEGMENT</u>	<u>SIZE</u> (octal)	<u>FUNCTION</u>
F9	23	bank-mode interpreter prologue
F15	36	page-mode interpreter prologue
F1	1233	arithmetic and logic
F2	224	trigonometric functions
F3	147	logarithm and exponent
F4U	243	} output controller, one only
F4M	321	
F5U	246	} input controller, one only
F5M	310	
F6	201	extended functions

The desired segments must be combined prior to assembly; using "PIP", for example: >T DR F SRC (AGW) ← DTO F9 SRC,F1 SRC,F2 SRC,F4U SRC). Each segment includes ".GLOBL's" for internally defined global variables, which may be deleted for absolute assemblies. The user must insert .ABS and .LOC in the combined source file for absolute assembly. The relocation-mode control pseudo-op ".DBREL" is included in 'F15'; ".EBREL" must be inserted in 'F9' for use with the PDP-15 assemblers.

There are currently three unused operation codes ("AND", "DZM" and "EAE") in the operand reference class, and four (754000 through 757000) in the "operate" class. These operation codes may be implemented for special purpose interpretive mode operations by the programmer who wishes to take advantage of internal FP9 routines or address interpretation logic, although the floating mode "JMS" should satisfy most requirements.

II. OPERAND MODES

FP9 memory reference (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 in "memory", 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, '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 18 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' through '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:

Because this form is most similar to that of the FAC and FMQ, very little conversion is required for operand store or fetch; this mode thus provides the fastest computation speed as well as the greatest precision for operand storage.

III. INITIATING AND TERMINATING INTERPRETIVE MODE

FP9 is essentially a subroutine that operates on a variable-length argument list. The subroutine arguments are the interpretive mode instructions, each of which contains information concerning a function to be performed (the operation code) and specification of an explicit or implicit operand. Interpretive mode is initiated, then, by calling the subroutine, JMS EIM, JMS* (EIM) or JMS* EIM, where, in the last example, EIM has been declared a global symbol. Because the subroutine argument list has an indeterminate length, there must be a special argument, or interpretive-mode instruction, to effect a return from the subroutine, i.e., a return to "normal" mode; the instruction 'NOP' (740000) is used for this function. Interpretive mode instructions, then, are delineated by the two operations: JMS* (EIM) and NOP. As an example, consider the following simple routine to get the natural logarithm of a data word and store it in a three-word variable:

EFM	=	start interpreter	=	JMS* (EIM)
MODE 0	=	set mode to data	=	ØPR 7000
LAC D	=	get data word	=	LAC D
JMS* (LOG)	=	call log routine	=	JMS* (LOG)
MODE 3	=	set mode to EP	=	OPR 7003
DAC B	=	put log in B	=	DAC B
LFM	=	leave interpreter	=	NØP

The 'NOP' instruction was selected as the argument list termination, the 'LFM' instruction, so that it will be a true no-operation code if encountered in normal mode; this is a necessary feature when program flow is controlled by conditional jumps and branches. Similarly, 'EFM' is a no-op if encountered in interpretive mode; an interpretive mode JMS to the interpreter entry point is trapped and ignored.

All of the FP9 function subroutines may be entered from either normal or interpretive mode, the return is always made in interpretive mode. The subroutine 'FLOAT' must be entered in normal mode (to float the contents of the AC into the FAC), return is always made in interpretive mode. The subroutine 'FIXL' (fix the FAC to the AC and leave) may be entered from either normal or interpretive mode, the return is always in normal mode. Thus, in addition to the explicit 'EFM' and 'LFM' operations, there are several implicit means to switch the program mode.

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

IV. INTERPRETIVE MODE INSTRUCTIONS

FP9 interpretive mode instructions have exactly the same forms as the PDP-9 and PDP-15 normal mode (hardware implemented) instructions. The memory reference class (operand or address reference), however, includes all operation codes from '00' (CAL) through '70' (IOT); the "operate" class, operation code '74', includes both register modification and "immediate" instructions.

The bank mode interpreter recognizes both direct and indirect addressing and interprets a 13-bit address field for each memory reference instruction. The page mode interpreter operates on a 12-bit address field, and uses the hardware index register in effective address computation if bit 5 of the instruction word is set. Neither interpreter is capable of addressing variables outside the current 32K memory field.

FP9 does not use the Auto-increment registers, and will not modify the hardware index register. The user is cautioned not to attempt use of auto-increment addressing while in interpretive mode; the results will, in general, be unpredictable.

The descriptions of the interpretive mode instructions are presented, below, in the following format:

MNEMONIC	Intelligible MNEMONIC expansion
	some of the acceptable forms for the instruction (not necessarily complete)
	Detailed description of the operation of the instruction
	Approximate execution time
	Special conditions, precautions, and examples, if required

The mnemonics are primarily for the convenience of discussion; they are not defined in the FP9 program package. Each user may define (by direct assignment) mnemonics of his choice; the required numeric operation codes required by the interpreter are displayed in Appendix A. A "PDP" mnemonic often adequately describes the instruction (LAC, ADD, JMP), and in these cases the interpretive mode mnemonics and operation codes are equivalent to the corresponding MACRO-9/15 mnemonics and operation codes.

INTERPRETER ENTER AND LEAVE

EFM Enter Floating Mode (Interpretive Mode)

JMS EIM for absolute assemblies
JMS* (EIM) for indirect (absolute)
JMS* EIM for relocatable assembly

Transfers program control to the FP9 interpreter. All instructions following, until LFM is encountered, are assumed to be interpretive-mode instructions.

Time: 2 or 3 memory cycles.

The floating accumulator is not modified by this instruction.

EFM is a no-op if encountered in interpretive mode; no-op time is approximately 50 cycles.

FLO FLOat the AC to the FAC and enter interpreter

JMS FLOAT direct absolute
JMS* (FLOAT) indirect absolute
JMS* FLOAT relocatable

The two's complement (signed) integer in the hardware accumulator is converted to floating point format in the floating accumulator, program control is then transferred to the FP9 interpreter.

Time: approximately 40 memory cycles.

The floating accumulator is modified by this instruction.

FLO must not be executed in interpretive mode; a HLT will occur (unless the interpretive EAE instruction is implemented).

LFM Leave Floating Mode (Interpretive Mode)

NØP
740000

Program control is returned to the normal (hardware) mode at the instruction following LFM.

Time: approximately 8 memory cycles.

The floating accumulator is not modified by this instruction.

FIX FIX the FAC to the AC and leave interpreter

JMS FIXL absolute direct
JMS* (FIXL) absolute indirect
JMS* FIXL relocatable

The contents of the floating accumulator are converted to a two's complement (signed) integer and left in the hardware accumulator, program control is transferred to normal (hardware) mode at the instruction following FIX.

Time: approximately 120 or 65 memory cycles.

The floating accumulator is not modified.

FIX may be used in either normal or interpretive mode, return is always in normal mode.

If the magnitude of the FAC contents exceeds 2^{35} , zero will be left in the AC; if the magnitude is in the range 2^{17} to 2^{35} , the low order 17 bits will be left in the AC.

LOAD AND STORE OPERATIONS

MODE Set Operand MODE

747000	set mode to 'DATA'
747001	set mode to 'INTEGER'
747002	set mode to 'FLOATING POINT'
747003	set mode to 'EXTENDED PRECISION'
MODE N	set mode to 'N'

The Operand reference mode is set as indicated by the low two bits of the instruction (refer to section II for explanation of the operand modes).

Time: approximately 44 memory cycles.

The floating accumulator is not affected by this instruction.

'MODE' does not affect the operation of 'FLO' and 'FIX'.

As well as setting the operand reference modes, this instruction also sets the address increment parameter (see 'INCR', below) to 1, 1, 2, or 3 for operand modes 0, 1, 2, 3, respectively.

LAC Load the floating ACcumulator from memory

LAC	Y	direct
LAC*	Y	indirect
LAC*	Y,X	indirect, indexed (PDP-15 only)
LAC	Y,X	indexed only

The variable in locations Y, Y+1, Y+2 or Y, Y+1, or Y, depending on the current operand mode, is loaded into the floating accumulator. The number is normalized if necessary.

Time: approximately 100 memory cycles.

Previous content of the FAC is lost.

LAW Load floating Accumulator With immediate operand

LAW N
LAW -N
-N

The low 13 bits of the instruction word are taken as a signed (two's complement) integer, and floated to the FAC.

Time: approximately 60 memory cycles.

Constants in the range -4096 to $+4095_{10}$ may be generated directly in the floating accumulator, previous FAC content is lost.

DAC Deposit floating Accumulator to memory

DAC Y direct
DAC* Y indirect
DAC* Y,X indirect, indexed
DAC Y,X indexed

The content of the floating accumulator is moved to the 1, 2 or 3 memory locations specified by the effective address, after conversion to the appropriate operand format (see section II).

Time: approximately 75 memory cycles.

The floating accumulator is not modified by DAC.

Negative 'DATA' (operand mode 0) will be stored as zero; if the FAC magnitude exceeds 2^{18} or 2^{17} , only low bits will be stored for 'DATA' or 'INTEGER'; if the FAC magnitude exceeds 2^{36} or 2^{35} for modes 0 or 1, zero will be stored.

FAC MODIFICATION

CLA Clear the floating Accumulator

CLA
750000

The floating accumulator magnitude is set to zero, the sign is set positive (normal zero).

Time: approximately 37 memory cycles.

SSP Set Sign Positive

742000

The absolute value of the content of the FAC replaces the content of the FAC; the sign is forced positive, the magnitude is not affected.

Time: approximately 32 memory cycles.

CHS CHange FAC Sign

743000

The sign of the FAC is inverted; if positive, it is made negative; if negative, it is made positive. The magnitude is not affected.

Time: approximately 36 memory cycles.

SGN Convert FAC to FAC SiGN

744000

The magnitude of the FAC is forced to unity, the sign is unchanged; positive numbers, including zero, are converted to +1; negative numbers are converted to -1.

Time: approximately 48 memory cycles.

ARITHMETIC INSTRUCTIONS

SQR SQuare Root of FAC

746000

The square root of the absolute value of the FAC is computed and left in the FAC.

Time: approximately 200 memory cycles.

Accuracy is 34-35 bits.

SQA SQuare floating Accumulator

741000

The content of the floating accumulator is multiplied by itself; the product is left in the FAC.

Time: approximately 175 memory cycles.

ADD ADD memory operand to FAC

ADD	Y	
ADD*	Y	
ADD*	Y,X	} page mode } interpreter only
ADD	Y,X	

The operand in locations Y, Y+1, Y+2 or Y, Y+1, or Y, depending on the current operand mode, is added to the content of the floating accumulator; the sum is left in the floating accumulator.

Time: approximately 240 memory cycles.

If the operand magnitudes differ by a factor of 2^{35} , no addition takes place; the operand with the larger magnitude is left in the FAC.

ADI ADd Immediate operand to FAC

752NNN
ADI N&777

The low nine bits of the instruction word are taken as a two's complement signed integer (range -255 to +255) which is floated to the operand register, then added to the FAC; the sum replaces the previous content of the FAC.

Time: approximately 200 memory cycles.

If the immediate operand is zero, and the FAC magnitude is less than $1/2$, FAC precision will be lost; one should avoid zero operands in the immediate add instruction.

SUB SUBtract memory operand from FAC

SUB Y
SUB* Y
SUB* Y,X
SUB Y,X

The memory operand is subtracted from the content of the FAC; the algebraic difference replaces the previous FAC content.

Time: approximately 250 memory cycles.

See 'ADD' for special conditions.

MPY MultiPLY FAC by memory operand

MPY Y
MPY* Y
MPY* Y,X
MPY Y,X

The content of the floating accumulator is multiplied by the variable beginning in the effective address; the product replaces the previous content of the FAC.

Time: approximately 240 memory cycles.

MPI MultiPly by ImmEDIATE operand

751NNN
MPI N&777

The low nine bits of the instruction word are taken as a two's complement signed integer (range -255 to +255₁₀), which is floated to the operand register. The content of the floating accumulator is then multiplied by this operand, with the result replacing the previous content of the FAC.

Time: approximately 210 memory cycles.

DVD DiViDe FAC by memory operand

DVD Y
DVD* Y
DVD* Y,X } page mode
DVD Y,X } interpreter only

The content of the floating accumulator is divided by the operand specified by the effective address; the quotient replaces the previous content of the FAC.

If either operand is zero, no divide occurs, but the FAC is cleared to zero.

Time: approximately 240 memory cycles.

If either the floating accumulator or the memory operand is zero, the result is zero.

RDV Reverse DiViDe (memory operand by FAC)

RDV Y
RDV* Y
RDV* Y,X
RDV Y,X

The memory operand specified by the effective address is divided by the number in the floating accumulator; the quotient replaces the content of the floating accumulator; the operand is not modified. If either the FAC or the memory operand is zero, the result is set to zero.

Time: approximately 290 memory cycles.

DVI DiViDe by ImmEDIATE operand

753NNN
DVI N&777

The low nine bits of the operand are a signed (two's complement) integer by which the content of the floating accumulator is divided; the quotient is left in the floating accumulator. No zero-test is made, division by zero will produce a meaningless result.

Time: approximately 200 memory cycles.

The divide algorithm is entered without testing for zero operands. Division of zero by a non-zero divisor will produce zero. Attempted division by zero will produce a meaningless result.

PROGRAM CONTROL INSTRUCTIONS

JMP JuMP to specified location

JMP L
JMP* L
JMP* L,X
JMP L,X

Program control is transferred to the location specified by the effective address. Interpretive mode remains in effect.

Time: approximately 37 memory cycles.

JMS JuMp to Subroutine at specified location

JMS L
JMS* L
JMS* L,X
JMS L,X

The address of the JMS instruction, incremented by 1, is stored in the location specified by the effective address. Program control is then transferred to the location immediately following the effective address. Interpretive mode remains in effect.

Time: approximately 50 memory cycles.

The floating accumulator is not modified by either the JMP or JMS instruction.

CAS Compare floating Accumulator to Storage

CAS Y
CAS* Y
CAS* Y,X
CAS Y,X

The number in the floating accumulator is algebraically compared to the one-, two- or three-word operand specified by the effective address. Program control is transferred to the first, second or third location following the CAS instruction depending on whether the FAC content is less than, equal to, or greater than the operand.

Time: approximately 140 memory cycles.

The floating accumulator is not modified by the CAS instruction.

Example:

```

L      CAS      Y
program control is transferred to:
      L+1      ;C(FAC) < C(Y)
      L+2      ;C(FAC) = C(Y)
      L+3      ;C(FAC) > C(Y)

```

CAI Compare floating Accumulator to ImmEDIATE operand

```

745NNN
CAI    N&777

```

The low nine bits of the instruction word are a signed two's complement operand (range -255 to +255₁₀) to which the number in the floating accumulator is algebraically compared. Program control is transferred to the first, second or third location following the CAI instruction in the same way as for the CAS instruction.

Time: approximately 120 memory cycles.

The floating accumulator is not modified.

CAZ Compare floating Accumulator to Zero

```

745000

```

This is a special case of CAI.

Time: approximately 105 memory cycles.

BEG BEGin program loop

```

BEG    N
BEG*   L

```

Initiate a program loop, delimited by the 'BEG' and 'END' instructions; execute the loop N times. The effective address (not the content) is loaded into a loop counter and the location of the BEG instruction is saved.

Time: approximately 48 memory cycles.

Example: see 'INCR' instruction explanation.

Note that if the indirect (and index) bits are not set, the low 13 (or 12 for the page mode interpreter) bits of the instruction are loaded to the loop counter (range $1 < N < 8191$, or 4095). If the indirect bit is set, the loop count will be loaded from location specified permitting a loop count up to 2^{18} .

BEG may not have a zero effective address.

END END program loop

END

The loop counter set by the 'BEG' instruction is decremented; if it becomes zero, program control is transferred (in interpretive mode) to the instruction following 'END'; if the loop counter is not decremented to zero, program control is transferred to the location immediately following the last 'BEG' instruction to be executed.

Time: approximately 40 memory cycles.

Example: see 'INCR' instruction explanation.

BEG-END loops may not be nested; there is only one loop counter in FP9. Several 'END' instructions may be associated with each begin instruction, however, if desired.

END must have an empty address field.

INCR INCRement the specified address

INCR L
INCR* L

The content (one word) of the effective address is incremented by the current operand mode indicator (1, 1, 2 or 3 for operand modes 0, 1, 2, 3, respectively).

Time: approximately 48 memory cycles.

INCR is used for operand pointer control in program loops.

Example: add ten two-word (floating point) numbers
starting at location 'L':

LFM		/leave interpreter to
LAC	(L)	/set pointer to "L"
DAC	Z	
EFM		/enter interpreter
MODE	2	/mode to 'FP'
CLA		/clear FAC
BEG	12	/begin 10-fold loop
ADD*	Z	/add one number
INCR	Z	/increment pointer by 2
END		/go back 'til done
ANY		/come here when done

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

V. FUNCTION SUBROUTINES

The standard function subroutines included in FP9 are: arctangent (ATN), sine (SIN), cosine (COS), natural logarithm (LOG) and natural exponent (EXP). The trigonometric routines are included in program segment "F2", the natural functions in segment "F3". Some additional functions are provided in the auxiliary program segment "F6".

The functions are all coded to assure 34-35 bit (extended precision) accuracy; the evaluations are performed in extended precision mode regardless of the current user operand mode. The functions all utilize common series (fraction for ATN) expansions, so the user may easily delete one or more terms to increase execution speed at the expense of accuracy. The functions may all be entered by either normal or interpretive-mode JMS; return is always made in interpretive mode.

A brief description of each algorithm is given below. The execution times given are for the PDP-9 for illustrative purposes only; for detail on execution times, refer to Appendix B. The sizes are not precise because routines in each program segment share scratch variable storage and constants.

