FLINT 36 A3D

DESCRIPTION AND OPERATING PROCEDURES

For presentation at the annual meeting of The Digital Equipment Computer Users Society, held at the Lawrence Radiation Laboratory, Livermore, California, on November 18 and 19, 1963

by

Jacob M. Baker and David J. Isenberg
CHARLES W. ADAMS ASSOCIATES, INC.
Consultants in Electronic Data Processing
Bedford, Massachusetts
Since FLINT (originally written in FRAP) was released about a year ago by Itek Corporation, through The Digital Equipment Computer Users Society, there has been considerable demand for improved documentation and a revised listing. As a service to DECUS, Adams Associates gladly offered to undertake the conversion and redocumentation of FLINT, and has done so with the permission and assistance of Itek. The results of its work are reported in this paper.

In the near future, new FRAP and MACRO listings will be made available by Adams Associates and other modifications are being considered. Among these are the production of a totally relocateable version of FLINT, the removal of exponent bias, and the addition of other floating-point instructions such as a floating index.

Adams Associates wishes to acknowledge with thanks the substantial contribution made by Edward J. Radkowski of Itek Corporation to the revision of FLINT. Readers of this paper are invited not only to request additional copies of it from Adams Associates but also to forward to the company any suggestions or criticisms. These should be marked to the attention of David J. Isenberg or Jacob M. Baker.
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Instruction Repertoire</td>
<td>1</td>
</tr>
<tr>
<td>Floating Operations</td>
<td>1</td>
</tr>
<tr>
<td>Entering Interpreter</td>
<td>2</td>
</tr>
<tr>
<td>Formats</td>
<td>2</td>
</tr>
<tr>
<td>Unfloating Routine</td>
<td>4</td>
</tr>
<tr>
<td>Input Routines</td>
<td>5</td>
</tr>
<tr>
<td>Output Routines</td>
<td>7</td>
</tr>
<tr>
<td>Description of Instructions</td>
<td>9</td>
</tr>
<tr>
<td>Possible Modifications by Users</td>
<td>11</td>
</tr>
<tr>
<td>Partially Relocateable Version</td>
<td>11</td>
</tr>
<tr>
<td>Expansion of Input Buffer</td>
<td>12</td>
</tr>
<tr>
<td>DECAL Listing</td>
<td>12</td>
</tr>
</tbody>
</table>
Introduction

FLINT is an interpretive routine that permits the Digital Equipment Corporation PDP-1 to perform double-precision floating-point arithmetic, input, output, and elementary function evaluation. Originally written in FRAP for use in lens design work (though nonetheless a general-purpose program), FLINT has now been translated into DECAL to be compatible with other programs in this language. Arithmetic and function evaluation are performed interpretively, input and output are handled by closed subroutines addressed directly by the user's programs, and overall format control is left to the user's routines.

Instruction Repertoire

The instructions currently available for the interpreter are listed below:

Floating Operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>Operation Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deposit floating accumulator</td>
<td>fda</td>
<td>00</td>
</tr>
<tr>
<td>Floating add</td>
<td>fad</td>
<td>02</td>
</tr>
<tr>
<td>Floating subtract</td>
<td>fsu</td>
<td>04</td>
</tr>
<tr>
<td>Load floating accumulator</td>
<td>flo</td>
<td>06</td>
</tr>
<tr>
<td>Floating square root</td>
<td>fsr</td>
<td>24</td>
</tr>
<tr>
<td>Floating sine</td>
<td>fsi</td>
<td>26</td>
</tr>
<tr>
<td>Floating cosine</td>
<td>fco</td>
<td>30</td>
</tr>
<tr>
<td>Floating skip</td>
<td>fsk</td>
<td>32</td>
</tr>
<tr>
<td>Floating multiply</td>
<td>fmu</td>
<td>54</td>
</tr>
<tr>
<td>Floating divide</td>
<td>fdi</td>
<td>56</td>
</tr>
<tr>
<td>Floating operate</td>
<td>fopr</td>
<td>76</td>
</tr>
</tbody>
</table>
Entering Interpreter

<table>
<thead>
<tr>
<th>Function</th>
<th>Mnemonic</th>
<th>Octal Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter interpretive mode</td>
<td><code>cal .</code></td>
<td>160000</td>
</tr>
<tr>
<td>Enter interpretive mode and</td>
<td><code>cal y</code></td>
<td>16yyyy</td>
</tr>
<tr>
<td>load floating accumulator</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Formats**

Floating-point quantities are expressed in the form $y \cdot 2^x$ where the magnitude of $y$ is less than one. Arithmetic is done using a floating-point accumulator (FLAC) which consists of four storage registers. The absolute value of $y$ is stored to double-precision accuracy in the first two registers, the sign of $y$ in the third, and $x + 11$ in the fourth. With a bias of +11, the exponent ranges from -42 to +20. This range was selected by Itek as being most useful for their work.

