Floating Point Arithmetic Package

The Floating Package is a group of arithmetic subroutines in which numbers are represented in the form $f \times 2^e$. f is a one's complement 18 bit fraction with the binary point between bits 0 and 1. e is a one's complement 18 bit integer exponent of 2. The largest magnitude numbers that can be represented are $\simeq 10^{39,000}$.

A number is normalized when $\frac{1}{2} \leq |f| < 1$. All the floating routines, except the two floating unnormalized adds, return a normalized answer. The fraction appears in the ac, the exponent in the io. Description of routines:

Floating Add - jda fad

One argument should appear in the ac-io. The other argument should have the addresses, direct or indirect, of the fraction and exponent in the two registers following the jda fad.

```
lac f1          /Load first argument.
lio e1
jda fad         /Call floating add.
  f2            /address of second fraction
  e2            /address of exponent for second fraction
dac             /Control returns to here with
                /normalized answer in ac-io.
```

Floating Multiply - jda fmp

```
lac f1          /Load multiplicand.
lio e1
jda fmp         /Call floating multiply
  f2            /address of fraction of multiplier
  e2            /address of exponent of multiplier
dac             /Control returns to here with normalized
                /answer in ac-io.
```

<u>Floating Divide</u> - jda fdv

```
        lac f1          /Load first argument.
        lio e1
        jda fdv         /Call divide.
          f2            /addresses for divisor, hlt will occur
          e2                 /if f2=0.
        dac             /Control returns here with normalized
                        /answer in ac-io.
```

<u>Floating Square Root</u> - jda fsq

Execution time $\simeq$ 385 μ sec.

```
        lac f           /Load argument; argument must be normalized
        lio e
        jda fsq         /Call square root; hlt will occur unless
                        /f≥0.
        dac             /Control returns here with normalized
                        /answer in ac-io.
```

<u>Floating Log, base 2</u> - jda log

```
        lac f           /Load argument.
        lio e
        jda log         /Call log; hlt will occur unless f > 0.
        dac             /Control returns here with normalized
                        /answer in ac-io.
```

<u>Floating Reciprocal</u> - jda rcp

```
        lac f           /Load argument.
        lio e
        jda rcp         /Call reciprocal; hlt will occur in fdh
                        /if f=0.
        dac             /Control returns here with normalized
                        /answer in ac-io.
```

Floating Input - jda fip

```
        jda fip         /Call input; ac-io don't matter.
        jsp             /This instruction is repeatedly
                        /executed (xct) in order to get the input
                        /characters.  The jsp (or jda) could call
                        /a typewriter or reader listen loop
                        /subroutine which should return the
                        /input characters in the
                        /low bits of the io.
        dac             /Control returns here with the answer
                        /in the ac-io after the first illegal
                        /character.
```

Legal characters for fip

x resets routine and starts forming a new number.
Spaces and code deleted characters are ignored.
Legal characters are: ., e, 0-9, -, x, and space.
The illegal character that terminated the number is in register fip.

Input examples:

```
6.9e1
690 e-1234
-6.9 e 17
```

Floating Output - jda fop

```
        lac f           /Load argument.
        lio e
        jda fop         /Call output.
        tyo             /an executed instruction (xct) for output
        dac             /Control returns here with normalized
                        /floating point input quantity.
```

The routine generates parity for each character, so the executed output instruction could be a ppa or a call to an output subroutine.

The output format is .71000 e2, 5 significant figures.

Floating Unnormalized Add - jda fua

```
          lac f1          /Load first argument.
          lio e1
          jda fua         /Call unnormalized add.
            f2            /addresses of second argument
            e2
```

The subroutine returns with a 35 bit number in the ac-io with the binary point after the bit number equal to the larger exponent of the two arguments. If the addition produces an overflow, the larger exponent is incremented by 1. In any case, the larger exponent, perhaps incremented, appears in fac+1.

Examples for subroutine:

```
          lac (200000     /½
          lio (0
          jda fua
            (0            /zero with exponent to cause the number
            (17.          /to be fixed.
```

At return ac,io equals 0,400000.

```
          lac (0          /0
          lio (16.
          jda fua
            (200000       /½
            (0
```

At return ac,io equals 1,0.

Floating Unnormalized Add and Round - jda fur

```
          lac f1          /Load argument
          lio e1
          jda fur         /Call unnormalized add and round
          f2                /addresses of second argument
          e2
        dac                 /Control returns here with fraction
                            /in ac and exponent in io.
```

This routine is the same as fad except that the answer is not normalized. The larger exponent returns in the io, unless overflow occured. Then the larger exponent +1 returns in the io. Example:

```
        lac (300000         /3
        lio (2
        jda fur             /Call subroutine.
          (0                /zero with exponent to
          (17.              /cause the answer to be fixed.
```

At return, ac,io equals 3,17.

Floating Exponentiation - jda f2x

This subroutine calculates 2^x. Execution time $\simeq$ 1.3 m sec.

```
        lac f               /Load argument.
        lio e
        jda f2x             /Call subroutine.
        dac                 /Control returns to here with normalized
                            /answer in ac-io.
```

Bill Gosper and Tom Eggers