1. ARCTANGENT (ATN)

The arctangent function subroutine is entered with the argument in the floating accumulator. The principle value result, in the range $\pm\pi/2$, accurate to 34-35 bits, is returned in the floating accumulator. If the argument, X, is greater than one, the function is calculated as $\pi/2 - \text{ATN}(1/X)$. If the argument (or the inverse) is greater than $\sqrt{2}-1$, then the arctangent is evaluated as:

$$\text{ATN}(X) = \pi/8 + \text{ATN}((X - \text{TAN}(\pi/8))/(1 + X \cdot \text{TAN}(\pi/8))).$$

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

$$\text{atn}(X) = X / (1 + X^2 / (3 + 4X^2 / (5 + \dots 25X^2 / (11 + 36X^2 / 13))),$$

from right to left. Deletion of the last term will reduce the accuracy to 27 bits, and the execution time by ~ 800 cycles.

SIZE (OCTAL)	$\sim$	110
TIME (MSEC)	$\sim$	6.8
ACCURACY (BITS)		34-35

2. SINE AND COSINE (SIN, COS)

The sine and cosine function subroutines are entered with the argument, in radians, in the floating accumulator; the result, accurate

to 34-35 bits*, is returned in the floating accumulator. The cosine is evaluated using the sine routine through the equality: $\cos(X) = \sin(X+\pi/2)$. The sine is evaluated by reducing the argument to the first quadrant, saving the original sign and quadrant for later sign and magnitude adjustment, and evaluating the series:

$$\sin(X) = X(1 - \frac{X^2}{6}(1 - \frac{X^2}{20}(1 - \dots - \frac{X^2}{210} (1),$$

from right to left. The first term ignored is $X^{17}/17!$ which is, at most, less than 2^{-36} . The execution time (and accuracy) of the routine may be decreased by modifying a single instruction (decrement 'LAC (16)' by 2, 4, etc.).

SIZE (OCTAL)	~	116
TIME (MSEC)	~	7.2 (SIN)
ACCURACY (BITS)		34-35

3. NATURAL LOGARITHM (LOG)

The natural logarithm routine is entered with the argument in the floating accumulator. The logarithm of the absolute value of the argument, accurate to 34-35 bits, is returned in the floating accumulator. If the argument is zero, the result returned will be 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 two's exponent of X, then evaluating:

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

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

SIZE (OCTAL)	~	70
TIME (MSEC)	~	6.9
ACCURACY (BITS)		34-35

4. EXPONENTIAL (EXP)

The exponential routine is entered with the argument in the floating accumulator; the natural exponential of the argument is returned, with an accuracy of 34-35 bits, in the floating accumulator. The function is evaluated by conversion to a two's exponent:

$$e^X = 2^{a \cdot X}; a = \log_2(e).$$

* Reduction to the first quadrant involves argument truncation, which will lead to greater error for inordinately large arguments.

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:

$$2^f = e^{f/a} = e^Z = 1 + Z + \frac{Z^2}{2} + \frac{Z^3}{3} + \dots + \frac{Z^9}{9} + \dots$$

The first term ignored is $Z^{10}/10!$ which is, at most, less than 2^{-37} . The accuracy and execution time may be modified by changing only two instructions in the subroutine (change 'OPR 13011' to 'OPR 13010' and '-10' to '-7', for example, to drop the $Z^9/9!$ term).

SIZE (OCTAL)	~	60
TIME (MSEC)	~	7.2
ACCURACY (BITS)		34-35

5. SQUARE ROOT (746000)

The square root operation is included in FP9 as an operate-class instruction rather than as a function subroutine. The algorithm is described in this section for the sake of completeness. The interpretive mode instruction 'SQR' (746000) causes the contents of the floating accumulator to be replaced with the square root of the absolute value; the accuracy is 34-35 bits. The argument is reduced to a number between 1/2 and 2 by removing the even component of the exponent; this number is divided by two for later addition to the exponent of the intermediate result. A first "guess" of $\sqrt{FAC} = 0.928862$ is made, and the Newton iteration:

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

is performed three times on the high mantissa using integer arithmetic, giving 17 bits of precision. A fourth iteration on the entire mantissa extends the accuracy to at least 34 bits.

SIZE (OCTAL)	=	51
TIME (MSEC)	~	0.2
ACCURACY (BITS)		34-35

6. EXTENDED FUNCTIONS

An auxiliary program segment (F6) is included in the FP9 package to provide some extended functions and to provide working examples of FP9 usage to the programmer. The functions are not necessarily optimized for execution speed, and are not completely general, in that they use FP9 internal utility routines (save and restore user operand mode, fast subroutine return) and directly access FP9 registers, constants and subroutines. The extended functions, like the basic functions, may

be entered in either normal or interpretive mode (return is always in interpretive mode), and are evaluated in extended precision regardless of the current operand mode of the calling routine. The extended functions include: random number, arcsine, arc-cosine, arctangent of two arguments, common logarithm and general exponentiation.

RANDOM NUMBER (FRAN)

An 18-bit pseudo-random number is generated by $R = (R \cdot 218 + 438)$ modulo 2^{18} ; the new value of R is then 'floated' to the floating accumulator as a number in the range -1 to 1. The function meets the requirements that $\langle R \rangle = 0$ and $\langle R^2 \rangle = 1/3$.

ARC SINE AND ARC COSINE (ARSIN, ARCOS)

The subroutines are entered with the argument, in the range -1 to 1, in the floating accumulator; the corresponding principal value result (0 to π for arc cosine and $-\pi/2$ to $\pi/2$ for arc sine) is returned in the floating accumulator. Arc cosine is evaluated using the arc sine routine and the relation: $\arccos(X) = \arcsin(X) + \pi/2$. The arc sine is evaluated using the arctangent function subroutine:

$$\arcsin(X) = \operatorname{atan}(X/(1-X^2)^{1/2})$$

If the magnitude of the argument is greater than one, arsin returns $\pm \pi/2$, depending on the sign of the argument.

ARCTANGENT OF TWO ARGUMENTS (ATN2)

The calling sequence for the two-argument arctangent routine is:

```
JMS*      (ATN2)
.DSA      X          /address of X
.DSA      Y          /address of Y
```

The variables X and Y must have the current operand storage mode, e.g., they must be two-word variables if the interpreter is operating in operand mode '2' when the subroutine is entered. The result is returned, in the range $-\pi$ to π , in the floating accumulator. The function is evaluated using the single-argument arctangent subroutine:

$$\begin{aligned} \operatorname{ATN2}(X,Y) &= \operatorname{ATN}(Y/|X|) && ; X > 0 \\ \operatorname{ATN2}(X,Y) &= -\operatorname{ATN}(Y/|X|) - \pi && ; X < 0, Y < 0 \\ \operatorname{ATN2}(X,Y) &= -\operatorname{ATN}(Y/|X|) + \pi && ; X < 0, Y > 0 \end{aligned}$$

If Y is zero, the result is set equal to zero or π depending on the sign of X; if X is zero, $\pm \pi/2$ will be returned depending on the sign of Y; if both X and Y are zero, the result will be zero.

COMMON LOGARITHM (LOG10)

The common logarithm is evaluated by calling the natural logarithm subroutine and multiplying the result by $\log_{10}(e)$.

GENERAL EXPONENTIATION (GXP)

The general exponentiation routine allows any number (forced positive), B, to be raised to any power, P. The calling sequence for GXP is:

JMS*	(GXP)	
.DSA	B	/address of base
.DSA	P	/address of exponent

The variables B and P must have the current operand storage mode (see ATN2, above). The function is evaluated using the natural exponential and logarithm routines:

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

The result is returned in the floating accumulator. Because the FP9 natural logarithm routine forces a positive argument, negative bases are not properly handled (the result will generally be complex, in any case); the exponent, P, may be either positive or negative. The arguments may have any real values, but if the base is zero, the result returned will be one.

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

VI. INPUT AND OUTPUT

FP9 includes a free-format input controller which converts ASCII data to internal numeric format, and a variable-format output controller which translates FP9 variables to ASCII character strings. The controllers use FP9 arithmetic and logical instructions, but are accessed by the user from normal mode coding. Data can be moved between internal-numeric vectors and "IOPS ASCII" line buffers using the monitor-KM-9/15-compatible controllers, or between user variable storage and user character handling routines with the user-oriented versions. An FP9 assembly may not include both versions of the output or input controllers, but may, for example, include the monitor-compatible output controller and the user-oriented input controller. The I/O controllers are program segments:

F4U	output, user oriented
F4M	output, monitor-compatible
F5U	input, user oriented
F5M	input, monitor-compatible

1. THE INPUT CONTROLLER (FIN)

The Input Controllers convert ASCII character strings to FP9 internal-variable format and store the resultant numbers in sequential memory locations that are specified in the "input call". The character strings that are read by 'FIN' consist of groups of the form "SXXX.YYESZZT", where "S" is a sign (+ or -), "." is the decimal point, "E" is the character 'E', and "T" is a number terminator (see below). Any of the above fields may be deleted, so that an input string may have the form (comma is a terminator):

1.,1,2.5,.3,-2.3,-1E10,-1E-10,3.67E6

The only limitation is that the mantissa (XXXYY) should contain not more than 10 significant digits, i.e., it must be smaller in magnitude than 2^{36} .

Both controllers are accessed from normal mode coding by the calling sequence:

	JMS*	(FIN)	
	.DSA	SOURCE	/line buffer or subroutine
	.DSA	DESTIN	/address for first number read
	-N		/number of numbers to be read
LE	JMS	ERROR	/error return
LN	ANY		/normal return

'FIN' is a global subroutine entry in either program segment 'F5M' or 'F5U', 'SOURCE' is the address of an "IOPS ASCII" line buffer or the entry point to a user-written character handling routine depending on which version is being used, 'DESTIN' is the destination address of the

first variable to be read, 'N' is the number of variables to be converted by the input controller before returning control to the user at location 'LN' by a normal return. If 'FIN' detects an illegal character or otherwise unacceptable input, it immediately returns control to the user at 'LE' (error return). On normal return, the Nth number will be in the floating accumulator, as well as in the appropriate variable storage locations. Variables are moved to user storage according to the currently effective operand reference mode; thus if the current mode is '2' (floating point), the numbers will be stored at DESTIN, DESTIN+2, DESTIN+4, etc.; if the mode is '3', they will be stored at DESTIN, DESTIN+3, DESTIN+6, etc. One must enter interpretive mode prior to using 'FIN' to select the appropriate operand reference mode, but must leave interpretive mode to call 'FIN'.

Characters acceptable to the monitor-compatible version (F5M) are:

0-9	60-71	decimal digits
E	105	exponent indicator (10)
-	55	negation (exponent or mantissa)
.	56	decimal point
+	53	(ignored) plus sign
(LF)	12	(ignored) line feed
null	0	(ignored) leader, null
RO	177	(ignored) rub-out, leader
,	54	terminator
(TAB)	11	terminator or ignored
(blank)	40	terminator or ignored
(CR)	15	terminator and EOR
(AM)	175	terminator and EOR

If any characters other than these are encountered, an error return is made immediately. If the error occurs in the Ith group, I-1 variables will have been moved correctly to specified user storage.

The user-oriented version (F5U) interprets all but two characters in the same way as the monitor-compatible version. Rub-Out (177) is not ignored, but causes a restart of the current variable. For example, the character strings: 1,23,467,1E10, and 1,256(RØ)23,4671E10(RØ)467, 4671(RØ)1E10, are equivalent; Rub-Out cancels the character string back to the previous terminator. In the user-oriented version, CNTRL-U (25) is a legitimate character; it cancels the entire character string by simulating re-entry to 'FIN' at the current call. This special handling of "Rub-Out" and "Erase" gives the user-oriented version some of the flexibility of monitor-controlled input. The user input routine may "echo" a special character in response to these input characters, if desired, then pass the 177 or 25 on to 'FIN'.

USER INPUT ROUTINE

Each time that the user-oriented version of 'FIN' is ready for its next character it calls (JMS) the specified user-written sub-routine. The only specific requirements on the user routine are: entry and exit must be in normal mode, the operand reference mode should not

be changed in the subroutine, and a legitimate ASCII character must be returned to 'FIN' in the low seven bits of the hardware accumulator ('FIN' masks all characters to seven bits, so the high eleven bits of the accumulator are ignored). The user routines may be as simple as:

UCR	0		UCR	0
	IØT	301		LAC* 16
	JMP	.-1		JMP* UCR
	IØT	312		
	JMP*	UCR		

or as complex as necessary.

IOPS ASCII INPUT

If KM9/15 IO handlers are used for data input, the monitor-compatible version of 'FIN' may be used. This version unpacks characters from IOPS ASCII line buffers, performing essentially a memory-to-memory data transfer with conversion to FP9 numeric format. For example, suppose one wishes to read six extended precision (mode 3) numbers from the teletype (.DAT slot -2) to consecutive locations beginning at 'FD':

LB	.BLØCK	42	/line buffer
FD	.BLØCK	22	/6*3 locations
.			
.			
.			
EFM			
MØDE	3		/operand mode 3
LFM			
.			
.			
.			
.			
.			
.READ	-2,2,LB,34		/read TTY
.WAIT	-2		/wait for TTY
JMS*	(FIN)		/convert data
.DSA	LB		
.DSA	FD		
-6			
JMP	INERR		
ANY			/data are ready

The '.READ' macro may be placed anywhere preceding the conversion coding, of course, but the '.WAIT' should directly precede the call to 'FIN' for maximum efficiency. The monitor-compatible version of 'FIN' checks the validity bits and the IO mode bits of the header words to assure valid IOPS ASCII data.

The user must be sure that all buffers and data storage areas are large enough to contain the requested input. It is possible to overlay the line buffer with the numeric data if one can be certain that the required numeric storage will lag behind the character pick-up from the common area. 'FIN' retrieves two words (five characters) at a time from the line buffer, so that, for example, if the input format is such that there will be at least five characters per number group, and the operand reference mode is 2, 1 or 0, then a common area may be used. This is not a recommended procedure for a free-format input environment, but may be safely applied if input is from paper tape, cards, or some other formatted medium.

INPUT TERMINATORS

Numbers are terminated by commas (54), blanks (40) or tabs (11); leading blanks or tabs are ignored, a blank (tab) occurring after a non-blank (non-tab), non-terminator is a terminator. Carriage return (15) or Alt-mode (175) terminates the last number in a character string and signals the end of the character string (EOR). If the call to 'FIN' specifies N input variables, then 'FIN' expects the Nth number to be terminated by an "EOR" character; if "EOR" occurs before N numbers have been interpreted, or if the Nth number is not terminated by "EOR", an error return is made.

ERROR RETURNS

If the input character string is entirely satisfactory, 'FIN' returns to the user at the fifth location following the "JMS", in normal mode, with the link and the AC set to zero and the last read variable in the floating accumulator.

There are three types of error return, all to the fourth location following the call, made by 'FIN': short record, long record, and bad input. If an "EOR" character (15 or 175) is encountered before the specified number of variables has been interpreted and moved to user storage, an error return is made with the link clear and the AC containing the "shortage". For example, if six numbers are to be read but (AM) is encountered after reading four numbers, return is made with "000002" in the accumulator; four numbers will have been read to user storage and the floating accumulator will contain the fourth number.

If the Nth variable is not terminated by an "EOR", but instead by a comma, tab, or blank, the long-record error return is made with the link set and the AC clear; the specified number of variables will have been stored and the floating accumulator will contain the Nth number read. For example, if the call to 'FIN' requests three numbers, and the character string is "1,2,3,4,5(EOR)", the numbers 1, 2 and 3 will be in user storage and the long record return will be made; the floating accumulator will contain "3.0".

If an illegal character is encountered or if (in the monitor-compatible version) the first word of the "header word pair" is not XXXX02, signifying valid "IOPS ASCII" data, the "bad-input" return is

made with both the link and the accumulator set (1,777777). The contents of the floating accumulator will, in general, be meaningless; any legitimate character groups preceding the group containing the bad character will have been correctly interpreted and moved to user storage. 'FIN' does not return the offending character to the user; a user-written character routine should be written to remember each character if this information is desired; for example:

```

UCR      0
          IØT    301          /KSF
          JMP     .-1
          IØT    312          /KRB
          DAC     .+2          /save character
          JMP*    UCR          /return
CHSV     HLT          /saved character

```

The user who wishes to ignore certain errors may put a "no-op" in the error return location and code a series of conditional skips (L,AC) from the normal return; for example:

```

JMS*     (FIN)
.DSA     UCR
.DSA     Y
-N
NØP
SPA
JMP      BDIN
ANY
          /no error jmp
          /test for bad input only,
          /ignore short or long record.

```

2. THE OUTPUT CONTROLLER (FOF)

The output controllers move numbers from user variable storage to IOPS ASCII line buffers or to user-written character-handling routines, formatted as specified in the call (JMS) to the output routine. Variables are taken from sequential memory locations according to the currently effective operand reference mode. The operand mode has no effect on the written form of the number; mode 0 or 1 variables may be written in floating point formats, and mode 2 or 3 floating point numbers may be written in integer format. Both the monitor-compatible (F4M) and user-oriented (F4U) versions are accessed from normal-mode coding by the calling sequence:

```

JMS*     (FOF)
.DSA     DESTIN          /line buffer or user routine
.DSA     SOURCE          /user variable storage
FORMAT
ANY
          /format indicator
          /return location

```

'FOF' is a global subroutine entry in both versions of the output controller, 'DESTIN' is the address of an IOPS ASCII line buffer or the entry point to a user-written character-handling routine depending on which version is used, 'SOURCE' is the address of the first variable to

be written, and 'FORMAT' is an octal constant that specifies the number of variables to be written and their written appearance (see below). Return is always made in normal mode at the fourth location after the call (JMS).