Operands for floating-point instructions are assumed by the interpreter to be stored in either two or three consecutive storage registers, depending on whether Program Flag 5 is off or on. In the two-register format (Program Flag 5 off), bit 0 (bits being numbered 0 to 17 from left to right) of the first register contains the sign of $y$. As shown in the diagram below, the first 17 bits of the absolute value of $y$ are stored in bits 1-17 of the first register, and the remaining 12 in bits 6-17 of the second register. Bits 0-5 of the second register contain the signed quantity equal to $x$ plus the exponent bias.

\[ \begin{array}{cccc} 
\text{a} & \text{bbbbbbbbbbbbbbbb} & \text{cccccc} & \text{dddddddddddd} \\
\end{array} \]

- **a**: sign of $y$
- **b**: first 17 bits of $y$
- **c**: $x$ plus exponent bias
- **d**: final 12 bits of $y$

**TWO-WORD FORMAT**
In the three-register format (Program Flag 5 on), as illustrated below, bit 0 of the first register contains the sign of $y$ and bits 1-17 are the first 17 bits of the absolute value of $y$. Bit 0 of the second register is always zero and bits 1-17 contain the remaining bits of the absolute value of $y$. The third register contains the value of the exponent incremented by the exponent bias. This three-word format is especially useful for saving and restoring FLAC and is often used only for that purpose.

<table>
<thead>
<tr>
<th></th>
<th>bbbbbbbbbbbbbbbbb</th>
<th>c</th>
<th>ddddddddddddddddd</th>
<th>eeeeeeeeeeeeeeeeeee</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>sign of $y$</td>
<td>b</td>
<td>first 17 bits of $y$</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>final 17 bits of $y$</td>
<td>e</td>
<td>$x$ plus exponent bias</td>
<td></td>
</tr>
</tbody>
</table>

**THREE-WORD FORMAT**

Instructions to be processed interpretively are written in the same format as normal PDP-1 instructions and are assembled with a five-bit operation code, an indirect address bit, and a twelve-bit address. This address refers to two or three consecutive locations, depending on the position of Program Flag 5. Thus, in the description below of the interpreted operations, the symbol $C(Y)$ refers to the contents of locations $Y$, $Y+1$, and optionally $Y+2$, where $Y$ is the address part (after indirect addressing, if any, has been performed) of the instruction being interpreted. If $Y$ is zero, the instruction is interpreted as referring to FLAC itself.

There are eleven floating-point interpretive instructions which, with their overflow and underflow conditions, are described in detail later.

When floating-point operations are to be performed, it is necessary to enter the interpretive portion of FLINT. This is accomplished by the PDP-1 instruction `cal`, which transfers control to location 101, with the location of the next instruction to be interpreted in the accumulator. Since it may often be necessary to enter and leave the interpretive mode, the `cal` instruction is interpreted as a floating load (flg) as well as an entry instruction whenever the address of
the cal is other than zero. Indirect addressing may not be used with the cal instruction since this is assembled as a jda instruction; therefore, if indirect addressing is desired, the correct sequence of instructions would be cal...; flo 'Y'.

The interpreter is so arranged that once the cal instruction is encountered, it will regard each succeeding instruction as a floating-point instruction until it encounters an exit instruction. Any instruction with an operation code number of 10 through 23, 34 through 47, or 60 through 75 will be regarded as an exit instruction with the exception of 16, the cal instruction.

Instructions with these operation code numbers will be simultaneously executed and used as exit instructions when encountered in the interpretive mode. All succeeding instructions will be considered normal machine instructions until another cal is encountered. Thus, such instructions as xor - operation code 06, and - operation code 02, or dio - operation code 32, may not be used in their normal sense while in the interpretive mode. The instructions whose operation codes have thus been preempted by floating instructions were selected because they are unlikely to be used while in floating mode. It is important to note that, once in the interpretive mode, instructions not having the operation codes cited in the preceding paragraph will be interpreted as floating instructions whether or not they are so intended.

Unfloating Routine

The instruction jda unflo enters a subroutine which converts the floating-point number stored in FLAC to a fixed-point integer. This integer is equal to the value of the contents of FLAC divided by the quantity two raised to the power of the contents of location fixexp. The integer resulting from this conversion is stored in the accumulator and the contents of FLAC are destroyed. (The unflo subroutine truncates rather than rounds the quotient obtained by dividing two to the appropriate power into C(FLAC). Thus if FLAC contains 1.48 and fixexp contains 0, jda unflo will put 1 into the accumulator; if FLAC contains 1.48 and fixexp contains 1, jda unflo will put 0 into the accumulator; if FLAC contains 1.48 and fixexp contains -1, jda unflo will put 3 into the accumulator.)
Input Routines

There are three input subroutines which, like the output subroutines, are addressed directly from the main program. The first, entered by the instruction jda readc, reads and translates single characters. The second, entered by the instruction jda readg, handles groups of characters. Each of these two routines reads from punched tape or from the console typewriter, depending on whether the input control word (icword) contains taper (for tape) or typer (for typewriter). FLINT is arranged so that icword contains taper unless this is altered by the user's routine. Such alteration is accomplished by writing: lac taper; dac icword; etc.

After a character is read, it is compared with the entries in a table containing the standard Fio-dec Code for each character as well as a control code that may have one of eight different values. Code 0 marks characters to be ignored, such as illegal configurations which do not correspond to typewriter or Flexowriter symbols. Code 1 marks characters such as space or tab, which serve as delimiters indicating the end of an alphanumeric word. Code 2 marks the decimal digits 0-9 and Code 3 marks the symbols used in floating-point numbers, such as a minus sign or a period (used as a decimal point). Codes 4-7 are assigned to the alphabetic characters; only one bit is tested and all characters having any of these four codes are treated identically.

The readc routine reads a single character, looks it up in the table to find the control code, and returns to the main program with the concise code (with 20 and 0 reversed) in bits 12-17 of the accumulator, which elsewhere is filled with zeros and the iotble entry in 10. If the control code is 0, another character is read and processed in the same manner before returning to the main program.