The monitor-compatible version sets the header word pair to '042002,0' and fills out the final word pair with Alt-modes (175), packing the numeric output characters to 5/7 ".ASCII" form between. The user-oriented version calls (JMS) the specified user-written character-handling routine each time it has a character ready for output; all characters for this version are in the form '760200+char', where "char" is the seven-bit ASCII character code. Except for the Alt-mode written by the monitor-compatible version, the only characters used are the decimal digits (0-9), blank, minus, and decimal point.

FORMAT CONTROL

The format word in the calling sequence is of the form "XXYYZZ". The first part (XX) specifies the number of variables to be written by this call to 'FOF'; it may have any value from 1 through 778; it is not tested for zero, and if set to zero will cause 'FOF' to write 2^{18} variables. 'YY' specifies the number of characters to be written to the left of the decimal point, i.e., the "integer" part of the number. 'ZZ' specifies the number of characters, including the decimal point, to be written to the right of the "integer" part:

AAA.BBB	AAA.BBB
└───┘	└───┘
ZZ	XX ZZ

If ZZ = 0, the written number has integer format; if ZZ = 1, the decimal point is written, but the "fraction" field is empty. 'XX' must be greater than zero to allow printing of the sign (which will be blank if the number is greater than zero, "0" if the number is equal to zero), and to provide spacing between written numbers. If the variable to be written is too large for the specified field width, a single blank is written and the field is then filled out with decimal points.

This form of the format word allows 'FOF' to handle easily both "F" and "I" (FORTRAN convention) formats:

<u>"FORTRAN"</u>	<u>FP9</u>
3F10.2	030703
6F15.6	061007
10I3	120300
3I10	031200

'FOF' does not generate any vertical control characters; this allows the user to mix formats on a single output line by multiple calls to 'FOF'. The user is required, then, to generate his own CR/LF, or equivalent, whenever it is needed. Consider the following example;

print the variables X, Y, and Z in the (FORTRAN) format (1H ,I10,2F10.3), assuming that Y and Z occupy sequential variable locations:

.WAIT	-3	JMS	CRLF
.WRITE	-3,2,CRLF,34	JMS*	(FOF)
JMS*	(FOF)	UCR	
LB		X	
X		011200	
011200		JMS*	(FOF)
.WAIT	-3	UCR	
.WRITE	-3,2,LB,34	Y	
.WAIT	-3	020604	
JMS*	(FOF)		
LB			
Y			
020604			
.WRITE	-3,2,LB,34		

'FOF' writes as many characters as necessary to satisfy the format word; the user must be sure that the size of his line buffer, if he is using the monitor-compatible version or if his character routine is building a buffer, is sufficient for the number of characters that he has requested.

USER OUTPUT ROUTINE

In the user-oriented version (F4U), each time the controller has a character ready it calls (JMS) the specified user-written sub-routine with the character in the low seven bits of the AC (and "760200" in the high eleven bits). The user routine must mask the desired bits, generate parity if required, and send the character on to an output device or to a buffer. The only requirement on the output routine is that it must not be an interpretive mode routine, and it must not enter interpretive mode for any reason. It may be otherwise as complex as required, or as simple as:

UØR	0	UØR	0
IØT	401	DAC*	16
JMP	.-1	JMP*	UØR
IØT	406		
JMP*	UØR		

The output controller sends no "EOR" character to the user routine (although the monitor-oriented version inserts alt-mode in the line buffer); the user knows that 'FOF' is finished only when control is returned at the fourth location after the call.

**THIS PAGE
WAS INTENTIONALLY
LEFT BLANK**

VII. PROGRAMMING EXAMPLES

Four programs illustrating the use of FP9 are displayed in this section; they are written solely for illustrative purposes and are not necessarily optimized for either execution time or memory requirements. These examples, with the FP9 function subroutines contained in program segments 'F2' and 'F3', and the extended functions in 'F6', should provide an adequate guide to the potential user. The examples in this section are written for relocatable independent assembly, and, where applicable, assume use of the KM9/15 monitor system and I/O handlers. The "internal" functions illustrate the additional flexibility available to the programmer who will make a combined assembly of FP9 with the application programs.

The first two programs are quite simple, illustrating the use of various arithmetic and logical operations. The third program demonstrates the use of doubly nested, dependent, indexed loops, and the fourth example illustrates the FP9 input/output capabilities. Equivalent FORTRAN algorithms for the last two examples are included to serve as "flow-charts", to clarify the assembly language coding.

1. A FASTER EXPONENTIAL

The subroutine 'FEXP' uses essentially the same algorithm described in section V.4, but reduces the magnitude of the expansion variable and truncates the series at the Z^3 term. The exponential function of the fractional part is then recovered by raising the result to the eighth power:

$$\exp(f) = (\exp(f/8))^8.$$

This algorithm will give a result that is accurate to 24-25 bits; the execution time will be approximately half that of 'EXP'.

'FEXP' must be entered in the interpretive mode, with the operand reference mode set to '2' (floating point); the variables and constants L2E, FI and IF are in two-word format.

After the argument is multiplied by $\log_2(e)$, the 'FIX' instruction rounds the FAC to the nearest integer, leaves the result in the AC, and invokes normal mode. This integer part is masked to a nine-bit exponent to construct 2^{**I} in IF and IF+1.

'FLO' converts the rounded argument back to floating point and causes floating mode to be entered; the rounded argument is then subtracted from the original argument to produce the fractional series expansion argument. (Note that CIA, FLO would be faster than FLO, CHS).

After the fractional argument is divided by $\log_2(e)$, for the expansion, it is divided by eight (10_8) to allow early truncation of the series. The series is evaluated using immediate-class instructions, so that the constants 1.0, 2.0, 3.0 and 8.0 do not have to be stored in memory, saving eight core locations and reducing the execution time.

```

      .TITLE      FEXP
/ TWO-WORD OPERAND MODE ONLY.
/ INTERPRETIVE ENTRY ONLY.
      .GLOBL      FEXP,EIM,FLOAT,FI XL
MPY=400000
CHS=743000
DVD=440000
DVI=753000
ADI=752000
SQA=741000
EFM=JMS* EIM
FIX=JMS* FIXL
FLO=JMS* FLOAT
FEXP      HLT      / 24-BIT PRECISION EXPONENT.
           MPY      / X*LOG(E).
           DAC      L2E
           FIX      FI
           LMQ      / GET INTEGER PART.
           TAD      / SAVE IN MQ.
           AND      (1)
           AND      (777)
           DAC      / CONSTRUCT 2**I.
           IF
           LACQ      / GET INTEGER PART.
           FLO      / FLOAT(FIX(X*L2E)).
           CHS      / NEGATE.
           ADD      FI
           DVD      L2E
           DVI      10
           DAC      / CONVERT BACK TO E'S EXPONENT.
           DVI      / DIVIDE BY 8.
           DAC      FI
           DVI      / Z.
           ADI      3
           ADI      / Z/3.
           MPY      1
           MPY      / 1+Z/3.
           DVI      FI
           ADI      2
           ADI      / Z(1+Z/3).
           MPY      / Z/2*(1+Z/3).
           ADI      1
           MPY      / 1+Z/2*(1+Z/3).
           ADI      FI
           ADI      1
           ADI      / Z(1+Z/2(1+Z/3)).
           ADI      / 1+Z(1+Z/2(1+Z/3)).
           SQA      / HAVE EXP(F/8), RAISE TO
           SQA      / 8TH POWER BY SQUARING
           SQA      / THREE TIMES.
           MPY      IF
           JMP*      / MULTIPLY BY 2**I
           FEXP      / RETURN.
L2E      355001
          270524      / LOG(BASE 2) OF E.
FI      .BLOCK      2
IF      0
          200000      / EXPONENT FOR 2**I.
          .END
$
END FEXP

```

2. SPECTRUM CONVERSION

'LCON' converts a spectrum (data vector) to logarithmic form, scaled to 10 bits, for a display file. Two separate passes through the spectrum are required: the first to locate the maximum value, the second to perform the scaling. Conversion to logarithm can be performed in either loop. This subroutine illustrates the use of the 'BEG' and 'END' loop control instructions, the three-way conditional branch instruction 'CAS', and the address increment and immediate load instructions.

The first executable instruction is 'LFM' (NOP) so that entry may be made from either normal or interpretive mode; return will be in interpretive mode. After picking up the arguments from the calling sequence, the subroutine enters interpretive mode and sets the operand reference mode to '0' to specify eighteen-bit positive data operands. 'MAX' is preset to zero and the first loop is initiated by 'BEG* N' (BEG N would load the address field of the instruction, or 'N', into the loop counter, BEG* N indicates that the loop counter should be loaded with the content of location N). This first loop converts each entry of the spectrum to its logarithm and finds the maximum of the spectrum using the 'CAS' instruction; each time a value greater than MAX is encountered, it replaces MAX. The logarithm is calculated in extended precision (mode 3) without disturbing the current user (LCON) operand reference mode; it is rounded to an (18-bit) integer upon return to the spectrum by 'DAC* A' using mode '0'.

The instruction 'INCR A' causes the address pointer, A, to be incremented by an amount appropriate to the current operand mode (1, in this example, for operand mode 0). The following 'END' instruction delimits the first loop.

The scaling factor is calculated from the maximum value and a second loop scales the spectrum to a maximum displacement of 10 bits:

$$A(I) = 1777_8 * \text{LOG}(A(I)) / \text{MAX}.$$

```

        .TITLE      LCON
        .GLOBL      LCON,EIM,LOG
/ SUBROUTINE TO CONVERT A DATA VECTOR (SPECTRUM)
/ TO A LOGARITHMIC DISPLAY FILE (SCALED TO TEN BITS)
/ CALLING SEQUENCE:
/      JMS*      LCON
/      .DSA      A      / ADDRESS OF DATA VECTOR.
/      N          / LENGTH OF VECTOR.
/      ANY        / INTERPRETIVE RETURN.
/
LCON    HLT          / ENTRY.
        LFM          / ASSURE NORMAL MODE TO
        LAC*      LCON / PICK UP PARAMETERS.
        DAC      A#   / NEED STARTING ADDRESS
        DAC      B#   / FOR TWO LOOPS.
        ISZ      LCON
        LAC*      LCON / GET SPECTRUM LENGTH.
        DAC      N#
        ISZ      LCON
        JMS*      EIM   / ENTER INTERPRETER.
        MODE      0     / OPERAND MODE TO DATA.
        CLA
        DAC      MAX#   / SET MAXIMUM TO ZERO.
        BEG*      N     / START FIRST LOOP:
        LAC*      A     / CONVERT TO LOGARITHM,
        JMS*      LOG   / AND
        DAC*      A     / REPLACE IN SPECTRUM.
        CAS      MAX   / COMPARE TO MAXIMUM:
        JMP      .+3    / IF GREATER, REPLACE
        JMP      .+2    / MAXIMUM.
        DAC      MAX
        INCR      A     / INCREMENT POINTER.
        END        / END FIRST LOOP.
        LAW      1777   / CONVERT MAX TO SCALING
        DVD      MAX   / FACTOR FOR TEN-BIT
        DAC      MAX   / DISPLAY FILE.
        BEG*      N     / BEGIN SECOND LOOP:
        LAC*      B     / SCALE LOGARITHMIC
        MPY      MAX   / SPECTRUM USING SECOND
        DAC*      B     / ADDRESS POINTER.
        INCR      B
        END        / END LOOP '2' AND RETURN
        JMP*      LCON  / IN INTERPRETIVE MODE.
        .END
$
END LCON

```

3. A SORTING ALGORITHM

The FP9 loop control instructions may not be nested; if nested loops are required, some of the control must be handled with normal mode coding. If computation within a loop is dependent on the loop index, then an artificial index must be used (with BEG, END) or the coding must be done in normal mode; the FP9 loop index is generally not available to external routines.

The subroutine 'DSORT' illustrates a satisfactory method for handling a doubly-nested loop situation where the inner loop index is a function of the outer loop index, and where the loop indices are used as vector subscripts. The routine, displayed in FORTRAN notation as 'FSORT', orders the elements of a pair of vectors, A and B, into increasing magnitude in the elements of A. For example, the pair A = (3,-2,6,4,1), B = (7,9,3,6,2) will be ordered to A = (-2,1,3,4,6), B = (9,2,7,6,3).

The subroutine searches A(1) . . . A(N) for the smallest element; if an element smaller than A(1) is found, it is exchanged with A(1) and then the corresponding elements of B are exchanged. This operation is repeated for A(2) . . . A(N), then A(3) . . . A(N) until A(N-1) . . . A(N). 'DSORT' follows the FORTRAN equivalent precisely, except for using an index range of zero to N instead of 1 through N to simplify the calculation of subscripted variables.

The routine may be entered from either normal or interpretive mode, but always returns in normal mode. It is completely independent of the operand reference mode set in the calling program: there are no internal constants, and variable storage areas are dimensioned to allow for up to three-word variables. The 'INCR' instruction is used to determine the operand reference mode of the calling program; the sequence:

```
DZM      MOD
EFM
INCR     MOD
LFM
```

sets a multiplier in the subscripted address subroutine (SUBSC) to 1,1,2 or 3, so that a loop index (I or J) incrementing by unity will effect variable references which increment by one, two or three.

'DSORT' executes in slightly less than 1/2 of the time required by 'FSORT' (PDP-9 and two-word operands), and requires only 80% of the memory locations.

```

SUBROUTINE FSORT (A,B,N)
DIMENSION A(1),B(1)
DO 1 I=1,N
X=A(I)
Y=X
M=0
K=I+1
DO 2 J=K,N
IF (A(J).GE.X) GO TO 3
X=A(J)
M=J
3 CONTINUE
2 CONTINUE
IF (M.EQ.0) GO TO 4
A(I)=X
A(M)=Y
X=B(I)
B(I)=B(M)
B(M)=X
4 CONTINUE
1 CONTINUE
RETURN
END
$                                END FSORT

```

```

        .TITLE      DSORT
        .GLOBL      DSORT,EIM
/ DOUBLE VECTOR SORTING ROUTINE
/ ORDERS VECTOR 'A' AND CARRIES
/ ASSOCIATED 'B' ELEMENTS
/ CALLING SEQUENCE:
/       JMS          DSORT
/       A              / ADDRESS OF 'A' VECTOR.
/       B              / ADDRESS OF 'B' VECTOR.
/       N              / NUMBER OF ELEMENTS.
/       ANY            / RETURN IN NORMAL MODE.
/
EFM=JMS* EIM
INCR=000000
CAS=540000
LFM=740000
/
DSORT      0
           LFM
           LAC*      DSORT      / GET ADDRESS OF 'A'
           DAC        A1        / VECTOR AND DISTRIBUTE.
           DAC        A2
           DAC        A3
           ISZ        DSORT
           LAC*      DSORT      / DISTRIBUTE ADDRESS OF
           DAC        B1        / 'B' VECTOR.
           DAC        B2
           ISZ        DSORT
           LAC*      DSORT      / GET VECTOR LENGTH.
           DAC        N#
           ISZ        DSORT
           DZM        MOD        / SET UP SUBSCRIPT ROUTINE
           EFM        / TO ACCOUNT FOR
           INCR       MOD        / CURRENT OPERAND MODE.
           LFM
           CLA        / INITIALIZE 'I' TO ZERO.
           JMP        .+3
ILOOP      LAC        I#        / OUTER LOOP, INCREMENT
           TAD        (1)      / 'I' AND TEST FOR DONE.
           DAC        I
           SAD        N        / DONE IF I = N.
           JMP*      DSORT
           DZM        M#        / CLEAR EXCHANGE FLAG.
           JMS       SUBSC
A1          HLT          / CALCULATE ADDRESS
           DAC        AI#      / OF A(I).
           EFM
           LAC*      AI
           DAC        XX      / XX = A(I)
           DAC        YY      / YY = A(I)

```


	LFM			
	LAC	I	/ J = I+1	
	SKP		/ START INNER LOOP.	
JLOOP	LAC	J#		
	TAD	(1)	/ INCREMENT 'J' AND	
	DAC	J	/ TEST FOR DONE.	
	SAD	N		
	JMP	JDONE		
	JMS	SUBSC	/ CALCULATE ADDRESS	
A2	HLT		/ OF A(J).	
	DAC	AJ#		
	EFM		/ ENTER INTERPRETER.	
	CAS*	AJ	/ COMPARE XX TO A(J).	
	LFM		/ IF XX.LE.A(J), IGNORE.	
	JMP	.+5		
	LAC*	AJ	/ IF XX.GT.A(J), XX=A(J), M=J.	
	LFM		/ KEEP LOWER NUMBER IN FAC	
	LAC	J	/ AND SET EXCHANGE FLAG.	
	DAC	M		
	LFM			
	JMP	JLOOP	/ END OF 'J' LOOP.	
JDONE	LAC	M	/ TEST FOR EXCHANGES.	
	SNA			
	JMP	ILOOP		
	JMS	SUBSC	/ ADDRESS OF A(M).	
A3	HLT			
	DAC	AM#		
	LAC	M		
	JMS	SUBSC	/ ADDRESS OF B(M).	
B1	HLT			
	DAC	BM#		
	LAC	I		
	JMS	SUBSC	/ ADDRESS OF B(I).	
B2	HLT			
	DAC	BI#		
	EFM		/ ENTER INTERPRETER:	
	DAC*	AI	/ A(I)=XX.	
	LAC	YY		
	DAC*	AM	/ A(M)=A(I).	
	LAC*	BI		
	DAC	XX	/ XX=B(I).	
	LAC*	BM		
	DAC*	BI	/ B(I)=B(M).	
	LAC	XX		
	DAC*	BM	/ B(M)=XX.	
	LFM			
	JMP	ILOOP	/ END OF 'I' LOOP.	
SUBSC	0			
	673122		/ SUBSCRIPT CALCULATION.	
MOD	HLT		/ NEEDS CURRENT OPERAND	

	LACQ		/ MODE.
	TAD*	SUBSC \	
	ISZ	SUBSC	
	JMP*	SUBSC	
XX	.BLOCK	3	
YY	.BLOCK	3	
\$	.END		
			END DSORT

4. AN INPUT/OUTPUT EXAMPLE

The program 'ASORT', which is presented in FORTRAN notation as 'RSORT', illustrates the use of the (monitor-compatible) input and output functions. The program reads an integer, N, and then N pairs of numbers into the vectors 'A' and 'B'. The elements of the vectors are rearranged using 'DSORT' (example 3) and then the numbers are written to the output device. The primary differences between 'ASORT' and the FORTRAN algorithm are: FP9 does not require format specification for input, but because there are no implicitly "real" or "integer" variables, does require explicit operand reference mode switching.

Use of the user-oriented versions of 'FIN' and 'FOF' would be coded in the same manner as presented in 'ASORT', except that the system IO Macros .INIT, .READ, etc., and the line buffer would not be used, and the addresses of the user-written input and output subroutines would be substituted for the line buffer address in the IO calls.

FP9 input and output requires that variables be in sequential core locations. Thus to read A(I) and B(I), the numbers must be read to sequential scratch locations (ATM, BTM) and then moved to the appropriate locations within the 'A' and 'B' vectors. For output, the process is reversed; A(I) and B(I) are moved to the scratch locations before 'FOF' is used to write the numbers into the line buffer.

In this example, equivalent PDP-9/15 mnemonics have been used instead of the FP9 mnemonics (e.g., IOT instead of BEG). This practice is often desirable to conserve space in the user symbol table during assembly, and is employed for all of the interpretive mode coding internal to FP9.

```

10      DIMENSION A(400),B(400)
11      FORMAT (I10)
12      FORMAT (2F10.0)
13      FORMAT (1H1,I10)
14      FORMAT (1H ,2F10.0)
15      READ (5,10) N
16      DO 1 I=1,N
17      1   READ (5,11) A(I),B(I)
18          CALL FSORT(A,B,N)
19          WRITE (6,12) N
20          DO 2 I=1,N
21          2   WRITE (6,13) A(I),B(I)
22      STOP 77
23      END
24 $
25                                END RSORT

```

	.TITLE	ASORT	
	.GLOBL	EIM,FIN,FOF,DSORT	
	.IODEV	5,6	
ASORT	.INIT	5,0,0	
	.INIT	6,1,0	
	.READ	5,2,LB,34	/ START READ N
	JMS*	EIM	/ EFM
	747000		/ MODE 0
	NOP		/ LFM
	.WAIT	5	/ WAIT INPUT
	JMS*	FIN	/ READ N TO N
	.DSA	LB	
	.DSA	N	
	-1		
	SKP		
	JMP	CONT	
DONE	.WAIT	5	/ ERROR OR DONE
	.WAIT	6	
	CAL		
	15		
CONT	.READ	5,2,LB,34	/ START READ ON A(1),B(1)
	JMS*	EIM	/ EFM
	747002		/ MODE 2
	720000	N	/ BEG* N : START READ LOOP
	NOP		/ LFM
	.WAIT	5	/ WAIT FOR INPUT
	JMS*	FIN	/ READ A(1),B(1)
	.DSA	LB	
	.DSA	ATM	
	-2		
	JMP	DONE	
	.READ	5,2,LB,34	/ START NEXT READ
	JMS*	EIM	/ NEED DUMMY RECORD AT END
	LAC	ATM	
	DAC*	AP	/ FIRST TO A(1)
	CAL	AP	/ INCREMENT 'A' POINTER
	LAC	BTM	
	DAC*	BP	/ SECOND TO B(1)
	CAL	BP	/ INCR BP
	IOT	0	/ END LOOP
	NOP		/ LFM
	.WAIT	5	/ WAIT FOR INPUT CLEAR
	JMS*	DSORT	/ CALL DOUBLE SORT
	A		
	B		
N	0		
	JMS	CRLF	
	JMS*	EIM	/ SET MODE TO DATA
	747000		/ TO PRINT N
	NOP		

	JMS*	FOF	/ FORMAT N
	.DSA	LB	
	.DSA	N	
	011200		
	JMS	OUT	/ PRINT N
	JMS	CRLF	
	JMS*	EIM	/ EFM
	747002		/ MODE 2
	720000	N	/ BEG* N ; OUTPUT LOOP
	LAC*	AK	
	CAL	AK	
	DAC	ATM	/ MOVE A(I),B(I) TO
	LAC*	BK	/ SEQUENTIAL STORAGE
	CAL	BK	
	DAC	BTM	
	NOP		/ LFM
	JMS*	FOF	
	.DSA	LB	/ FORMAT A(I),B(I)
	.DSA	ATM	
	021200		
	JMS	OUT	/ WRITE A(I),B(I)
	JMS	CRLF	
	JMS*	EIM	
	IOT	0	/ END OUTPUT LOOP
	NOP		/ LFM
	JMS	CRLF	
	JMP	DONE	
CRLF	0		
	.WAIT	6	
	.WRITE	6,2,CR,4	
CR	JMP*	CRLF	
	4002		
	0		
	.ASCII	<15>	
OUT	0		
	.WAIT	6	
	.WRITE	6,2,LB,34	
	JMP*	OUT	
LB	.BLOCK	42	/ I/O LINE BUFFER
ATM	.BLOCK	2	/ INPUT TEMPORARY
BTM	.BLOCK	2	
AP	A		/ INPUT POINTERS
BP	B		
AK	A		/ OUTPUT POINTERS
BK	B		
A	.BLOCK	1440	
B	.BLOCK	1440	
	.END	ASORT	
\$			END ASORT

APPENDIX A

INSTRUCTION SUMMARY

<u>Instruction</u>	<u>"Op-Code"</u>	<u>Class</u>	<u>Description</u>
EFM	JMS EIM JMS* EIM [†] JMS* (EIM)	(misc.)	enter interpretive (floating) mode
LFM	740000	(misc.)	leave interpreter
MODE	747000	(misc.)	set operand mode
FLO	JMS FLOAT JMS* FLOAT [†] JMS* (FLOAT)	(misc.)	float AC to FAC (signed), enter interpretive mode
FIX	JMS FIXL JMS* FIXL [†] JMS* (FIXL)	(misc.)	fix FAC to AC (signed), leave interpretive mode
LAC	200000	(op. ref.)	load FAC
LAW	760000	(immed.)	load FAC
DAC	040000	(op. ref.)	store FAC
ADD	300000	(op. ref.)	add to FAC
ADI	752000	(immed.)	add to FAC
SUB	340000	(op. ref.)	subtract from FAC
MPY	400000	(op. ref.)	multiply FAC
MPI	751000	(immed.)	multiply FAC
DVD	440000	(op. ref.)	divide FAC
RDV	240000	(op. ref.)	divide by FAC
DVI	753000	(immed.)	divide FAC
CAS	540000	(op. ref.)	compare and branch
CAI	745000	(immed.)	compare and branch
CAZ	745000	(operate)	compare to zero
SQR	746000	(operate)	square root of FAC
SQA	741000	(operate)	square of FAC
CLA	750000	(operate)	zero to FAC
SSP	742000	(operate)	set sign positive
CHS	743000	(operate)	change FAC sign
SGN	744000	(operate)	magnitude to one
JMS	100000	(addr. cont.)	jump to subroutine
JMP	600000	(addr. cont.)	jump
BEG	700000	(addr. cont.)	start loop
END	700000	(addr. cont.)	end loop
INCR	000000	(addr. cont.)	increment address

FUNCTIONS

FATN	JMS* (ATN)	arctangent
FSIN	JMS* (SIN)	sine
FCOS	JMS* (COS)	cosine
FLOG	JMS* (LOG)	natural logarithm
FEXP	JMS* (EXP)	natural exponent

[†] .GLØBL EIM, FLØAT, FIXL

INPUT/OUTPUT

FINP	JMS* (FIN) .DSA LB (or UCR) .DSA FD -N JMP ERRØR	read N numbers to storage starting at FD
FOUT	JMS* (FOF) .DSA LB (or UCR) .DSA FD FØRMAT	output data accord- ing to format con- trol word, starting at FD

APPENDIX B

INSTRUCTION EXECUTION TIMES

Execution times for most FP9 instructions are variable, depending primarily on operand reference mode and relative magnitude and sign of the operands. The "add" instructions (ADD, ADI, SUB), in particular, are very sensitive to the relative signs and magnitudes; the time required to add a smaller number to a larger number (memory operand less than FAC) is fifty cycles less than for the addition of a large number to a small number, if the operand signs are the same (different for SUB), the operation will be sixteen cycles faster than if they are different. The operation "5+2" is 66 cycles faster than "2+(-5)".

Execution times were measured for all of the FP9 instructions, using the 60 Hz real-time clock, for 8000₁₀ random numbers in the range $\pm 1000_{10}$ (range ± 255 for immediate class, ± 2 for arctangent, $\pm 4\pi$ for sine and cosine), on a PDP-9 (1 μ sec memory) and a PDP-15 using the bank and page mode interpreters. PDP-15 bank-mode times are typically 76% faster than the corresponding PDP-9 times. Times measured on a PDP-15 with 1 microsecond memory (relocation option) were approximately 87% of the PDP-9 times.

The measured execution times, in microseconds, for the instructions and functions are tabulated below; the four rows for the operand reference instructions give the execution times for modes '3' through '0', from top to bottom. Reading from left to right, the times are for PDP-9, PDP-15 using the page-mode interpreter, PDP-15 using the bank-mode (PDP-9) interpreter. Indirect addressing will add seven cycles; indexed addressing, allowed with the page mode interpreter, will add eight cycles.

The user with a 4K or 8K memory may modify the interpreter coding that merges the bank bits into the operand address and save six cycles on each address reference instruction. The user who intends to implement FP9 on a PDP-15 only may improve execution time by changing all of the 'JMS CIA' instructions in the FP9 coding to '744031' (CLL!CMA!IAC).

1. OPERAND REFERENCE

<u>LAC</u>	102	90	81
	123	98	90
	100	98	73
	100	83	77
<u>DAC</u>	73	65	58
	75	67	58
	112	67	79
	102	77	69

<u>ADD</u>	242	198	190
	260	206	198
	237	206	183
	221	177	171

<u>SUB</u>	248	202	196
	269	210	204
	244	210	190
	265	212	204

<u>MPY</u>	235	185	177
	254	194	184
	231	194	169
	233	177	169

<u>DVD</u>	240	192	181
	258	200	192
	235	200	175
	235	185	175

<u>RDV</u>	287	227	219
	306	235	227
	283	235	212
	287	223	215

<u>CAS</u>	138	119	110
	157	127	119
	134	113	104
	146	123	113

2. IMMEDIATE CLASS

<u>LAW</u>	62	44	44
------------	----	----	----

<u>ADI</u>	204	153	153
------------	-----	-----	-----

<u>MPI</u>	217	158	158
------------	-----	-----	-----

<u>DVI</u>	206	148	148
------------	-----	-----	-----

<u>CAI</u>	126	96	96
------------	-----	----	----

3. OPERATE CLASS

<u>SQR</u>	198	150	150
------------	-----	-----	-----

<u>SQA</u>	175	130	130
------------	-----	-----	-----

<u>CLA</u>	37	29	29
------------	----	----	----

<u>SSP</u>	32	25	25
------------	----	----	----

<u>CHS</u>	36	28	28
<u>SGN</u>	49	34	34
<u>CAZ</u>	107	79	79

4. ADDRESS MODIFICATION

<u>JMP</u>	37	35	27
<u>JMS</u>	52	46	38
<u>BEG</u>	48	45	36
<u>END</u>	41	43	36
<u>INCR</u>	48	44	36

5. MISCELLANEOUS

<u>EFM</u>	3 cycles if executed (JMS*)		
<u>EFM</u>	48	42	35 (interpretive no-op)
<u>LFM</u>	7	6	6
<u>MODE</u>	44	32	32
<u>FLO</u>	40	30	30
<u>FIX</u>	118	91	84 (interpretive entry)
<u>FIX</u>	66	55	46 (normal mode entry)

6. FUNCTIONS

<u>ATN</u>	6805	5244	5109
<u>SIN</u>	7204	5492	5369
<u>COS</u>	7460	5707	5567
<u>LOG</u>	6917	5248	5137
<u>EXP</u>	7242	5544	5425

(JMS time may be subtracted if entered
in normal mode)

7. INPUT/OUTPUT

The output interpreter requires an average of 4558, 3623, 3471 microseconds to format random numbers in the range ± 1000 to "010604", which corresponds to (Fortran Notation) 1F10.3.

The input interpreter requires 1556, 1196, 1183 microseconds to read "314159E-5" from a character-per-word buffer, and move the number to user storage.

APPENDIX C

FP9 PROGRAM SOURCE LISTING

F9	Bank Mode Interpreter	1
F15	Page Mode Interpreter	2
F1	Arithmetic and Logic	3
F2	Trigonometric Functions	18
F3	Natural Functions	21
F4M	Output (MONITOR)	24
F4U	Output (USER)	29
F5M	Input (MONITOR)	33
F5U	Input (USER)	38
F6	Extended Functions	42

PAGE 1 FLOATING POINT ARITHMETIC

```

.TITLE      FPP9E
.GLOBL      EIM,FLOAT,FXL

/
/ INTERPRETER FOR PDP-9 OR
/ FOR PDP-15 IN BANK MODE
/
/ DATA STORAGE MODES
/ 3 THREE-WORD FLOATING FORMAT
/ 2 TWO-WORD FLOATING FORMAT
/ 1 ONE-WORD SIGNED INTEGERS
/ 0 ONE-WORD POSITIVE INTEGERS
/
/ W. R. MYERS
/ SCIENTIFIC DATA SYSTEMS
/ AEROJET NUCLEAR COMPANY
/ ANCR 1039
/ OCTOBER 1971
/
EIM          HLT          / START INTERPRETIVE MODE.
              LAC*        EIM          / GET NEXT INSTRUCTION.
              SAD          FIXL+1      / LEAVE INTERPRETER IF NOP.
              JMP*        EIM
              653605      / OP-CODE TO AC AND
              RCR          / INDIRECT BIT TO LINK.
              SAD          (17)        / OPERATE CLASS
              JMP          FOPR        / TESTED SEPARATELY.
              TAD          .+13        / SET UP BRANCH
              DAC          .+7         / TO FUNCTION.
              641505      / 13 BIT ADDRESS TO MQ,
              LAC          EIM          / COMBINE WITH BANK BITS.
              AND          (60000)
              640002
              DAC          ADR
              744400      / INDIRECT ADDRESS IF REQUIRED
              HLT          / AND GO TO EXECUTE FUNCTION.
              LAC*        ADR
              JMP          .-4
              .EOT

$
END F9

```

PAGE 2 FLOATING POINT ARITHMETIC

.TITLE FPP15
 .GLOBL EIM,FLOAT,FXL
 .DBREL

/

/ INTERPRETER FOR PDP-15 IN PAGE MODE

/

/ DATA STORAGE MODES

/ 3 THREE-WORD FLOATING FORMAT

/ 2 TWO-WORD FLOATING FORMAT

/ 1 ONE-WORD SIGNED INTEGERS

/ 0 ONE-WORD POSITIVE INTEGERS

/

/ W. R. MYERS

/ SCIENTIFIC DATA SYSTEMS

/ AEROJET NUCLEAR COMPANY

/ ANCR 1039

/

EIM	HLT		/ START INTERPRETIVE MODE.
	LAC*	EIM	/ GET NEXT INSTRUCTION.
	SAD	FIXL+1	/ LEAVE INTERPRETER IF NOP.
	JMP*	EIM	
	653605		/ OP-CODE TO AC AND
	RCR		/ INDIRECT BIT TO LINK.
	SAD	(17)	/ OPERATE CLASS TESTED
	JMP	FOPR	/ SEPARATELY.
	TAD	+.26	/ SET UP BRANCH TO
	DAC	+.20	/ FUNCTION.
	LAC*	EIM	
	AND	(7777)	/ DETERMINE 12 BIT ADDRESS
	DAC	ADR	/ AND ADD PAGE BITS.
	LAC	EIM	
	AND	(70000)	
	TAD	ADR	
	DAC	ADR	
	744400		/ TEST FOR INDIRECT.
	JMP	+.3	
	LAC*	ADR	
	DAC	ADR	
	641601		/ IF INDEXED, ADD CONTENTS
	744200		/ OF HARDWARE INDEX
	JMP	+.3	/ REGISTER.
	LAC	ADR	
	HLT		/ GO TO FUNCTION.
	724000		
	TAD	ADR	
	DAC	ADR	
	JMP	-.4	
	.EOT		

\$

END F15

PAGE 3 FLOATING POINT ARITHMETIC

.TITLE FPPI ARITHMETIC

```

/
/ W. R. MYERS
/ SCIENTIFIC DATA SYSTEMS
/ AEROJET NUCLEAR COMPANY
/ ANCR 1039
/ OCTOBER 1971
/
/ FLOATING POINT ARITHMETIC
/ DATA FORMATS:
/     3 THREE WORD FLOATING
/     2 TWO WORD FLOATING
/     1 ONE WORD SIGNED INTEGER
/     0 ONE WORD DATA
/ INTERPRETIVE INSTRUCTIONS:
/     JMS     EIM      / ENTER INTERPRETER
/     NOP
/     DAC     X        / STORE FAC IN X
/     LAC     X        / LOAD FAC FROM X
/     LAW     N        / IMMEDIATE LOAD N TO FAC
/     DZM
/     AND
/     EAE
/     IOT     N        / START N-FOLD LOOP
/     CAL     L        / INCREMENT L BY DATA MODE
/     IOT     O        / LOOP END INDICATOR
/     ADD     X        / ADD X TO FAC
/     TAD     X        / SUBTRACT X FROM FAC
/     XCT     X        / MULTIPLY FAC BY X
/     ISZ     X        / DIVIDE FAC BY X
/     XOR     X        / DIVIDE X BY FAC TO FAC
/     JMS     L        / JMS TO L, FLOATING MODE
/     JMP     L        / TRANSFER CONTROL TO L
/     SAD     X        / COMPARE AND BRANCH
/     OPR     0        / LEAVE INTERPRETER
/     OPR     1000     / SQUARE OF FAC
/     OPR     2000     / SET FAC SIGN POSITIVE
/     OPR     3000     / CHANGE SIGN OF FAC
/     OPR     4000     / SET MAGNITUDE TO ONE
/     OPR     5NNN     / COMPARE IMMEDIATE, BRANCH
/     OPR     6000     / SQUARE ROOT OF FAC
/     OPR     700N     / SET DATA MODE TO N
/     OPR     10000    / SET FAC TO ZERO
/     OPR     11NNN    / IMMEDIATE MULTIPLY
/     OPR     12NNN    / IMMEDIATE ADD
/     OPR     13NNN    / IMMEDIATE DIVIDE
/ INDIRECT ADDRESSING MAY BE USED WITH
/ LAC, DAC, ADD, TAD, XCT, ISZ,
/ XOR, JMS, JMP, SAD, CAL, IOT.
/