The readg routine reads numerical or alphabetic groups and determines which group is being read by noting the control code of the first character. If the code is 4 through 7, the group is alphabetic; if 2 or 3, it is numeric; if 0 to 1, the character is ignored and the next character treated as the first.

When reading from paper tape, location buff4 must be set to zero before a call to readg the first time that this instruction is called, and if successive calls to readg are interspersed with calls to any of the other read routines which are also reading from paper tape.
If the group is alphabetic, the characters are translated and their concise codes are saved until either a delimiter (control code 1) is encountered or four characters with control codes 2 through 7 have been read. Characters with control code 0 are always ignored.

The concise codes of the one, two or three characters preceding either the delimiter or the fourth character are then assembled in the accumulator, each occupying six bits with the first one to the left and the whole group right-justified, with zeros on the left if necessary. The control and the concise codes of the delimiter or fourth character are put in 10 bits 0-2 and 12-17, respectively. Program Flag 4 is on if four characters were read, and off if a delimiter was encountered. Control is then returned to the main program.

If the group is numeric, characters are read until a delimiter or a character with control code 4 through 7 is encountered. A plus or minus sign may, but need not, appear anywhere in the number, and there may be a maximum of ten decimal digits. (In FLINT, a plus sign is indicated by ",", a left parenthesis, rather than by "+", the conventional plus symbol. If there are two or more minus signs, all but the last are ignored.)

If a decimal point appears, the resulting number is considered to be a floating-point integer and is formed in FLAC, Program Flag 4 is turned off, and overflow or underflow is signalled as in floating add. If two or more decimal points appear, all but the last are ignored. If no decimal point occurs, the result is considered to be a fixed-point integer. Program Flag 4 is turned on and, if it exceeds 131,071 in magnitude, Program Flag 5 is also turned on. The fixed-point integer appears in the accumulator when control is returned to the main program. Whether the integer is floating-point or fixed-point, the control and the concise codes of the character which served as a delimiter appear in 10 bits 0-2 and 12-17, respectively, and the previous contents of FLAC are destroyed.

The third subroutine, entered by the instruction \texttt{jsp buff}, brings characters from paper tape to the IO register. Before the \texttt{jsp}, the instruction \texttt{dzm buff4} should be given. The first succeeding \texttt{jsp buff} instruction will then read enough characters from paper tape (459, as the buffer length is now set) to fill the buffer and put the Flexowriter code of the first character into 10 bits 10-17. The next \texttt{jsp buff}
instruction places the second character read from the buffer into IO bits 10-17, and each such succeeding instruction brings another character from the buffer into the IO register until all the characters have been brought in. The next jsp buff instruction reads another buffer full of characters from tape, and the entire process is repeated.

Output Routines

There are three output subroutines, all of which write information on punched tape, the console typewriter, or both, depending on whether the output control word, location ocword, contains tapew (tape only), typew (typewriter only), or bothw (tape and typewriter). There is also the write-I0 routine (entered by the instruction jda writio) which writes on paper tape the eight-bit character contained in bits 10-17 as many times as specified by the number in IO bits 0-7. If IO bits 0-7 are zeros, the eight-bit character is written once. No look-up or conversion is performed and the character is written on tape regardless of the contents of the output control word.

The write-character routine, (entered by the instruction jda write) writes the six-bit concise code character contained in IO bits 12-17 as many times as specified by the contents of IO bits 0-7, using the same convention as the write-I0 routine.

The write-integer routine (entered by jda write) writes the integer in the accumulator converted to decimal form, followed by the character in IO bits 12-17. The final character may be written repeatedly according to IO bits 0-7 in the same manner as the write-I0 routine. Insofar as the sign and initial spacing or zero suppression is concerned, the format is controlled by the value of the format control word, format.

The write-floating routine (entered by jda writf) writes the contents of FLAC converted to decimal form, followed by the character in IO bits 12-17 exactly as in the write-integer routine. The contents of FLAC are destroyed after calls to either the write or the writf routine.
Format control is specified by the contents of location format as follows:

Bits 0-5 - The number of digits to the left of the decimal point. If zero or less than the number of significant digits, all significant digits will be printed; otherwise spaces or zeros will appear on the left to fill out the required number of spaces to right-justify the column; this must be \( 12_a \) or less for fixed-point numbers.

Bits 6-11 - The number of digits to the right of the decimal point. This must be zero for fixed-point integers, if zero for floating-point numbers, no decimal point will be printed.

Bits 12-14 - Sign control. If zero, no sign will be printed; if 1, 2 or 3, a minus sign will be printed for negative numbers and nothing, space or plus sign, respectively, for positive numbers.

Bits 15-17 - Zero control. If zero, spaces are used in place of initial zeros; if one, initial zeros are printed, this being useful for handling long integers and fixed-point numbers other than integers.

The contents of format may be altered by the following sequence of instructions: \texttt{lac nf; dac format; etc.}, where \texttt{nf} contains the desired contents of format.