```


PAGE 4 FLOATING POINT ARITHMETIC

	JMP	.+1	
	JMP	FINQ	/ CAL = INCREMENT.
	JMP	FDAC	/ DAC = DEPOSIT FAC.
	JMP	FJMS	/ JMS = JMS.
	HLT		/ DZM NOT USED.
	JMP	FLAC	/ LAC = LOAD FAC.
	JMP	RDIV	/ XOR = INVERT DIVIDE.
	JMP	FADD	/ ADD = FLOATING ADD.
	JMP	FSUB	/ TAD = SUBTRACT.
	JMP	FMPY	/ XCT = MULTIPLY.
	JMP	FDIV	/ ISZ = DIVIDE.
	HLT		/ AND NOT USED.
	JMP	FCAS	/ SAD = THREE-WAY COMPARE.
	JMP	SRTN	/ JMP = JMP.
	HLT		/ EAE NOT USED.
FOPR	JMP	FINX	/ IOT = LOOP CONTROL.
	745400		/ OPERATE CLASS DECODE.
	JMP	FLOD	/ LAW IS IMMEDIATE LOAD.
	641604		
	TAD	.+3	
	DAC	.+1	
	HLT		
	JMP*		
	SQAX		/ 1000 SQUARE FAC.
	FZAC+3		/ 2000 ABSOLUTE VALUE.
	FZAC+5		/ 3000 CHANGE SIGN.
	FSGN		/ 4000 UNIT MAGNITUDE.
	FCZ		/ 5000 BRANCH ON ZERO.
	SQRX		/ 6000 SQUARE ROOT.
	FSOM		/ 7000 DATA MODE.
	FZAC		/ 10000 CLEAR FAC.
	MPIR		/ 11000 IMMEDIATE MULTIPLY.
	ADIR		/ 12000 IMMEDIATE ADD.
	DVIR		/ 13000 IMMEDIATE DIVIDE.
FZAC	DZM	AC+3	/ ZERO TO FAC.
	DZM	AC+2	
	DZM	AC+1	
	DZM	AC	/ SET SIGN PLUS.
	JMP	BAK	
	LAC	(1)	/ CHANGE SIGN.
	XOR	AC	
	DAC	AC	
	JMP	BAK	
FSDM	641611		/ SET DATA MODE.
	DAC	DM	
	745202		
	GLK		
	DAC	DM+1	
	JMP	BAK	

PAGE 5 FLOATING POINT ARITHMETIC

AC	.BLOCK	4	/ FLOATING ACCUMULATOR.
BC	.BLOCK	4	/ OPERAND OR 'MQ'.
ADR	0		/ OPERAND ADDRESS.
DM	2		/ DATA MODE INDICATOR.
	2		
CIA	HLT		/ TWO'S COMPLEMENT AC.
	744001		
	TAD	(1)	
	JMP*	CIA	
ACZ	HLT		/ TEST FAC FOR ZERO.
	LAC	AC+2	
	ADD	AC+3	
	744200		/ RETURN TO LOC+1 IF
	JMP	.+4	/ ZERO, TO LOC+2 IF
	DZM	AC+1	/ NOT ZERO.
	DZM	AC	
	JMP*	ACZ	
	ISZ	ACZ	
	JMP*	ACZ	
BCZ	HLT		/ TEST OPERAND ZERO.
	LAC	BC+2	
	ADD	BC+3	
	744200		
	ISZ	BCZ	
	JMP*	BCZ	
SGN	HLT		/ COMPARE SIGNS OF
	LAC	AC	/ FAC AND OPERAND.
	XOR	BC	
	JMP*	SGN	
XAB	HLT		/ EXCHANGE OPERAND AND
	LAC	AC	/ FLOATING ACCUMULATOR.
	RAR		
	LAC	BC	
	DAC	AC	
	GLK		
	DAC	BC	
	LAC	AC+1	
	652000		
	LAC	BC+1	
	DAC	AC+1	
	641002		
	DAC	BC+1	
	LAC	AC+2	
	652000		
	LAC	BC+2	
	DAC	AC+2	
	641002		
	DAC	BC+2	
	LAC	AC+3	
	652000		

PAGE 6 FLOATING POINT ARITHMETIC

	LAC	BC+3	
	DAC	AC+3	
	641002		
	DAC	BC+3	
	JMP*	XAB	
SVMD	HLT		/ SAVE DATA MODE,
	NOP		/ THEN SET TO THREE.
	LAC	DM	
	DAC	.+5	
	LAC	(3)	
	DAC	DM	
	LAC	SVMD	
	JMP	SRTN	/ LEAVE IN INTERPRETIVE MODE.
	HLT		
RSMD	HLT		/ RESTORE USER DATA
	NOP		/ MODE AND LEAVE
	LAC	.-3	/ INTERPRETER.
	DAC	DM	
	JMP*	RSMD	
	EIM		
FJMS	SAD	.-1	/ FLOATING MODE JMS.
	JMP	BAK	
	LAC	EIM	/ JMS EIM AND JMS* (EIM) ARE
	TAD	(1)	/ NO-OPS IN FLOATING MODE.
	DAC*	ADR	
	LAC	ADR	
	TAD	(1)	
	JMP	SRTN	
FLOD	641002		/ IMMEDIATE LOAD 13-BIT
	660505		/ SIGNED INTEGER TO FAC.
	JMS	FLOQ	
	JMP	BAK	
FLOAT	HLT		/ FLOAT AN 18-BIT SIGNED
	JMS	FLOQ	/ TWO'S COMPLEMENT INTEGER
	LAC	FLOAT	/ TO FAC AND ENTER
SRTN	DAC	EIM	/ INTERPRETIVE MODE.
	JMP	EIM+1	
FLOQ	HLT		/ UTILITY ROUTINE FOR
	664000		/ FLOATING A SIGNED
	652000		/ INTEGER TO FAC.
	GLK		
	DAC	AC	
	641002		
	TAD	AC	
	650422		
	DAC	AC+2	
	SNA		
	JMP	.+4	
	-100		
	640001		

PAGE 7 FLOATING POINT ARITHMETIC

	CMA		
	DAC	AC+1	
	DZM	AC+3	
	JMP*	FLOQ	
FLAC	JMS	GET	/ FLOATING LOAD FAC
	LAC	BC	/ FROM MEMORY, GET
	DAC	AC	/ OPERAND THEN MOVE
	LAC	BC+1	/ TO FAC.
	DAC	AC+1	
	LAC	BC+2	
	DAC	AC+2	
	LAC	BC+3	
	DAC	AC+3	
	JMP	BAK	
FSGN	LAC	(1)	/ SET FAC MAGNITUDE
	DAC	AC+1	/ TO ONE, LEAVE
	660720		/ SIGN UNCHANGED.
	DAC	AC+2	
	DZM	AC+3	
	JMP	BAK	
FCZ	JMS	IMOP	/ THREE-WAY COMPARE
	JMP	FCAS+1	/ IMMEDIATE OPERAND.
	LAC	AC	
	SZA		
	JMP	BAK	
	ISZ	EIM	
	ISZ	EIM	
	JMP	BAK	
FCAS	JMS	GET	/ THREE-WAY COMPARE FAC
	JMS	ACZ	/ TO OPERAND, GO TO
	JMP	.+17	/ LOC+1 IF FAC LESS,
	JMS	BCZ	/ LOC+2 IF EQUAL,
	JMP	FCZ+2	/ LOC+3 IF GREATER.
	JMS	SGN	
	SZA		
	JMP	FCZ+2	
	JMS	CASQ	
	SNA		
	JMP	FCAS-2	
	SMA		
	JMP	FCZ+2	
	LAC	BC	
	SZA		
	JMP	FCAS-3	
	JMP	BAK	
	JMS	BCZ	
	JMP	FCAS-2	
	JMP	.-6	
CASQ	HLT		/ COMPARE MAGNITUDES OF
	LAC	BC+1	/ FAC AND OPERAND.

PAGE 8 FLOATING POINT ARITHMETIC

	JMS	CIA	/ RETURN NEGATIVE IF
	TAD	AC+1	/ FAC LESS, ZERO IF
	SZA		/ EQUAL, POSITIVE NON-ZERO
	JMP*	CASQ	/ IF FAC GREATER.
	LAC	BC+2	
	JMS	CIA	
	TAD	AC+2	
	SZA		
	JMP*	CASQ	
	LAC	BC+3	
	JMS	CIA	
	TAD	AC+3	
	740202		
	RAR		
	JMP*	CASQ	
FDAC	LAC	AC	/ DEPOSIT FLOATING
	RCR		/ ACCUMULATOR TO MEMORY.
	750020		
	XOR	AC+2	/ MOVE HIGH MANTISSA
	652000		/ AND SIGN TO MQ, THEN
	LAC	DM	/ TEST DATA MODE.
	RCR		
	SNA		/ DATA MODE 2 OR 3 ?
	JMP	.+21	/ NO, MUST BE 1 OR 0.
	LAC	AC+3	/ YES.
	SZL		/ DATA MODE 2 ?
	JMP	.+6	/ NO.
	AND	(777000)	/ YES, PACK LOW MANTISSA
	DAC	BC+3	/ AND EXPONENT INTO
	LAC	AC+1	/ ONE WORD.
	AND	(777)	
	XOR	BC+3	
	DAC*	ADR	/ DEPOSIT FIRST WORD.
	641002		/ SECOND WORD IN MQ.
	ISZ	ADR	
	DAC*	ADR	/ DEPOSIT SECOND WORD.
	744400		/ DATA MODE 3 ?
	JMP	BAK	/ NO, RETURN.
	LAC	AC+1	/ YES, SET-UP TO
	JMP	.-5	/ STORE EXPONENT.
	SZL		/ MODE ZERO OR ONE.
	JMP	.+7	
	LAC	AC	/ MODE ZERO, IF FAC
	750200		/ NEGATIVE, STORE ZERO.
	JMP	.+5	
	JMS	FXQ	/ CONVERT TO ROUNDED
	TAD	(1)	/ 18-BIT INTEGER.
	741020		
	JMS	FIXL	/ MODE ONE, CONVERT
	DAC*	ADR	/ TO SIGNED INTEGER.

PAGE 9 FLOATING POINT ARITHMETIC

	JMP	BAK	
FINQ	LAC*	ADR	/ ADD CURRENT DATA MODE
	TAD	DM+1	/ TO SPECIFIED LOCATION.
	JMP	.-4	
FXQ	HLT		/ ROUTINE TO CONVERT
	LAC	AC+1	/ FAC TO AN INTEGER.
	744101		
	JMP	.-+3	/ LEAVES 18 BIT
	651000		/ UNSIGNED INTEGER
	JMP*	FXQ	/ IN L, AC-0 TO AC-16.
	TAD	(43)	/ AC-17 CONTAINS THE
	AND	(77)	/ MOST SIGNIFICANT
	XOR	(660500)	/ FRACTION BIT.
	DAC	.-+4	
	LAC	AC+3	
	652000		
	LAC	AC+2	
	HLT		
	RAR		
	641002		
	JMP*	FXQ	
FIXL	HLT		/ FLOATING MODE
	NOP		/ FUNCTION TO
	JMS	FXQ	/ TRANSLATE FAC TO
	TAD	(1)	/ SIGNED INTEGER AND
	RCR		/ RETURN IN NORMAL MODE.
	RCL		
	XOR	AC	/ MAY BE CALLED FROM
	RAR		/ NORMAL MODE.
	SZL		
	JMS	CIA	
	JMP*	FIXL	
	.BLOCK	2	
FINX	740201		/ LOOP CONTROL.
	JMP	.-+6	/ IF ZERO OPERAND,
	ISZ	FINX-2	/ INCREMENT LOOP COUNTER,
	SKP		/ CONTINUE LOOP OR LEAVE.
	JMP	BAK	
	LAC	FINX-1	/ SAVE LOOP START LOCATION
	JMP	SRTN	/ AND SET LOOP COUNTER.
	TAD	(1)	
	DAC	FINX-2	
	ISZ	EIM	
	LAC	EIM	
	DAC	FINX-1	
	JMP	EIM+1	
GET	HLT		/ OPERAND FETCH.
	LAC	DM	/ TEST DATA MODE FOR
	RCR		/ FLOATING OR INTEGER.
	SZA		

PAGE 10 FLOATING POINT ARITHMETIC

JMP	..+20	/ FLOATING.
LAC*	ADR	/ INTEGER, GET ONE WORD.
740600		/ TEST MODE FOR 0 OR 1.
JMP	..+3	
JMS	BLSO	/ MODE 1, USE SIGNED INTEGER
JMP*	GET	/ LOAD ROUTINE.
SMA		/ IF MODE 0 AND POSITIVE,
JMP	..-3	/ USE SIGNED LOAD ROUTINE.
RCR		/ IF 'NEGATIVE', THEN
DAC	BC+2	/ NORMALIZED DATA LOAD
750020		/ IS DEFINED EXACTLY.
DAC	BC+3	
LAC	(22)	
DAC	BC+1	
DZM	BC	
JMP*	GET	
LAC*	ADR	/ FLOATING FORMAT, 2 OR 3.
652000		/ PUT FIRST WORD IN MQ.
ISZ	ADR	
LAC*	ADR	/ GET SECOND WORD.
660000		/ SAVE SIGN IN LINK.
AND	(377777)	/ MASK OFF SIGN AND
DAC	BC+2	/ PUT IN HIGH MANTISSA.
GLK		/ MOVE LINK TO SIGN.
DAC	BC	
LAC	DM	
RCR		
641002		
SNL		
AND	(777000)	
DAC	BC+3	/ IF MODE 2, DECOMPOSE
SNL		/ FIRST WORD TO LOW
JMP	..+4	/ MANTISSA AND EXPONENT.
ISZ	ADR	/ IF MODE 3, GET THIRD
LAC*	ADR	/ WORD FOR EXPONENT.
JMP	..+3	
640633		
660511		
DAC	BC+1	/ NORMALIZE FLOATING
LAC	BC+2	/ OPERAND REGISTER.
RCL		/ IF BIT ONE IS SET,
SPA		/ THEN ALREADY NORMAL.
JMP*	GET	
ADD	BC+3	/ IF MANTISSA IS ZERO,
SZA		/ THEN CLEAR EXPONENT
JMP	..+4	/ AND SIGN FOR
DZM	BC	/ NORMAL ZERO.
DZM	BC+1	
JMP*	GET	
LAC	BC+3	/ OTHERWISE, SHIFT LEFT

PAGE 11 FLOATING POINT ARITHMETIC

	652000		/ UNTIL NORMALIZED
	LAC	BC+2	/ AND ADJUST EXPONENT.
	660444		
	DAC	BC+2	
	641002		
	DAC	BC+3	
	-100		
	640001		
	TAD	(43)	
	CMA		
	TAD	BC+1	
	DAC	BC+1	
	JMP*	GET	
BLSO	HLT		/ LOAD SIGNED INTEGER
	664000		/ OPERAND TO FLOATING
	652000		/ OPERAND REGISTER.
	GLK		
	DAC	BC	
	641002		
	TAD	BC	
	650422		
	DAC	BC+2	
	SNA		
	JMP	+.4	
	-100		
	640001		
	CMA		
	DAC	BC+1	
	DZM	BC+3	
	JMP*	BLSO	
IMOP	HLT		/ MOVE IMMEDIATE OPERAND
	641002		/ TO FLOATING OPERAND
	660511		/ REGISTER.
	JMS	BLSO	
	JMP*	IMOP	
ADIR	JMS	IMOP	/ ADD IMMEDIATE.
	JMP	FADD+3	
DVIR	JMS	IMOP	/ DIVIDE IMMEDIATE.
	JMP	FDIV+5	
MPIR	JMS	IMOP	/ MULTIPLY IMMEDIATE.
	JMP	FMPY+1	
NOR	HLT		/ NORMALIZE FAC.
	LAC	AC+2	
	745110		/ 'OVERFLOW' ?
	JMP	+.23	/ YES.
	SPA		/ NO, NORMALIZED ?
	JMP*	NOR	/ YES, DONE.
	JMS	ACZ	/ NO, ZERO ?
	JMP*	NOR	/ YES, DONE.
	LAC	AC+3	/ NO, NORMALIZE