Listed below are system symbols declared by FLINT; therefore, they should not be used by a program which uses FLINT and is assembled with it:

\begin{verbatim}
ioctl
fixexp
unflo
writf
write
writio
bothw
tapew
typew
ocword
write
readg
buff
typer
taper
icword
readc
buff4
format
\end{verbatim}
Description of Instructions

flo - floating load: Unpack C(Y) from its two- or three-word format into the four-word format and place in FLAC.

fad - floating add: Place the arithmetic sum of C(Y) and C(FLAC) in FLAC. If the sum is greater than $2^{131061}$, the result is incorrect and Program Flag 6 is turned on. If the result is less than $2^{-131084}$, or if the mantissa of the sum is zero, the mantissa of FLAC will be positive zero and the exponent of FLAC will be -42 upon completion of the operation. Such astronomical exponents can be obtained only because an entire 18-bit word is allocated to the exponent in FLAC.

fsu - floating subtract: C(Y) is subtracted from C(FLAC) and the difference is put in FLAC. Overflow and underflow are handled as in floating add.

fmu - floating multiply: The product of C(Y) and C(FLAC) is placed in FLAC. Overflow and underflow are handled as in floating add.

fdi - floating divide: C(FLAC) is divided by C(Y) and the quotient is put in FLAC. Overflow and underflow are handled as in floating add.

fsr - floating square root: The square root of C(Y) is put in FLAC if C(Y) is positive. Overflow conditions are not possible. If C(Y) is negative, the contents of FLAC are left undisturbed and Program Flag 4 is turned on.

fsi - floating sine: C(Y) is treated as an angle in radians. The sine of this angle is put into FLAC. Error conditions are not possible.

fco - floating cosine: Cos C(Y) replaces C(FLAC) as in floating sine.
**fda** - floating deposit accumulator: C(FLAC) is packed into the two- or three-word format depending on the position of Program Flag 5, and deposited into locations Y, Y+1, and optionally Y+2. With Program Flag 5 off, if the magnitude is as large as $2^{20}$, Program Flag 6 is turned on. If less than $2^{-43}$, the quantity deposited has a mantissa of zero and an exponent of -43. If Program Flag 5 is on (three-word format), no such check is performed.

**fsk** - floating skip: The interpreter clears the IO register and sets the sign of the accumulator to the sign of C(FLAC), then loads the most significant bits of the mantissa in bits 1-17. It then skips or executes the next sequential instruction, depending on whether the condition tested for is true or false.

**fopr** - floating operate: This instruction places the sign of FLAC in the accumulator, executes the instruction specified by the address part of the fopr (e.g., fopr 200 - clear accumulator and therefore sign register) and returns the result to FLAC.

It is possible that the fopr specified may not change the accumulator (e.g., fopr 15 - set Program Flag 5). In this case the operation will leave the sign of FLAC unchanged.

In preparing a DECAL symbolic tape which will make use of the floating skip and floating operate instructions, the required format is fsk or fopr followed first by the indirect bit if required, and then by the address of the appropriate skip or operate instruction. Thus a floating skip on non-zero accumulator would be written as fsk 100 and a floating complement accumulator as fopr 1000.
Possible Modifications by Users

Partially relocateable version:

All but the first 100₃ instructions for FLINT may be relocated. To do so, the following changes should be made in the symbolic tape:

1. The instruction immediately before the comment "divide here" (on page 15) should be followed by "blk" and "fin"; this is the end of the fixed part.

2. The instruction immediately after the comment "divide here" should be preceded by "blk"; this is the beginning of the relocateable part.

3. The following should be declared as system symbols at the beginning of the fixed part:

\[
\begin{align*}
\text{norm}^4 & \quad \text{fadr} \\
\text{flor} & \quad \text{fsur} \\
a5 & \quad \text{fsrr} \\
a3 & \quad \text{fsir} \\
a4 & \quad \text{fcor} \\
5y & \quad \text{fskr} \\
brkpt & \quad \text{fmur} \\
fdar & \quad \text{fdir} \\
foprr
\end{align*}
\]

These symbols must be located in the relocateable part and their delimiters changed to " ; " (apostrophe).

4. The following should be declared as system symbols at the beginning of the relocateable part:

\[
\begin{align*}
q & \\
a2a & \\
a1 & \\
\text{pc}
\end{align*}
\]

These symbols must be located in the fixed part and their delimiters changed to " ' " (apostrophe).

5. The two parts should be assembled and two loader tapes obtained. The fixed part must be loaded into locations starting at 100₃. The relocateable part may be loaded into any 205₁₆ consecutive locations.
Expansion of input buffer:

The size of the "read group" buffer area may be altered by changing; first, the number currently set at buff42 to the desired value; secondly, the number currently set at buffl+1 to the new value in buff42-1; and, thirdly, the number currently set at buff2a+4 to the new value in buff42.

DECAL Listing

A printout of the symbolic tape of FLINT 36 A3D appears on the next 26 pages.
... FLINT-36 A3D Decal version released October 29, 1963

fopr ewd 760000
fdi ewd 560000
fsk ewd 320000
fmu ewd 540000
fad ewd 020000
fda ewd 000000
fsu ewd 040000
fsr ewd 240000
flo ewd 060000
fsl ewd 260000
fco ewd 300000
z ewd 400000
m ewd 300000
l ewd 200000
s ewd 000000
blk

enter: .
...ac on entry

sub = oct 1
dap pc
law 7777
and'pc
sza'
jmp norm4
dap q
jmp flor

pc:.
lac ..
...program counter
dap q
sma spa szo'
llo = oct 4403
rcl 5
dio +1

a2: ..
...becomes lio reference
spi
jmp a5
...'pc, leave interpretive
... mode

a1:.
spa
jmp a3
...indirectly addressed
ril 1
spi'
jmp q

a2a:.
lac'a2
...flo,fda,fsk
dap +1
jmp..
q:.