PAGE 12 FLOATING POINT ARITHMETIC

	652000		/ MANTISSA AND
	LAC	AC+2	/ ADJUST EXPONENT.
	660444		
	DAC	AC+2	
	641002		
	DAC	AC+3	
	-100		
	640001		
	TAD	(43)	
	CMA		
	TAD	AC+1	
	DAC	AC+1	
	JMP*	NOR	
	RTR		/ OVERFLOW, MOVE DOUBLE
	DAC	AC+2	/ WORD MANTISSA ONE
	LAC	AC+3	/ BIT TO RIGHT AND
	RAR		/ INCREMENT EXPONENT.
	DAC	AC+3	
	ISZ	AC+1	
	JMP*	NOR	
	JMP*	NOR	
FSUB	JMS	GET	/ SUBTRACT, GET OPERAND,
	LAC	BC	/ COMPLEMENT, AND
	XOR	(1)	/ GO TO ADD.
	DAC	BC	
	SKP		
FADD	JMS	GET	/ ADD, GET OPERAND.
	JMS	BCZ	/ IS ADDEND ZERO ?
	JMP	BAK	/ YES, DONE.
	JMS	ACZ	/ NO, IS AUGEND ZERO ?
	JMP	FLAC+1	/ YES, MOVE ADDEND TO FAC.
	JMS	CASQ	/ NO, COMPARE MAGNITUDES.
	SPA		/ IS ADDEND GREATER ?
	JMS	XAB	/ YES, EXCHANGE.
	LAC	BC+1	/ CALCULATE EXPONENT
	JMS	CIA	/ DIFFERENCE FOR
	TAD	AC+1	/ DENORMALIZATION OF 'MQ'.
	741201		/ IS DIFFERENCE ZERO ?
	JMP	+.17	/ YES.
	652004		/ NO, SHIFT 'MQ' TO
	TAD	(43)	/ RIGHT UNTIL EXPONENTS
	SPA		/ ARE EQUALIZED.
	JMP	BAK	
	641002		
	XOR	(660500)	
	DAC	+.4	
	LAC	BC+3	
	652000		
	LAC	BC+2	
	HLT		

PAGE 13 FLOATING POINT ARITHMETIC

	DAC	BC+2	
	641002		
	DAC	BC+3	
	JMS	SGN	/ COMPARE SIGNS.
	745200		/ DIFFERENT ?
	JMP	.+11	/ NO.
	LAC	BC+3	/ YES, COMPLEMENT 'MQ'.
	JMS	CIA	
	DAC	BC+3	
	LAC	BC+2	
	741401		
	TAD	(1)	
	DAC	BC+2	
	CLL		
	LAC	AC+3	/ PERFORM DOUBLE
	TAD	BC+3	/ PRECISION ADD.
	DAC	AC+3	
	GLK		
	TAD	AC+2	
	TAD	BC+2	
	DAC	AC+2	/ GO TO NORMALIZE
	JMP	AOT	/ AND OUT.
MPTN	HLT		/ MULTIPLY FAC BY 10.
	LAC	AC+1	
	TAD	(3)	/ ADD 3 TO EXPONENT.
	DAC	AC+1	
	LAC	AC+3	/ PUT TWO-WORD MANTISSA
	652000		/ IN AC AND MQ.
	LAC	AC+2	
	660502		/ DIVIDE BY FOUR.
	DAC	DVTN	/ SAVE HIGH MANTISSA.
	641002		/ GET ADJUSTED LOW
	TAD	AC+3	/ MANTISSA AND ADD TO
	DAC	AC+3	/ ORIGINAL.
	GLK		/ GET OVERFLOW.
	TAD	DVTN	/ ADD ADJUSTED HIGH
	TAD	AC+2	/ MANTISSA AND ORIGINAL.
	DAC	AC+2	/ PUT IN HIGH MANTISSA.
	JMS	NOR	/ NORMALIZE.
	JMP*	MPTN	
DVTN	HLT		/ DIVIDE FAC BY 10.
	-4		/ REDUCE EXPONENT
	TAD	AC+1	/ BY FOUR.
	DAC	AC+1	
	LAC	AC+3	/ DIVIDE TWO-WORD
	652000		/ MANTISSA BY 10.
	LAC	AC+2	
	660323		
	500000		
	DAC	MPTN	/ SAVE REMAINDER.

PAGE 14 FLOATING POINT ARITHMETIC

FMPY	641002			
	DAC	AC+2	/	STORE QUOTIENT IN HIGH.
	LAC	MPTN		
	650323		/	DIVIDE REMAINDER BY 10.
	500000			
	641002		/	STORE SECOND QUOTIENT
	DAC	AC+3	/	IN LOW MANTISSA.
	JMS	NOR	/	NORMALIZE.
	JMP*	DVTN		
	JMS	GET	/	MULTIPLY.
	JMS	SGN	/	GET SIGN OF RESULT.
	DAC	AC		
	LAC	AC+1	/	ADD EXPONENTS.
	TAD	BC+1		
	DAC	AC+1		
	LAC	BC+3	/	DISTRIBUTE LOW ORDER
	RCL		/	MULTIPLIER, SHIFTED
	DAC	.+11	/	TO IMPROVE PRECISION.
	DAC	.+27		
	LAC	BC+2	/	DISTRIBUTE HIGH
	RAL		/	MULTIPLIER, SHIFTED.
	DAC	.+11		
	DAC	.+40		
	DZM	BC+2	/	ZERO HIGH PRODUCT.
	LAC	AC+3		
	653122			
	HLT		/	HIGH PART OF LOW * LOW,
	DAC	BC	/	THIRD WORD OF PRODUCT.
	LAC	AC+3		
	653122			
	HLT		/	HIGH PART OF LOW * HIGH,
	DAC	BC+1	/	MIDDLE WORD OF PRODUCT.
641002				
TAD	BC	/	ADD LOW OF LOW * HIGH	
DAC	BC	/	TO THIRD WORD AND	
745400		/	CARRY OVERFLOWS.	
ISZ	BC+1			
745000				
ISZ	BC+2			
LAC	AC+2			
653122				
HLT		/	ADD HIGH OF HIGH * LOW	
TAD	BC+1	/	TO MIDDLE WORD AND	
DAC	BC+1	/	CARRY OVERFLOW.	
745400				
ISZ	BC+2			
641002				
TAD	BC	/	ADD LOW OF HIGH * LOW	
745400		/	AND CARRY OVERFLOWS.	
ISZ	BC+1			

	745000		
	ISZ	BC+2	
	LAC	AC+2	
	653122		/ HIGH PART OF HIGH * HIGH
	HLT		/ ADDED TO PREVIOUS HIGH
	TAD	BC+2	/ PRODUCT GOES TO
	DAC	AC+2	/ HIGH MANTISSA.
	641002		
	TAD	BC+1	/ ADD LOW PART OF HIGH * HIGH
	DAC	AC+3	/ AND CARRY OVERFLOW.
	745400		
	ISZ	AC+2	/ RETURN.
RDIV	JMP	AOT	
	JMS	GET	
	JMS	BCZ	/ INVERSE DIVIDE, GET
	JMP	FZAC	/ OPERAND, THEN EXCHANGE.
	JMS	XAB	
	JMP	.+4	
FDIV	JMS	ACZ	/ DIVIDE, IS FAC ZERO ?
	JMP	BAK	/ YES, DONE.
	JMS	GET	/ NO, GET OPERAND.
	JMS	BCZ	/ IS OPERAND ZERO ?
	JMP	FZAC	/ YES, CLEAR FAC AND OUT.
	JMS	SGN	/ NO, GET SIGN OF
	DAC	AC	/ QUOTIENT.
	LAC	BC+1	
	JMS	CIA	/ SUBTRACT EXPONENTS FOR
	TAD	AC+1	/ EXPONENT OF RESULT.
	DAC	AC+1	
	LAC	BC+3	/ LOW ORDER DIVISOR,
	RCL		/ SHIFTED LEFT TO
	DAC	.+10	/ AVOID OVERFLOW.
	LAC	BC+2	
	RAL		/ HIGH ORDER DIVISOR
	DAC	.+7	/ SHIFTED AND DISTRIBUTED
	DAC	.+17	/ FOR USE BY DIVIDES.
	DAC	.+24	
	LAC	AC+2	/ USE $(A+BK)/(C+DK) =$
	653122		/ $(A+(B-AD/C)K)/C,$
	HLT		/ ALGORITHM BY D D METCALF.
	640323		
	HLT		
	641002		/ AD/C.
	JMS	CIA	
	TAD	AC+3	
	652000		/ $(B-AD/C)K.$
	754400		
NEG1	-1		
	TAD	AC+2	/ FINISH WITH DOUBLE
	660323		/ PRECISION DIVIDE

PAGE 16 FLOATING POINT ARITHMETIC

	HLT		/ BY HIGH DIVISOR.
	DAC	BC+3	
	641002		
	DAC	AC+2	
	LAC	BC+3	
	650323		
	HLT		
	641002		
SQO	DAC	AC+3	
AOT	JMS	NOR	/ ARITHMETIC RETURN.
BAK	ISZ	EIM	/ INCREMENT FPC AND
	JMP	EIM+1	/ CONTINUE INTERPRETER.
SQAX	DZM	AC	/ SQUARE OF FAC.
	LAC	AC+1	
	RCL		
	DAC	AC+1	
	LAC	AC+2	
	DAC	BC+2	
	LAC	AC+3	
	JMP	FMPY+7	
SQRX	JMS	ACZ	/ SQUARE ROOT OF FAC.
	JMP	BAK	
	-3		
	DAC	AC	
	LAC	AC+1	/ DIVIDE EXPONENT BY TWO.
	660000		
	RAR		
	DAC	AC+1	
	SZL		/ ADJUST MANTISSA FOR EVEN
	JMP	.+7	/ EXPONENT, THEN PERFORM
	LAC	AC+2	/ NEWTON ITERATION ON
	RCR		/ HIGH ORDER PART OF A
	DAC	AC+2	/ NUMBER BETWEEN
	LAC	AC+3	/ ONE-HALF AND TWO.
	RAR		/ ALGORITHM BY D D METCALF.
	DAC	AC+3	
	LAC	AC+2	
	TAD	(166712)	
	DAC	.+6	
	DAC	.+21	
	LAC	AC+3	/ ITERATE THREE TIMES
	652000		/ ON HIGH ORDER:
	LAC	AC+2	/ $G(I+1) = (G(I) + X/G(I))/2.$
	660323		
	HLT		
	DAC	BC	
	641002		
	TAD	.-3	
	RAR		
	ISZ	AC	

PAGE 17 FLOATING POINT ARITHMETIC

JMP $\cdot -14$
DAC AC+2 / HIGH ORDER ROOT.
750020
DAC BC+1 / EXTEND PRECISION
LAC BC / TO LOW ORDER
640323 / WITH ROUNDING.
HLT
641002
RCR
TAD BC+1
JMP SQO
END

END F1

PAGE 18 FLOATING POINT ARITHMETIC

	.TITLE	FPP2 TRIGONOMETRY
/ ANCR 1039		
	.GLOBL	ATN,COS,SIN
ATN	HLT	/ ARCTANGENT.
	NOP	
	LAC	AC
	DAC	COS
	DZM	AC
	LAC	AC+1
	754302	
	RAL	
	DAC	SIN
	SNA	
	JMP	+.6
	JMS	EIM
	DAC	COS-3
	LAW	1
	ISZ	COS-3
	NOP	
	JMS	SVMD
	SAD	PIOT-6
	JMP	+.11
	OPR	2000
	DAC	COS-3
	XCT	PIOT-6
	OPR	12001
	DAC	COS-6
	LAC	PIOT-6
	TAD	COS-3
	ISZ	COS-6
	DAC	COS-3
	OPR	1000
	DAC	COS-6
	OPR	13015
	OPR	11044
	OPR	12013
	XOR	COS-6
	OPR	11031
	OPR	12011
	XOR	COS-6
	OPR	11020
	OPR	12007
	XOR	COS-6
	OPR	11011
	OPR	12005
	XOR	COS-6
	OPR	11004
	OPR	12003
	XOR	COS-6
	OPR	12001

PAGE 19 FLOATING POINT ARITHMETIC

	XOR	COS-3	
	OPR	5000	
	ADD	PIOT-3	
	NOP		
	NOP		
	LAC	SIN	
	DAC	AC	
	SNA		
	JMP	.+4	
	JMS	EIM	
	ADD	PIOT	
	NOP		
	JMS	RSMD	
	LAC	COS	
	DAC	AC	
	LAC	ATN	
	JMP	SRTN	
	463201		
	324047		
	-1		
	552421		
	711037		
	-1		
PIOT	552421		
	311037		
	1		
	.BLOCK	6	
COS	HLT		/ COSINE.
	NOP		
	DZM	AC	
	DZM	COS-6	
	LAC	COS	
	DAC	SIN	
	JMS	SVMD	
	ADD	PIOT	
	JMP	.+7	
SIN	HLT		/ SINE.
	NOP		
	LAC	AC	
	DAC	COS-6	
	DZM	AC	
	JMS	SVMD	
	ISZ	PIOT	
	DAC	COS-3	
	NOP		
	JMS	FXQ	
	RCR		
	DAC	COS	
	JMS	CIA	
	JMS	FLOAT	

PAGE 20 FLOATING POINT ARITHMETIC

ADD	COS-3
NOP	
LAC	COS
RAR	
SNL	
JMP	.+4
JMS	EIM
OPR	12777
NOP	
LAC	COS
RTR	
GLK	
XOR	COS-6
DAC	AC
JMS	EIM
XCT	PIOT
DAC	COS-3
OPR	1000
DAC	COS-6
LAW	1
NOP	
LAC	(16)
DAC	.+3
744001	
653122	
HLT	
640611	
LAW	753
640611	
DAC	.+3
JMS	EIM
XCT	COS-6
HLT	
OPR	12001
NOP	
-2	
TAD	.-13
SZA	
JMP	.-20
JMS	EIM
XCT	COS-3
NOP	
JMS	RSMD
LAC	SIN
JMP	SRTN
.END	

END F2

PAGE 21 FLOATING POINT ARITHMETIC

	.TITLE	FPP3 E-FUNCTIONS
/ ANCR 1039		
/		
LOG	.GLOBL	EXP, LOG
	HLT	/ NATURAL LOGARITHM.
	NOP	
	JMS	ACZ
	JMP	LOO
	DZM	AC
	LAC	AC+1
	RCL	
	TAD	NEG1
	DAC	DVTN
	LAC	(1)
	DAC	AC+1
	JMS	SVMD
	XCT	EXP-14
	OPR	12002
	DAC	EXP-3
	OPR	12774
	ISZ	EXP-3
	DAC	EXP-3
	OPR	1000
	DAC	EXP-6
	OPR	11011
	OPR	13013
	OPR	12001
	XCT	EXP-6
	OPR	11007
	OPR	13011
	OPR	12001
	XCT	EXP-6
	OPR	11005
	OPR	13007
	OPR	12001
	XCT	EXP-6
	OPR	11003
	OPR	13005
	OPR	12001
	XCT	EXP-6
	OPR	13003
	OPR	12001
	OPR	11002
	XCT	EXP-3
	DAC	EXP-3
	NOP	
	LAC	DVTN
	JMS	FLOAT
	OPR	13002
	ISZ	EXP-11

PAGE 22 FLOATING POINT ARITHMETIC

	ADD	EXP-3	
	NOP		
	JMS	RSMD	
LOO	LAC	LOG	
	JMP	SRTN	
	714640		
	265011		
	1		
	354513		
	270524		
	1		
	2		
	33		
	444		
	5555		
	66666		
	777777		
EXP	HLT		/ NATURAL EXPONENT.
	NOP		
	JMS	SVMD	
	XCT	EXP-11	
	DAC	EXP-3	
	JMS	FIXL	
	DAC	MPTN	
	JMS	CIA	
	JMS	FLOAT	
	ADD	EXP-3	
	ISZ	EXP-11	
	DAC	EXP-3	
	OPR	13011	
	OPR	12001	
	NOP		
	-10		
	DAC	DVTN	
	LAC	.-5	
	DAC	.-4	
	JMP	.-6	
	JMS	EIM	
	XCT	EXP-3	
	HLT		
	OPR	12001	
	NOP		
	CLC		
	TAD	.-4	
	DAC	.-5	
	ISZ	DVTN	
	JMP	.-11	
	JMS	EIM	
	XCT	EXP-3	
	OPR	12001	

PAGE 23 FLOATING POINT ARITHMETIC

NOP	
LAC	MPTN
TAD	AC+1
DAC	AC+1
JMS	RSMD
LAC	EXP
JMP	SRTN
.END	

END F3

PAGE 24 FLOATING POINT ARITHMETIC

```

        .TITLE      FPP4F FIXED POINT OUTPUT
        .GLOBL      FOF
/ CALLING SEQUENCE
/      JMS*         FOF
/      .DSA         LB      / IOPS ASCII LINE BUFFER
/      .DSA         FD      / DATA STARTING ADDRESS
/      FMT          / FORMAT INDICATOR *
/      ANY          / RETURN LOCATION
/ * FORMAT INDICATOR IS A SIX-DIGIT OCTAL
/   NUMBER OF THE FORM XXYYZZ, WHERE
/       XX IS THE NUMBER OF NUMBERS
/       YY IS THE WIDTH OF THE INTEGER PART
/       ZZ IS THE WIDTH OF THE FRACTION PART
/ FIXED POINT IOPS ASCII OUTPUT
/ W. R. MYERS
/ SCIENTIFIC DATA SYSTEMS
/ AEROJET NUCLEAR COMPANY
/ ANCR 1039
/ OCTOBER 1971
/
FOF      HLT
        LAC*         FOF      / GET LINE BUFFER ADDRESS.
        DAC          AO
        ISZ          FOF
        LAC*         FOF      / GET DATA START ADDRESS.
        DAC          AO+1
        ISZ          FOF
        LAC*         FOF      / PICK UP AND DECOMPOSE
        653606        / FORMAT INDICATOR.
        JMS          CIA
        DAC          AO+2      / DATA COUNT.
        641606
        JMS          CIA      / 'MANTISSA' WIDTH INCLUDES
        DAC          AO+10     / SPACE FOR SIGN.
        641606
        JMS          CIA      / 'FRACTION' WIDTH INCLUDES
        DAC          AO+11     / SPACE FOR DECIMAL POINT.
        ISZ          FOF
        SMA
        CLC          / CALCULATE ROUNDING TERM
        DAC          OUT      / FROM FRACTION FIELD WIDTH.
        JMS          SVM0     / ENTER INTERPRETER.
        LAW          5        / 5.0 TO FAC.
        NOP          / LEAVE INTERPRETER.
        JMS          DVTN     / DIVIDE BY TEN AND
        ISZ          OUT      / REPEAT UNTIL ROUNDING
        JMP          -2        / TERM IS 5*10**(-ZZ).
        JMS          EIM      / SAVE FOR LATER USE.
        DAC          DGO-6
        JMS          RSMD