law ..
sza!
jmp a4
lac'q
dac sy
jmp brkpt

...program counter

...move flac to y

...address present, unpack

...sign

...to relocatable portion

table:
l fdar
s fadr
s fsur
l flor
z
z
z
z
z
z
z
z
s fsrr
s fsir
s fcor
m fskr
z
z
z
z
z
z
z
s fmur
s fdir
z
z
z
z
z
z
m foprr

...m l fskr in previous

... versions

...m l foprr in previous

... versions
... divide here

fopr  ewd 760000
fdi   ewd 560000
fsk   ewd 320000
fmu   ewd 540000
fad   ewd 020000
fda   ewd 000000
fsu   ewd 040000
fsr   ewd 240000
flo   ewd 060000
fsi   ewd 260000
fco   ewd 300000
z     ewd 400000
m     ewd 300000
l     ewd 200000
s     ewd 000000
blk
brkpt: and = oct 377777
         ...bits 1-17
         dac y
         idx q
         lac'q
         szf 5
         jmp a99
         and = oct 7777
         rai 5
         dac yp
         lac'q
         sar 6
         sar 6
         dac ey
         jmp a2a

a3:.
    lac'q
    dap.q
    rai 5
    jmp a1

a4:.
    lac a
    dac y
    lac ap
    dac yp
    lac sa
    dac sy
    lac ea
    dac ey
    jmp a2a

...pick indirect address

...move flac to y
a5:.
    ril 1
    spi'
    jmp'pc
    jmp a2a

flor:.
    lac'q
    dac sa
    and = oct 377777
    dac a
    idx q
    lac'q
    szf 5
    jmp a98
    and = oct 7777
    ral 5
    dac ap
    lac'q
    sar 6
    sar 6
    dac ea
    jmp norm4

fdar:.
    szf 5
    jmp ++7
    lac ea
    spa
    cma
    scr 5
    sza
    jmp fdar1
    lac sa
    and = oct 400000
    ior a
    dac'q
    idx q
    lac ap
    szf 5
    jmp a97
    add = oct 20
    dac ap
    szo'
    jmp ++14
    dzm ap
    idx a
    sma
    jmp ++4

...execute floating skip
rar 1
dac a
idx ea
law 1
add q
dac q
jmp fdar
ral 1
lio ea
rcr 6
dac'aq
jmp norm4

fdar1:
lac ea
sma
jmp fdar2
lac = oct 0
dac'aq
lio = oct 400000
idx q
dia'aq
jmp norm4

fdar2:
str 6
jmp norm4

fmur:
lac ea
sub factor
add ey
dac ea
szo
jmp fdir5
lac a
mul y
dac temp1
rir 1
dio temp
lac a
mul yp
add temp
and = oct 377777
dac temp
lac temp1
dac a
szo
idx a
lac y
mul ap
add temp
and = oct 377777
dac ap
szo
idx a
lac sa
xor sy
jmp fadr5y

fdir:
cli
lac = oct 200000
div y
jmp fdir3

fdir1:
dac y
dio temp
lac yp
mul y
cma
add temp
mul y

fdir2:
dac temp
spa
jmp fdir4
add temp
and = oct 377777
dac yp
szo
idx y
law 1
add ea
add factor
sub ey
jmp fmur+3

fdir3:
lac y
sas = oct 200000
jmp fadr3y

fdir6:
lac = oct 377776
lio = oct 377776
jmp fdir1

fdir4:
law'1
add y
dac y
lac temp
add = oct 200000
jmp fdir2
fdir5: smo
    jmp →+7
dzm a
dzm ap
dzm sa
law'37
dac ea
    jmp norm4
    stf 6
    jmp fmrur+6
fskr:. lac'pc
    and = oct 17777
    ior = oct 640000
dac fskr1
    lac sa
    and = oct 400000
    ior a
    cli
fskr1: loc
    jmp norm4 ...
done
idx pc
    jmp norm4 ...
done
dsur:. lac sy
cma
dac sy
fadr:. lac ea
    sub ey
    sza!
    jmp fadr2 ...
exponents equal
spa
    jmp fadr7 ...
ea shift
sub = oct 11
dac temp
    sma ...
ey shift
cla
    add shtble ...
table start loc
dap →+4
lac y
lio yp
rli i
xct ..
dac y
cla
rcr 1
dio yp
lac temp
sma sza
    jmp fadr+6
fadr2:     lac sa
         xor sy
         spa
         jmp fadr3     ... signs differ
         lac ap
         add yp
         dac ap
         cla
         szo
         law 1
         add a
         add y
         dac a
         szo'
         jmp norm
         sma
         jmp →+6
         lac y
         sas = oct 377777
         jmp →+3
         law'0
         dac a
         law 1
         add ea
         dac ea
         lac a
         lio ap
         ril 1
         rcr 1
         and = oct 377777
         dac a
         cla
         rcr 1
         dio ap
         szo'
         jmp norm
         spa
         jmp fdir5+2

fadr3y:   stf 6

fadr3:    lac a
         sub y
         dac a
         sza'
jmp fadr4          ...zero result
spa
jmp fadr5          ...minus
lac ap
sub yp
dac ap
sma
jmp norm          ...done
fadr3a:    add = oct 200000
add = oct 200000
dac ap
law'1
add a
dac a
jmp norm          ...done
fadr4:    lac ap
sub yp
dac ap
sma
jmp norm          ...done
fadr5:    cma
dac a
lac sa
cma
dac sa
lac yp
sub ap
jmp fadr3a-3
fadr5y:    dac sa
jmp norm          ...done
fadr5:    cma
dac a
lac sa
cma
dac sa
lac yp
sub ap
jmp fadr3a-3
fadr7:    cma
sub = oct 11
dac temp
sma
cla
add shtble
dap + 4
lac a
lio ap
ril 1
xct ..
dac a
cla
rcr 1
dio ap
lac ey
dac ea
lac temp
sma sza
jmp fadr7+1
jmp fadr2

norm: lac a
sza'
jmp norm2
lio ap
ril 1

norm1: rcl 1
sma'
jmp norm3
dac temp
law'1
add ea
dac ea
lac temp
jmp norm1

norm2: lac ap
sza'
jmp fdir5+2
law'21
add ea
dac ea
lac ap
lio a
jmp norm1-1

norm3: rcr 1
dac a
cla
rcr 1
dio ap

norm4: idx pc
jmp pc

...normalize

...program counter plus one
foprr:  lac sa
        xct'pc
daq sa
        jmp norm4
a97:  dac q
        idx q
        lac ea
        jmp fdar1-2
a98:  dac ap
        idx q
        lac q
        jmp fdar-2
a99:  dac yp
        idx q
        lac q
        jmp a3-2
shtble: loc shtble+11
        scr 1
        scr 2
        scr 3
        scr 4
        scr 5
        scr 6
        scr 7
        scr 8
        scr 9
a:    loc
ap:    loc
sa:    loc
ea:    loc
y:    loc
yp:    loc
sy:    loc
ey:    loc
factor: oct 13
temp:  loc
temp1: loc
ptc:  loc
cc:  loc
format: loc
buff4: loc
        lve oct 46
buff3: loc buff3
        blk
        blk
readc

dap readox

icword

jsp buff

rir 7

spi

jmp icword

rcl 7

and = oct 77

jmp xam

rs5:

oct 764201

szf'1

jmp ->-1

clf 1

tyi

rcl 9

rcl 9

xam:

sza'

jmp ->+4

sad = oct 20

cla

jmp ->+2

claw 20

dac readc

add rs3

dap ->+1

lio ..

cla

rcl 3

sza'

jmp icword

rcr 3

lac readc

readox:

jmp ..

rs3:

and iotble

taper

jsp buff

typer

jmp rs5

buff'

dap buff1

lac buff4

add buff3

dap ->+1

lio ..

isp buff4

...gets jda' to

...to get back

...to get tape character

...tape channel 7 punched?

...yes-get new character

...no-get character into AC

...get concise code in AC

...to exchange 0 and 20

...to accept typewriter

...character

...wait till key hit

...character into AC

...zero ?

...zero

...no-twenty then?x

...then replace with zero

...then okay as is-leave

...table constant to get

... iotble entry

...iotble entry into IO

...control code into AC

...control code zero ?

...yes-get new character

...iotble entry back into IO

...concise code into AC

...exit

...table constant

...paper tape

...typewriter

...pick character

...any left in buffer
buff1: jmp .. ...
    law^45 ...
    dac buff^4 ...
    law buff^4+1 ...
    dap buff2a ...

buff2: rpa'
buff2a: dio .. ...
    idx ← 1 ...
    isp buff^4 ...
    jmp buff2 ...
    law^46 ...
    dac buff^4 ...
    jmp buff^4+1 ...

savr:
dap axt ...
    lac pc ...
    dap rest ...
    lac q ...
    sad = oct 700000 ...
    jmp ←+4 ...
    sub = oct 1 ...
    szf 5 ...
    sub = oct 1 ...
    dap ←+1 ...
    cal .. ...
    law norm^4 ...
    jmp savec ...

save:
loc ...
    dap axt ...
    lac save ...

savec:
dap fx ...
    law 5 ...
    szf 5 ...
    law 15 ...
    dap fx ...
    stf 5 ...

axt:
jmp .. ...

rest:
    law .. ...
    dap pc ...

fx:
oc t 760000 ...

fx:
jmp .. .
readgt loc
jda save
dzm writc
lac rg10c
dac rg7a
stf 4
dzm ptc
...point counter
dzm cc
...char. counter
dzm a
...clear flag
dzm ap
...set exponent
dzm sa
law 55
dac ea
rg1:
jda readc
spi'
jmp rg5
rg2:
dio temp
dac readg
spi
jmp rg2a
rll 1
spi'
jmp rg3
...cc is 4-7
rg2a:
rcr 6
lac cc
...put away character
rcr 6
dac cc
idx ptc
sad = ...
4
jmp rg3a
jda readc
jmp rg2a
rg3:
clf 4
...set I/O exit word
rg3a:
lac = oct 700000
and temp
ior readg
rcr 9
rcr 9
lac cc
jmp fxf
rg5:
rll 1
...none alpha
spi'
...code is 2-3
jmp rg1
...code is one
rg6:  ril 1
    spi
    jmp rg14
    ...code is 3
rg7:  dac readg
    sza
    jmp rg15
    ...code is 2
    idx write
    lac ap
    mul = oct 12
    dac temp
    rir 1
    rcr 9
    rcr 9
    add readg
    dac ap
    lac a
    mul = oct 12
    rir 1
    rcl 9
    rcl 9
    add temp
    dac a
    idx ptc
    idx cc
    lio rg15c
    sad = oct 12
    dio rg7a
    ...10 significant characters
rg8:  jda readc
    spi
    jmp rg9
    ...alpha
    ril 1
    spi
    jmp rg6
    rir 1
rg9:  dac write
    dio writc
    szf'4
    jmp rg11
    ...save AC, IO
rg10:  lac a
    sza
    stf 6
    ...fixed pt. int.
rg10c:  lac ap
        lio sa
        spi
        cma
        lio write
        jmp fxf