```

PAGE 25 FLOATING POINT ARITHMETIC

	-5		/ SET UP 5/7 ASCII
	DAC	AO+5	/ BUFFER COUNTER
	LAC	BO-2	/ AND POINTER.
	DAC	BO-1	
	LAC	(42002)	
	DAC*	AO	/ PRESET LINE BUFFER
	ISZ	AO	/ HEADER WORD PAIR.
	DZM*	AO	
	ISZ	AO	
	JMP	.+12	/ JUMP OVER FINISH TEST.
TDO	JMS	RSMD	/ RESTORE USER DATA MODE.
	ISZ	AO+2	/ OUTPUT FINISHED ?
	JMP	.+7	/ NO.
	LAW	175	/ YES, INSERT EOR ALT-MODE
	JMS	OUT	/ IN OUTPUT BUFFER AND
	-5		/ FILL LAST WORD PAIR.
	SAD	AO+5	
	JMP*	FOF	/ RETURN.
	JMP	.-5	
	LAC	AO+10	
	DAC	AO+3	
	LAC	AO+11	
	DAC	AO+4	
	JMS	EIM	/ CONTINUE OUTPUT.
	LAC*	AO+1	/ GET FLOATING NUMBER.
	CAL	AO+1	/ INCREMENT POINTER.
	JMS	SVMD	/ SAVE DATA MODE, SET TO 3.
	OPR	5000	/ TEST POSITIVE OR
	TAD	DGO-6	/ NEGATIVE, ADD OR
	NOP		/ SUBTRACT ROUNDING TERM.
	ADD	DGO-6	
	NOP		/ LEAVE INTERPRETER.
	JMS	ACZ	/ IS FAC EQUAL ZERO ?
	SKP		/ YES.
	JMP	.+4	/ NO.
	LAW	260	/ YES, SET 'SIGN' TO '0'
	DAC	AO+6	/ AND GO TO FRACTION.
	JMP	FRO-6	
	LAC	AC	/ FAC NOT ZERO,
	RAR		/ SAVE SIGN AND
	LAW	240	/ SET POSITIVE.
	SZL		
	LAW	255	
	DAC	AO+6	
	DZM	AC	
	LAC	AC+1	
	740300		/ IS FAC LESS THAN ONE ?
	JMP	FRO+12	/ NO, GO PROCESS INTEGER PART.
	LAC	(1)	/ YES, OUTPUT YY-1 BLANKS
	TAD	AO+3	/ AND SIGN OR ZERO.

FRO	SPA		
	JMS	SPAC	
	LAC	AO+6	
	JMS	OUT	
	LAC	AO+4	/ IS FRACTION WIDTH ZERO ?
	SMA		/ NO.
	JMP	TDO	/ YES, TEST FOR OUTPUT DONE.
	LAW	256	/ NO, PRINT DECIMAL POINT.
	JMS	OUT	
	ISZ	AO+4	/ IS FRACTION FINISHED ?
	SKP		
	JMP	TDO	/ YES, TEST FOR OUTPUT DONE.
	JMS	DGO	/ NO, OUTPUT A DIGIT.
	JMP	.-4	/ LOOP BACK TO TEST.
	LAC	AO+10	
	RCL		
	TAD	AO+10	
	TAD	AC+1	
	SMA		
	JMP	.-+13	
	DZM	AO+7	/ FAC .GE. ONE,
	JMS	DVTN	/ DIVIDE BY TEN AND COUNT
	ISZ	AO+7	/ UNTIL LESS THAN ONE.
	LAC	AC+1	
	740300		
	JMP	.-4	
	LAC	AO+3	/ NUMBER TOO LARGE
	TAD	AO+7	/ FOR FIELD ?
	SPA		
	JMP	.-+13	/ NO.
	LAC	AO+3	/ YES, FILL NUMBER
	TAD	AO+4	/ FIELD WITH PERIODS.
	DAC	AO+3	
	LAW	240	
	SKP		
	LAW	256	
	JMS	OUT	
	ISZ	AO+3	
	JMP	.-3	
	JMP	TDO	
	TAD	(1)	
	SPA		
	JMS	SPAC	
	LAC	AO+7	
	CMA		
	DAC	AO+7	
	LAC	AO+6	/ OUTPUT SIGN.
	JMS	OUT	
	ISZ	AO+7	/ TEST FOR MANTISSA DONE.
	SKP		

PAGE 27 FLOATING POINT ARITHMETIC

	JMP	FRO	/ YES, GO PROCESS FRACTION.
	JMS	DGO	/ NO, PRINT A DIGIT.
	JMP	.-4	/ LOOP BACK TO TEST.
	.BLOCK	6	
DGO:	HLT		/ DIGIT PRINT ROUTINE.
	JMS	MPTN	/ MULTIPLY BY TEN
	JMS	EIM	/ ENTER INTERPRETER.
	DAC	DGO-3	/ SAVE 10*FAC.
	NOP		/ LEAVE INTERPRETER.
	JMS	FXQ	/ TRUNCATE FAC TO HIGH
	RCR		/ ORDER DIGIT, THEN
	DAC	SPAC	/ SAVE FOR MODIFICATION:
	TAD	FRO-23	/ CONVERT TO ASCII
	JMS	OUT	/ OUTPUT.
	LAC	SPAC	
	JMS	CIA	/ FLOAT TRUNCATED NUMBER
	JMS	FLOAT	/ AND SUBTRACT FROM
	ADD	DGO-3	/ 10.0 * PREVIOUS FAC.
	NOP		
	JMP*	DGO	
SPAC	HLT		
	DAC	DGO	/ OUTPUT N BLANKS.
	LAW	240	/ -N IN AC.
	JMS	OUT	
	ISZ	DGO	
	JMP	.-3	
	JMP*	SPAC	
OUT	HLT		/ ASCII PACK ROUTINE.
	AND	(177)	/ MASK TO 7 BITS AND
	DAC*	BO-1	/ PUT IN 'FIVE' BUFFER.
	ISZ	BO-1	/ INCREMENT POINTER.
	ISZ	AO+5	/ INCREMENT COUNTER.
	JMP*	OUT	/ RETURN OR
	-5		/ RESTORE POINTER
	DAC	AO+5	/ AND COUNTER, THEN
	LAC	BO-2	/ CONTINUE ON TO PACK.
	DAC	BO-1	
	CLL		
	LAC	BO+4	/ PACK FIVE CHARACTERS
	650507		/ IN TWO WORDS.
	LAC	BO+3	
	640507		
	LAC	BO	
	640707		
	XOR	BO+1	
	640704		
	DAC	BO	
	LAC	BO+2	
	640503		
	XOR	BO	

PAGE 28 FLOATING POINT ARITHMETIC

	DAC*	AO	/ INSERT IN LINE BUFFER.
	ISZ	AO	
	641002		
	DAC*	AO	
	ISZ	AO	
	JMP*	OUT	/ RETURN.
	BO		
	BO		
BO	.BLOCK	5	
AO	.BLOCK	12	
	.END		
\$			END F4M

PAGE 29 FLOATING POINT ARITHMETIC

```

        .TITLE      FPP4FU FIXED POINT OUTPUT
        .GLOBL      FOF
/ CALLING SEQUENCE
/      JMS*        FOF
/      .DSA        UCR      / USER CHARACTER ROUTINE
/      .DSA        FD       / DATA STARTING ADDRESS
/      FMT         / FORMAT INDICATOR *
/      ANY         / RETURN LOCATION
/ * FORMAT INDICATOR IS A SIX-DIGIT OCTAL
/   NUMBER OF THE FORM XXYYZZ, WHERE
/       XX IS THE NUMBER OF NUMBERS
/       YY IS THE WIDTH OF THE INTEGER PART
/       ZZ IS THE WIDTH OF THE FRACTION PART
/ FIXED POINT OUTPUT UTILIZING CHARACTER
/ HANDLING ROUTINE IN CALLING PROGRAM
/ W. R. MYERS
/ SCIENTIFIC DATA SYSTEMS
/ AEROJET NUCLEAR COMPANY
/ ANCR 1039
/ OCTOBER 1971
/
FOF      HLT
        LAC*      FOF      / GET ADDRESS OF USER
        DAC       AO      / CHARACTER ROUTINE.
        ISZ       FOF
        LAC*      FOF      / GET DATA START ADDRESS.
        DAC       AO+1
        ISZ       FOF
        LAC*      FOF      / PICK UP AND DECOMPOSE
        653606    / FORMAT INDICATOR.
        JMS       CIA
        DAC       AO+2    / DATA COUNT.
        641606
        JMS       CIA
        DAC       AO+7    / 'MANTISSA' WIDTH INCLUDES
        641606    / SPACE FOR SIGN.
        JMS       CIA
        DAC       AO+10   / 'FRACTION' WIDTH INCLUDES
        ISZ       FOF    / SPACE FOR DECIMAL POINT.
        SMA
        CLC
        DAC       AO+5    / CALCULATE ROUNDING TERM
        JMS       SVMD    / FROM FRACTION FIELD WIDTH.
        LAW       5       / ENTER INTERPRETER.
        NOP      / 5.0 TO FAC.
        JMS       DVTN    / LEAVE INTERPRETER.
        ISZ       AO+5    / DIVIDE BY TEN AND
        JMP       -2      / REPEAT UNTIL ROUNDING
        JMS       EIM     / TERM IS 5*10**(-ZZ).
        DAC       DGO-6   / SAVE FOR LATER USE.

```

	JMS	RSMD	
	JMP	.+5	/ JUMP OVER FINISH TEST.
TDO	JMS	RSMD	/ RESTORE USER DATA MODE.
	ISZ	AO+2	/ OUTPUT FINISHED ?
	SKP		/ NO.
	JMP*	FOF	/ YES, RETURN.
	LAC	AO+7	
	DAC	AO+3	
	LAC	AO+10	
	DAC	AO+4	
	JMS	EIM	/ CONTINUE OUTPUT.
	LAC*	AO+1	/ GET FLOATING NUMBER.
	CAL	AO+1	/ INCREMENT POINTER.
	JMS	SVMD	/ SAVE DATA MODE, SET TO 3.
	OPR	5000	/ TEST POSITIVE OR
	TAD	DGO-6	/ NEGATIVE, ADD OR
	NOP		/ SUBTRACT ROUNDING TERM.
	ADD	DGO-6	
	NOP		/ LEAVE INTERPRETER.
	JMS	ACZ	/ IS FAC EQUAL ZERO ?
	SKP		/ YES.
	JMP	.+4	/ NO.
	LAW	260	/ YES, SET 'SIGN' TO '0'
	DAC	AO+5	/ AND GO TO FRACTION.
	JMP	FRO-6	
	LAC	AC	/ FAC NOT ZERO,
	RAR		/ SAVE SIGN AND
	LAW	240	/ SET POSITIVE.
	SZL		
	LAW	255	
	DAC	AO+5	
	DZM	AC	
	LAC	AC+1	
	740300		/ IS FAC LESS THAN ONE ?
	JMP	FRO+12	/ NO, GO PROCESS INTEGER PART.
	LAC	(1)	/ YES, OUTPUT YY-1 BLANKS
	TAD	AO+3	/ AND SIGN OR ZERO.
	SPA		
	JMS	SPAC	
	LAC	AO+5	
	JMS*	AO	
FRO	LAC	AO+4	/ IS FRACTION WIDTH ZERO ?
	SMA		
	JMP	TDO	/ YES, TEST OUTPUT DONE.
	LAW	256	/ NO, PRINT DECIMAL POINT.
	JMS*	AO	
	ISZ	AO+4	/ IS FRACTION FINISHED ?
	SKP		
	JMP	TDO	/ YES, TEST OUTPUT DONE.
	JMS	DGO	/ NO, OUTPUT A DIGIT.

PAGE 31 FLOATING POINT ARITHMETIC

JMP	.-4	/ LOOP BACK TO TEST.
LAC	AO+7	
RCL		
TAD	AO+7	
TAD	AC+1	
SMA		
JMP	.-+13	
DZM	AO+6	/ FAC .GE. ONE,
JMS	DVTN	/ DIVIDE BY TEN AND COUNT
ISZ	AO+6	/ UNTIL LESS THAN ONE.
LAC	AC+1	
740300		
JMP	.-4	
LAC	AO+3	/ NUMBER TOO LARGE FOR
TAD	AO+6	/ FIELD ?
SPA		
JMP	.-+13	/ NO.
LAC	AO+3	
TAD	AO+4	/ YES, FILL NUMBER
DAC	AO+3	/ FIELD WITH PERIODS.
LAW	240	
SKP		
LAW	256	
JMS*	AO	
ISZ	AO+3	
JMP	.-3	
JMP	TDO	
TAD	(1)	
SPA		
JMS	SPAC	
LAC	AO+6	
CMA		
DAC	AO+6	
LAC	AO+5	/ OUTPUT SIGN.
JMS*	AO	
ISZ	AO+6	/ TEST MANTISSA DONE.
SKP		
JMP	FRO	/ YES, GO PROCESS FRACTION.
JMS	DGO	/ NO, PRINT A DIGIT.
JMP	.-4	/ LOOP BACK TO TEST.
.BLOCK	6	
HLT		
JMS	MPTN	/ DIGIT SEND ROUTINE.
JMS	EIM	
DAC	DGO-3	/ SAVE 10.0*FAC
NOP		
JMS	FXQ	/ TRUNCATE TO HIGH
RCR		/ ORDER DIGIT,
DAC	SPAC	/ SAVE.
TAD	FRO-23	/ CONVERT TO ASCII

DGO

PAGE 32 FLOATING POINT ARITHMETIC

	JMS*	AO	/ AND SEND TO USER.
	LAC	SPAC	
	JMS	CIA	/ FLOAT TRUNCATED NUMBER
	JMS	FLOAT	/ AND SUBTRACT FROM
	ADD	DGO-3	/ 10.0 * PREVIOUS FAC.
	NOP		
	JMP*	DGO	
SPAC	HLT		
	DAC	DGO	/ SEND N BLANKS.
	LAW	240	/ -N IN AC.
	JMS*	AO	
	ISZ	DGO	
	JMP	.-3	
	JMP*	SPAC	
AO	.BLOCK	11	
	.END		
\$			END F4U

	.TITLE	FPP5	FLOATING POINT INPUT	
	.GLOBL	FIN		
/	CALLING SEQUENCE			
/	JMS*	FIN		
/	.DSA	LB	/	IOPS ASCII LINE BUFFER
/	.DSA	FD	/	DATA INPUT BUFFER
/	-N		/	NUMBER OF NUMBERS
/	JMP	ERR	/	ERROR RETURN
/	ANY		/	NORMAL RETURN
/	FREE FORMAT IOPS ASCII INPUT			
/	W. R. MYERS			
/	SCIENTIFIC DATA SYSTEMS			
/	AEROJET NUCLEAR COMPANY			
/	ANCR 1039			
/	OCTOBER 1971			
/				
FIN	HLT			
	LAC*	FIN		
	DAC	AI	/	GET ASCII BUFFER ADDRESS.
	ISZ	FIN		
	LAC*	FIN		
	DAC	AI+1	/	GET DATA VECTOR START.
	ISZ	FIN		
	LAC*	FIN		
	DAC	AI+2	/	GET DATA COUNT.
	ISZ	FIN		
	LAC*	AI	/	TEST VALIDITY BITS FOR
	AND	(77)	/	ZERO AND DATA MODE
	SAD	(2)	/	FOR IOPS ASCII.
	JMP	.+3		
BI	754003		/	RETURN WITH AC AND LINK
	JMP*	FIN	/	ALL ONES IF BAD INPUT.
	TAD	AI		
	DAC	AI		
	DZM	AI+3	/	CLEAR END OF RECORD.
	DZM	AI+4	/	SET 'BLANK' NONTERMINATOR.
	DZM	AI+5	/	CLEAR 'FIVE' COUNTER.
NUMR	DZM	AC	/	SET UP TO INTERPRET.
	DZM	AC+2		
	DZM	AC+3		
	DZM	BC		
	DZM	BC+1		
	DZM	BC+2		
	DZM	BC+3		
NUM1	JMS	CHAR	/	GET A CHARACTER.
	744100		/	MINUS SIGN ?
	JMP	.+4	/	NO.
	LAC	(1)	/	YES, SET FAC NEGATIVE
	DAC	AC	/	AND GET ANOTHER CHARACTER.
	JMP	NUM1		

	DAC	CHAR	
	LAC	AC+2	/ DIGIT, MULTIPLY CURRENT
	653122		/ DOUBLE WORD MANTISSA
	12		/ BY TEN, AND ADD
	641002		/ NEW DIGIT.
	DAC	AC+2	
	LAC	AC+3	
	653122		
	12		
	TAD	AC+2	
	DAC	AC+2	
	641002		
	TAD	CHAR	
	DAC	AC+3	
	SZL		
	ISZ	AC+2	
	LAC	BC	
	TAD	BC+1	/ UPDATE FRACTION
	DAC	BC	/ COUNTER.
	JMP	NUM1	
DEC1	CLC		/ SET FRACTION INDICATOR.
	DAC	BC+1	
	JMP	NUM1	
EXPN	JMS	CHAR	/ GET CHARACTER FOR EXPONENT.
	744100		/ MINUS SIGN ?
	JMP	.+3	/ NO.
	DAC	BC+3	/ YES, SET INDICATOR.
	JMP	EXPN	
	DAC	CHAR	/ MUST BE DIGIT,
	LAC	BC+2	/ MULTIPLY CURRENT
	653122		/ EXPONENT BY TEN
	12		/ AND ADD NEW DIGIT.
	641002		
	TAD	CHAR	
	DAC	BC+2	
	JMP	EXPN	
INAN	LAC	BC+3	
	RAL		
	LAC	BC+2	/ DETERMINE EFFECTIVE
	745400		/ EXPONENT FROM
	JMS	CIA	/ EXPLICIT EXPONENT
	TAD	BC	/ AND FRACTION COUNT.
	DAC	BC	
	744100		/ SAVE MAGNITUDE AND SENSE.
	JMS	CIA	
	DAC	BC+1	
	LAC	{43}	
	DAC	AC+1	/ NORMALIZE ACCUMULATED
	JMS	NOR	/ EFFECTIVE MANTISSA.
	LAC	BC	