rg11:  law +2
        dap pc
        jmp norm
        lac ptc
        sza'
        jmp rg12
        cal ..
        fm u tenth
        law'1
        add ptc
        dac ptc
        jmp rg11+3

rg12:  lac write

rg14:  sad plus
        jmp +10
        sad minus
        jmp +5
        clf 4
        dzm ptc
        idx write
        jmp rg8
        law'0
        dac sa
        jmp rg8

rg15:  lac write
        sza
        jmp rg7a

rg15c:  jmp rg8
        loc
        dap w3
        cla
        rcl 8
        cma
        dac temp
        cla
        rcl 5
        rcl 5
        dac write

...check assembly
sza 1
Jmp w2
sad = oct 20
c1a

w4:
dac temp1
lio temp1

ocword:
jmp tapewa

w1:
isp temp
Jmp w4+1

w3:
jmp ..
w2:
lac = oct 20
Jmp w4
typew:
tyo

tapew:
jmp tapewa
tapewa:
lac write
add rs3
dap →+1
lio .
ppa' jmp w1

bothw:
jmp typew

writio:
loc
dap →+11
c1a
rcl 8

cma
dac temp
rcr 8
ppa' isp temp
jmp →-2
jmp ..

write:
loc
..write integer
jda save
lac write
dzm sa
dzm ap
sma
jmp wr2
dac sa
cma

wr2:
dac a
law 34
dac ea
dio wrt37
law \rightarrow 2
dap pc
jmp norm
lac wrt34
jmp writfd

writf':
loc
jda save
lac wrt35
dio wrt37

writfd:
dac wrt6z
lio sa
dzm sa
dio unflo
oct 760204
lio format
rcl 6
dac readc
sza'
jmp wrt2
cma
dac write

wrt1:
cal ..
fmu tenth
isp write
jmp wrt1

wrt2:
lac ea
sub factor
sma
jmp wrt20
lio format
rcl 6
cla
rcl 6
add readc.
sub = oct 12
sma sza
jmp wrt6x
add = oct 12
mul = oct 452525
scl 2
add factor
dac sixtb
cal ..
fad sixt

...store n positive

...no character to left

...store n negative

...x 1-10

...make flac less than 1

...check assembly
law 20
dac sixthb
lac ea
sub factor
spa
jmp wrt6

cal ..
fmu tenth
idx reade

wrt6x:  lac reade
sza'

wrt6z: jmp wrt5

wrt6: law ->+2
dap pc
jmp norm
fmu ten
lac factor
sub ea
sma sza
jmp wrt3ab

cal ..
fad sixt
lac = oct 170000
and a
ral 6
dac writio
lac a
and = oct 7777
dac a
lac writio
sza
jmp wrt4

wrt3ab: cla
szf 4
jmp wrt3
lio format
rcl 6
sub reade
spa
jmp wrt3c
rcr 6
rir 1
spi!
law 20
wrt3: rcl 9
      rcl 9
      jda wrtc
wrt3c: lac readc
      sub = oct 1
      dac readc
      jmp wrt6-2
wrt5: stf 4
      lio format
      ril 6
      rcl 6
      sza'
      jmp wrt30
      dac readc
      lio point
      jda write
      lac wrt34
      dac wrt6-1
      jmp wrt6
wrt34: jmp wrt30
wrt35: jmp wrt5
wrt31: loc →+1
      lio minus
      jmp wrt36
      lio = oct 20
      lio plus
wrt30: law 70
      and format
      sza'
      jmp wrt36
      rar 3
      lio unflo
      spi
      law ..
      add wrt31
      dap →+1
      xct ..
      jda wrtc
wrt36: clf 4
      lio wrt37
      jda wrtc
      jmp fxf
wrt37: loc

...print point
plus: \text{ oct 57} \\
minus: \text{ oct 54} \\
point: \text{ oct 73} \\
tenth: \text{ oct 31\textfraction{1}631} \\
\quad \text{ oct 231\textfraction{1}631} \\
\quad \text{ oct 10} \\
ten: \text{ oct 240000} \\
loc \quad \text{ oct 17} \\
sixt: \text{ oct 200000} \\
loc \quad \text{ oct 20} \\
sixtb: \text{ oct 20} \\
wrt4: \text{ stf 4} \\
\quad \text{ jmp wrt3} \\
wrt20: \text{ idx readc} \\
\quad \text{ jmp wrt1} \\
unflo1 \text{ loc} \\
dap un5 \\
law 34 \\
\quad \text{ sub ea} \\
\quad \text{ add fixexp} \\
dis \text{ sza1} \\
\quad \text{ jmp un4} \\
\quad \text{ lio right} \\
\quad \text{ spa} \\
\quad \text{ lio left} \\
\quad \text{ dio un3} \\
\quad \text{ sma} \\
\quad \text{ cma} \\
dac unflo \\
un2: \text{ lac a} \\
\quad \text{ lio ap} \\
\quad \text{ ril 1} \\
un3: \text{ loc} \\
dac a \\
\quad \text{ cla} \\
\quad \text{ rcr 1} \\
dac ap \\
\quad \text{ isp unflo} \\
\quad \text{ jmp un2} \\
un4: \text{ lio sa} \\
\quad \text{ lac a} \\
\quad \text{ spi} \\
\quad \text{ cma} \\
\quad \text{ jmp ..} \\
\quad \text{ ..ok as is}
Table

<table>
<thead>
<tr>
<th>Date</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 200001</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200002</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200203</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200004</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200205</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200206</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200007</td>
<td>1</td>
</tr>
<tr>
<td>Oct 20010</td>
<td>1</td>
</tr>
<tr>
<td>Oct 200211</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000100</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000101</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000102</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000103</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000104</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000105</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000106</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000107</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000108</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000109</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000110</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000111</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000112</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000113</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000114</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000115</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000116</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000117</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000118</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000119</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000120</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000121</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000122</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000123</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000124</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000125</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000126</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000127</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000128</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000129</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000130</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000131</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000132</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000133</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000134</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000135</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000136</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000137</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000138</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000139</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000140</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000141</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000142</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000143</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000144</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000145</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000146</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000147</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000148</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000149</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000150</td>
<td>1</td>
</tr>
<tr>
<td>Oct 000151</td>
<td>1</td>
</tr>
</tbody>
</table>

**...zero, not space**

**...space, not zero**
oct 000100
oct 000100
oct 300054
oct 000255
oct 000256
oct 300057
oct 000100
oct 400061
oct 400062
oct 400263
oct 400064
oct 400265
oct 400266
oct 400067
oct 400070
oct 400271
oct 700272
oct 300073
oct 700274
oct 700075
oct 000100
oct 100277

fsrr1:

dac ey
lac y
sar 1
add = oct 200000
jmp fsrr3

fsrr1:

dac ey
lac y
sar 1
add = oct 200000
jmp fsrr3

lac sy
spa
jmp fserr
lac y
sza!
jmp norm4
law'5
dac fscon
jsp savsr
lac ey
sub factor
scr 1
spa
jmp fsme
spl
jmp fsodd
add factor

...square root routine
...sign mantissa
...test for minus
...yes exit

...initialize x sub 1 counter
...exponent
...remove bias
...square root of exponent
...test for add positive exp.
...yes
...test for odd pos. exp.
...yes

...store new exponent
...compute initial x sub 1
...y over 2
fsrr2:  
   lac y 
   sar 1 
   div fxsi  
          ... y over x sub i 
   nop 
   add fxsih 

fsrr3:  
   dac fxsi  
          ... yields new x sub i 
   sar 1  
   dac fxsih  
   isp fscon  
   jmp fsrr2  
   lac ey  
   dac fxsih  
   cal ..  
   fda num  
   flo zero  
   fad fxsi  
   fda fxsi  
   flo num  
   fdi fxsi  
   fad fxsi  
   law'1  
   add ea  
   dac ea  
   jmp rest  
          ... divide above sum by two 

fsme:  
   spi  
   jmp fsrr1-1  
          ... no 
   jmp fsodda  
          ... yes 
/fsodd:  
   add = oct 1  
          ... add one to exponent 

fsodda:  
   add factor 
   dac ey  
   lac y  
   sar 1  
   dac y  
   jmp fsrr1+2  

fserr:  
   str 6  
   jmp norm'4  
          ... set flag 
   jmp rest  
          ... exit 

fxsi:  
   loc  

fscon:  
   loc 

fxsih:  
   loc 

fcor:  
   jsp savsr  
          ... cosine routine 
   cal ..  
   fad ftpi2  
          ... add pi over 2 to make 
          ... like sin 
   jmp fsira  
          ... exit to sin rout.
fsir.:  jsp savsr  ...sine routine
fsira:  cal ..
fdi ftp12  ...convert radians to x
lac sa  ...sign of x
spa
jmp fsir1
fsir2:  cal ..
fsu ftf0r  ...
lac sa  ...
sma
jmp fsir2
cal ..
fad ftc9  ...
lac sa
spa
jmp fsir3
fsir4:  cal ..
fsu ftc9  ...
fsir7:  cal ..
fda fxs1  ...
fmu ..
fda ftx2  ...
fmu ftc7  ...
fmu ftc9  ...
fad ftc5  ...
fmu ftc2  ...
fmu ftc3  ...
fmu ftc2  ...
fmu ftx2  ...
fmu ftc1  ...
fmu fxsi  ...
fsir8:  jmp rest
fsir1:  cal ..
fad ftf0r  ...
jmp fsir+3
fsir3:  cal ..
fad ftc9  ...
law'13  ...
add ea  ...
sma sza
jmp fsir5  ...
lac sa
oma
dac sa
jmp fsir7
fsir5: cal ..
    rad fttwo
    jmp fsir7
ftx2: loc
    loc
    loc
ftone: oct 200000
    loc
    oct 14
fttwo: oct 200000
    loc
    oct 15
ftfor: oct 200000
    loc
    oct 16
ftpl2: oct 311037
    oct 265211
    oct 14
ftc1: oct 311037
    oct 265101
    oct 14
ftc3: oct 645273
    oct 301325
    oct 13
ftc5: oct 243150
    oct 257313
    oct 13
ftc7: oct 631114
    oct 306213
    oct 10
ftc9: oct 236657
    oct 164425
    oct 777776
num: loc
    loc
    loc
zero: loc
    loc
    loc
    blk
fin.