PAGE 35 FLOATING POINT ARITHMETIC

	741210		/ IS NET EXPONENT ZERO ?
	JMP	+.10	/ YES.
	LAC	AI-1	/ NO, SET UP TO MULTIPLY
	SZL		/ OR DIVIDE BY TEN.
	LAC	AI-2	
	DAC	+.1	
	HLT		/ MULTIPLY OR DIVIDE FAC BY
	ISZ	BC+1	/ TEN UNTIL EFFECTIVE
	JMP	.-2	/ EXPONENT IS ZERO.
	JMS	EIM	/ ENTER FLOATING MODE.
	DAC*	AI+1	/ STORE FAC IN USER WORD.
	CAL	AI+1	/ INCREMENT POINTER.
	NOP		/ LEAVE FLOATING MODE.
	LAC	AI+3	
	ISZ	AI+2	/ INCREMENT DATA COUNT.
	JMP	+.7	
	755200		/ DONE, TEST END OF RECORD.
	JMP	+.3	
	ISZ	FIN	/ OK, BUMP RETURN AND
	JMP*	FIN	/ RETURN L=0, AC=0.
	754002		/ LONG RECORD, ERROR
	JMP*	FIN	/ RETURN L=1, AC=0.
	755201		/ NOT DONE, TEST END OF RECORD.
	JMP	NUMR	/ NOT EOR, INTERPRET MORE DATA.
	LAC	AI+2	/ SHORT RECORD, ERROR
	JMS	CIA	/ RETURN L=0, AC=SHORTAGE.
	JMP*	FIN	
	JMS	DVTN	
	JMS	MPTN	
AI	.BLOCK	13	
	AI+6		
	AI+6		
CHAR	HLT		/ CHARACTER PROCESSOR.
	LAC	AI+5	
	744200		/ UNPACK REQUIRED ?
	JMP	+.26	/ NO.
	LAC*	AI	/ YES, PICK UP
	653607		/ FIVE CHARACTERS.
	DAC	AI+6	
	641607		
	DAC	AI+7	
	641607		
	DAC	AI+10	
	ISZ	AI	
	LAC*	AI	
	653603		
	XOR	AI+10	
	DAC	AI+10	
	641607		
	DAC	AI+11	


```

641607
DAC      AI+12
ISZ      AI
LAC      CHAR-2      / SET POINTER TO
DAC      CHAR-1      / UNPACKED CHARACTERS.
-5       / SET CHARACTER
DAC      AI+5        / COUNTER TO FIVE.
LAC*     CHAR-1      / PICK UP CHARACTER.
ISZ      AI+5        / INCREMENT COUNTER.
ISZ      CHAR-1      / INCREMENT POINTER.
SAD      (40)        / BLANK IS TERMINATOR IF
JMP      TERM-5      / FOLLOWS NON-BLANK
745200   / NON-TERMINATOR.
JMP      CHAR+1      / IGNORE NULL.
SAD      INAN-5      / IGNORE LINE FEED.
JMP      CHAR+1
SAD      (53)        / IGNORE 'PLUS'.
JMP      CHAR+1
SAD      (177)       / IGNORE RUB-OUT.
JMP      CHAR+1
SAD      (15)        / 'CR' EOR AND TERMINATOR.
JMP      TERM-1
SAD      (175)       / ALT-MODE EOR AND TERMINATOR.
JMP      TERM-1
SAD      (54)        / COMMA TERMINATOR.
JMP      TERM
SAD      (11)        / TAB TERMINATOR.
JMP      TERM-5
DAC      AI+4        / SET BLANK TO TERMINATE.
SAD      (105)       / IF 'E', GO TO EXPONENT.
JMP      EXPN
SAD      (56)        / IF '.', SET INDICATOR.
JMP      DEC1
SAD      (55)        / IF 'MINUS', RETURN FLAG.
751001
SKP
JMP*     CHAR
640504   / TEST FOR CHARACTER CODE
SAD      (3)        / BETWEEN 60 AND 71,
SKP      / ERROR EXIT IF NOT.
JMP      BI
641604
652000
CMA
TAD      INAN-5
755100
JMP      BI
641002   / RETURN FOUR-BIT DIGIT.
JMP*     CHAR
LAC      AI+4

```

PAGE 37 FLOATING POINT ARITHMETIC

	744200		/ IGNORE BLANK IF
	JMP	.+3	/ NOT TERMINATOR.
	JMP	CHAR+1	
	ISZ	AI+3	/ SET EOR FLAG.
TERM	DZM	AI+4	/ SET BLANK NONTERMINATOR.
	JMP	INAN	/ GO TO ANALYZE.
	.END		
\$			END F5M

```

        .TITLE      FPP5U FLOATING POINT INPUT
        .GLOBL      FIN
/ CALLING SEQUENCE
/      JMS*         FIN
/      .DSA         UCR      / USER CHARACTER ROUTINE
/      .DSA         FD       / DATA START ADDRESS
/      -N           / NUMBER OF NUMBERS
/      JMP          ERR      / ERROR RETURN
/      ANY          / NORMAL RETURN
/ FREE FORMAT INPUT
/ CALLING PROGRAM SUPPLIES ALL
/ CHARACTERS THROUGH 'UCR' ROUTINE
/ W. R. MYERS
/ SCIENTIFIC DATA SYSTEMS
/ AEROJET NUCLEAR COMPANY
/ ANCR 1039
/ OCTOBER 1971
/
FIN      HLT
        LAC*         FIN      / GET USER CHARACTER
        DAC          AI      / ROUTINE ENTRY POINT.
        ISZ          FIN
        LAC*         FIN      / GET DATA START ADDRESS.
        DAC          AI+1
        ISZ          FIN
        LAC*         FIN      / GET DATA COUNT.
        DAC          AI+2
        ISZ          FIN
        DZM          AI+3      / CLEAR END OF RECORD.
        DZM          AI+4      / SET BLANK NONTERMINATOR.
NUMR     DZM          AC       / SET UP TO INTERPRET.
        DZM          AC+2
        DZM          AC+3
        DZM          BC
        DZM          BC+1
        DZM          BC+2
        DZM          BC+3
NUM1     JMS         CHAR      / GET A CHARACTER.
        744100       / MINUS SIGN ?
        JMP          .+4      / NO.
        LAC          (1)      / YES, SET FAC NEGATIVE
        DAC          AC       / AND GET ANOTHER CHARACTER.
        JMP         NUM1
        DAC         CHAR
        LAC         AC+2
        653122       / DIGIT, MULTIPLY CURRENT
        12           / DOUBLE WORD MANTISSA
        641002       / BY TEN, AND ADD
        DAC         AC+2      / NEW DIGIT.
        LAC         AC+3

```

	653122		
	12		
	TAD	AC+2	
	DAC	AC+2	
	641002		
	TAD	CHAR	
	DAC	AC+3	
	SZL		
	ISZ	AC+2	
	LAC	BC	
	TAD	BC+1	/ UPDATE FRACTION
	DAC	BC	/ COUNTER.
	JMP	NUM1	
DEC1	CLC		/ SET FRACTION INDICATOR.
	DAC	BC+1	
	JMP	NUM1	
EXP1	JMS	CHAR	/ GET CHARACTER FOR EXPONENT.
	744100		/ MINUS SIGN ?
	JMP	.+3	/ NO.
	DAC	BC+3	/ YES, SET INDICATOR.
	JMP	EXP1	
	DAC	CHAR	/ MUST BE DIGIT,
	LAC	BC+2	/ MULTIPLY CURRENT
	653122		/ EXPONENT BY TEN
	12		/ AND ADD NEW DIGIT.
	641002		
	TAD	CHAR	
	DAC	BC+2	
	JMP	EXP1	
INAN	LAC	BC+3	
	RAL		
	LAC	BC+2	/ DETERMINE EFFECTIVE
	745400		/ EXPONENT FROM
	JMS	CIA	/ EXPLICIT EXPONENT
	TAD	BC	/ AND FRACTION COUNT.
	DAC	BC	
	744100		/ SAVE MAGNITUDE AND SENSE.
	JMS	CIA	
	DAC	BC+1	
	LAC	(43)	
	DAC	AC+1	/ NORMALIZE ACCUMULATED
	JMS	NOR	/ EFFECTIVE MANTISSA.
	LAC	BC	
	741210		/ IS NET EXPONENT ZERO ?
	JMP	.+10	/ YES.
	LAC	AI-1	/ NO, SET UP TO MULTIPLY
	SZL		/ OR DIVIDE BY TEN.
	LAC	AI-2	
	DAC	.+1	
	HLT		/ MULTIPLY OR DIVIDE FAC BY

	ISZ	BC+1	/ TEN UNTIL EFFECTIVE
	JMP	.-2	/ EXPONENT IS ZERO.
	JMS	EIM	
	DAC*	AI+1	/ STORE FAC IN USER WORD.
	CAL	AI+1	/ INCREMENT POINTER.
	NOP		
	LAC	AI+3	
	ISZ	AI+2	/ INCREMENT WORD COUNT.
	JMP	+.7	
	755200		/ DONE, TEST END OF RECORD.
	JMP	+.3	
	ISZ	FIN	/ OK, MAKE NORMAL RETURN
	JMP*	FIN	/ WITH L=0, AC=0.
	754002		/ LONG RECORD, ERROR
	JMP*	FIN	/ RETURN L=1, AC=0.
	755201		/ NOT DONE, TEST EOR.
	JMP	NUMR	/ NOT EOR, INTERPRET MORE DATA.
	LAC	AI+2	/ SHORT RECORD, ERROR
	JMS	CIA	/ RETURN L=0, AC=SHORTAGE.
	JMP*	FIN	
BI	754003		/ BAD FORMAT, ERROR
	JMP*	FIN	/ RETURN L=1, AC=777777.
	JMS	DVTN	
	JMS	MPTN	
AI	.BLOCK	5	
RI	-3		/ SIMULATE RE-ENTRY
	TAD	FIN	/ TO 'FIN' IF CNTRL-U
	DAC	FIN	/ IN INPUT RECORD.
	JMP	FIN+1	
CI	LAC	AI+4	/ TEST BLANK FOR TERMINATOR.
	744200		
	JMP	+.3	
	JMP	CHAR+1	
TERM	ISZ	AI+3	/ SET EOR IF CR OR ALT-MODE.
	DZM	AI+4	/ SET BLANK NONTERMINATOR.
	JMP	INAN	/ GO TO ANALYZE.
CHAR	HLT		
	JMS*	AI	/ GET USER CHARACTER.
	AND	(177)	/ MASK TO SEVEN BITS.
	SAD	(40)	/ TEST FOR BLANK.
	JMP	CI	
	745200		/ IGNORE NULL.
	JMP	CHAR+1	
	SAD	INAN-5	/ IGNORE LINE FEED.
	JMP	CHAR+1	
	SAD	(53)	/ IGNORE 'PLUS'.
	JMP	CHAR+1	
	SAD	(177)	/ RESTART ON RUB-OUT.
	JMP	NUMR-1	
	SAD	(15)	/ CARRIAGE RETURN AND

PAGE 41 FLOATING POINT ARITHMETIC

JMP	TERM	/ ALT-MODE TERMINATE
SAD	(175)	/ AND END OF RECORD.
JMP	TERM	
SAD	(25)	/ CNTRL-U RESTARTS
JMP	RI	/ INPUT PROCESSOR.
SAD	(54)	/ COMMA AND TAB
JMP	TERM+1	/ TERMINATORS.
SAD	(11)	
JMP	CI	
DAC	AI+4	/ SET BLANK TERMINATOR.
SAD	(105)	/ IF 'E', GO TO
JMP	EXPONENT	/ EXPONENT.
SAD	(56)	/ IF DECIMAL, SET
JMP	DEC1	/ INDICATOR.
SAD	(55)	/ IF MINUS RETURN FLAG.
751001		
SKP		
JMP*	CHAR	
640504		/ TEST FOR CODES
SAD	(3)	/ 60 THROUGH 71,
SKP		/ ERROR RETURN IF NOT.
JMP	BI	
641604		
652000		
CMA		
TAD	INAN-5	
755100		
JMP	BI	/ RETURN FOUR BIT DIGIT.
641002		
JMP*	CHAR	
END		
		END F5U

PAGE 42 FLOATING POINT ARITHMETIC

	.TITLE	FPP6 EXTENDED FUNCTIONS	
/ ANCR 1039			
/ RANDOM NUMBER			
FRAN	.GLOBL	FRAN	
	0		/ RETURNS A RANDOM NUMBER
	NOP		/ IN THE FLOATING ACCUMULATOR.
	LAC	(21)	
	673122		/ RANGE IS MINUS ONE TO ONE.
	265341		
	641002		
	TAD	(43)	
	DAC	.-3	
	641601		
	DAC	AC	
	DZM	AC+1	
	641002		
	DAC	AC+3	
	RCR		
	640444		
	RTL		
	RCR		
	DAC	AC+2	
	JMS	NOR	
	LAC	FRAN	
	JMP	SRTN	
/ ARC SINE AND ARC COSINE			
	.GLOBL	ARCOS, ARSIN	
ARCOS	0		/ ARC COSINE OF FAC.
	NOP		
	JMS	ARSIN	
	OPR	3000	
	JMS	SVMD	
	ADD	PIOT	
	JMS	RSMD	
	LAC	ARCOS	
	JMP	SRTN	
ARSIN	0		/ ARC SINE OF FAC.
	JMS	SVMD	
	DAC	TM6	
	OPR	1000	
	OPR	5000	
	JMP	+.2	
	JMP	+.16	
	OPR	3000	
	OPR	12001	
	OPR	5000	
	JMP	+.7	
	JMP	+.6	
	OPR	6000	
	XOR	TM6	

```

JMS      RSMD
JMS      ATN
JMP*     ARSIN
LAC      TM6
OPR      4000
XCT      PIOT
JMS      RSMD
LAC      ARSIN
JMP      SRTN
TM6      .BLOCK 3
/ ARCTANGENT OF TWO ARGUMENTS
.GLOBL   ATN2
LAC*     ATN2-1      / RETURN PLUS OR MINUS PI/2
OPR      4000        / IF X IS ZERO.
JMP      .+5
LAC*     ATN2-2      / RETURN ZERO OR PI
OPR      4000        / IF Y IS ZERO.
OPR      12777
OPR      2000
XCT      PIOT
JMS      RSMD
LAC      ATN2
JMP      SRTN
HLT      / ADDRESS OF X.
HLT      / ADDRESS OF Y.
ATN2     HLT
NOP      / LEAVE INTERPRETER.
LAC*     ATN2
DAC      ATN2-2      / GET X ADDRESS
ISZ      ATN2        / (FIRST ARGUMENT).
LAC*     ATN2        / GET Y ADDRESS
DAC      ATN2-1      / (SECOND ARGUMENT).
ISZ      ATN2
JMS      SVMMD       / SAVE MODE, SET TO 3.
LAC*     ATN2-1
OPR      5000        / TEST Y FOR ZERO.
JMP      .+2
JMP      ATN2-12
LAC*     ATN2-2
OPR      5000        / TEST X FOR ZERO.
JMP      .+2
JMP      ATN2-15
OPR      2000        / ABS(X)
XOR*     ATN2-1      / Y/ABS(X)
JMS      RSMD
JMS      ATN        / ATN(Y/ABS(X)) IS AN ANGLE
JMS      SVMMD      / BETWEEN -PI/2 AND PI/2.
DAC      TM6
LAC*     ATN2-2      / DETERMINE CORRECT QUADRANT
OPR      5000        / FROM SIGN OF X.

```


PAGE 44 FLOATING POINT ARITHMETIC

```

      JMP      .+4
      JMP      ATN2-15
      LAC      TM6      / IF X POSITIVE, NO
      JMP      ATN2-5    / CORRECTION REQUIRED.
      LAC      TM6
      OPR      4000      / IF X NEGATIVE, MAKE
      XCT      PIOT      / CORRECTION OF
      OPR      11002     / PLUS OR MINUS PI.
      TAD      TM6
      JMP      ATN2-5

/ COMMON LOGARITHM
      .GLOBL   LOG10
LOG10  0      / COMMON LOGARITHM.
      NOP
      JMS      LOG
      JMS      SVMD
      XCT      .+4
      JMS      RSMD
      LAC      LOG10
      JMP      SRTN
      542520
      336267
      777777
/ GENERAL EXPONENT
      .GLOBL   GXP
GXP    0      / GENERAL EXPONENT.
      NOP      / LEAVE INTERPRETER.
      LAC*     GXP      / GET ADDRESS OF BASE
      DAC      LOG      / (FIRST ARGUMENT).
      ISZ      GXP
      LAC*     GXP      / GET EXPONENT ADDRESS
      DAC      EXP      / (SECOND ARGUMENT).
      ISZ      GXP
      JMS      EIM
      LAC*     LOG      / NATURAL LOG OF BASE.
      JMS      LOG
      XCT*     EXP      / MULTIPLY BY EXPONENT.
      JMS      EXP      / NATURAL EXPONENT.
      JMP*     GXP
      .END

```

END F6