http://www.6502.org/source/floats/wozfp1.txt
31 October 2004

TABLE OF CONTENTS

Floating Point Routines for the 6502 by Roy Rankin and Steve Wozniak

Originally published in the August 1976 issue of Dr. Dobb's Journal, these floating point routines allow 6502 users to perform most of the more popular and desired floating point and transcendental functions, namely: Natural Log, Common Log, Addition, Subtraction, Multiplication, Division, and conversions between floating and fixed point numbers.

Errata for Rankin's 6502 Floating Point Routines by Roy Rankin

In the November/December issue of Dr. Dobb's Journal Roy Rankin published three error corrections to the Floating Point Routines presented above.

Floating Point Implementation in the Apple II by Steve Wozniak

An almost identical set of the above routines appeared in the original manual for the Apple II (the Red Book, January 1978). Documentation for these routines appeared in another book, the Wozpak II, in November 1979.

==

Floating Point Routines for the 6502 by Roy Rankin and Steve Wozniak

Originally published in the August 1976 issue of Dr. Dobb's Journal, these floating point routines allow 6502 users to perform most of the more popular and desired floating point and transcendental functions, namely: Natural Log, Common Log, Addition, Subtraction, Multiplication, Division, and conversions between floating and fixed point numbers.

Dr. Dobb's Journal, August 1976, pages 17-19.

Floating Point Routines for the 6502

by Roy Rankin, Department of Mechanical Engineering,
Stanford University, Stanford, CA 94305
(415) 497-1822

and

Steve Wozniak, Apple Computer Company
770 Welch Road, Suite 154
Palo Alto, CA 94304
(415) 326-4248

Editor's Note: Although these routines are for the 6502, it would appear that one could generate equivalent routines for most of the "traditional" microprocessors, relatively easily, by following the flow of the algorithms given in the excellent comments included in the program listing. This is particularly true of the transcendental functions, which were directly modeled after well-known and proven algorithms, and for which, the comments are relatively machine independent.

These floating point routines allow 6502 users to perform most of the more popular and desired floating point and transcendental functions, namely:

Natural Log - LOG
Common Log - LOG10
Exponential - EXP
Floating Add - FADD
Floating Subtract - FSUB
Floating Multiply - FMUL
Floating Divide - FDIV
Convert Floating to Fixed - FIX
Convert Fixed to Floating - FLOAT

They presume a four-byte floating point operand consisting of a one-byte exponent ranging from -128 to +127 and a 24-bit two's complement mantissa between 1.0 and 2.0.

The floating point routines were done by Steve Wozniak, one of the principals in Apple Computer Company. The transcendental functions were patterned after those offered by Hewlett-Packard for their HP2100 minicomputer (with some modifications), and were done by Roy Rankin, a Ph.D. student at Stanford University.

There are three error traps; two for overflow, and one for prohibited logarithm argument. ERROR (1D06) is the error exit used in the event of a non-positive log argument. OVFLW (1E3B) is the error exit for overflow occuring during calculation of e to some power. OVFL (1FE4) is the error exit for overflow in all of the floating point routines. There is no trap for underflow; in such cases, the result is set to 0.0.

All routines are called and exited in a uniform manner: The arguments(s) are placed in the specified floating point storage locations (for specifics, see the documentation preceeding each routine in the listing), then a JSR is used to enter the desired routine. Upon normal completion, the called routine is exited via a subroutine return instruction (RTS).

Note: The preceeding documentation was written by the Editor, based on phone conversations with Roy and studying the listing. There is a high probability that it is correct. However, since it was not written nor reviewed by the authors of these routines, the preceeding documentation may contain errors in concept or in detail.

-- JCW, Jr.

```
    In the Exponent:
    00 Represents -128
          ...
    7F Represents -1
    80 Represents 0
    81 Represents +1
          ...
    FF Represents +127
```

```
  Exponent    Two's Complement Mantissa
  SEEEEEEE  SM.MMMMMM  MMMMMMMM  MMMMMMMM
     n         n+1       n+2       n+3
```

```
                *           JULY 5, 1976
                *     BASIC FLOATING POINT ROUTINES
                *       FOR 6502 MICROPROCESSOR
                *       BY R. RANKIN AND S. WOZNIAK
                *
                *     CONSISTING OF:
                *        NATURAL LOG
                *        COMMON LOG
                *        EXPONENTIAL (E**X)
                *        FLOAT      FIX
                *        FADD       FSUB
                *        FMUL       FDIV
                *
                *
                *     FLOATING POINT REPRESENTATION (4-BYTES)
                *                    EXPONENT BYTE 1
                *                    MANTISSA BYTES 2-4
                *
                *     MANTISSA:    TWO'S COMPLIMENT REPRESENTATION WITH SIGN IN
                *       MSB OF HIGH-ORDER BYTE.  MANTISSA IS NORMALIZED WITH AN
                *       ASSUMED DECIMAL POINT BETWEEN BITS 5 AND 6 OF THE HIGH-ORDER
                *       BYTE.  THUS THE MANTISSA IS IN THE RANGE 1. TO 2. EXCEPT
                *       WHEN THE NUMBER IS LESS THAN 2**(-128).
                *
                *     EXPONENT:    THE EXPONENT REPRESENTS POWERS OF TWO.  THE
                *       REPRESENTATION IS 2'S COMPLIMENT EXCEPT THAT THE SIGN
                *       BIT (BIT 7) IS COMPLIMENTED.  THIS ALLOWS DIRECT COMPARISON
                *       OF EXPONENTS FOR SIZE SINCE THEY ARE STORED IN INCREASING
                *       NUMERICAL SEQUENCE RANGING FROM $00 (-128) TO $FF (+127)
                *       ($ MEANS NUMBER IS HEXADECIMAL).
                *
                *     REPRESENTATION OF DECIMAL NUMBERS:    THE PRESENT FLOATING
                *       POINT REPRESENTATION ALLOWS DECIMAL NUMBERS IN THE APPROXIMATE
                *       RANGE OF 10**(-38) THROUGH 10**(38) WITH 6 TO 7 SIGNIFICANT
                *       DIGITS.
                *
                *
0003                   ORG 3       SET BASE PAGE ADRESSES
0003  EA        SIGN   NOP
0004  EA        X2     NOP         EXPONENT 2
0005  00 00 00  M2     BSS 3       MANTISSA 2
0008  EA        X1     NOP         EXPONENT 1
0009  00 00 00  M1     BSS 3       MANTISSA 1
000C            E      BSS 4       SCRATCH
0010            Z      BSS 4
0014            T      BSS 4
0018            SEXP   BSS 4
001C  00        INT    BSS 1
                *
1D00                   ORG $1D00   STARTING LOCATION FOR LOG
                *
                *
                *     NATURAL LOG OF MANT/EXP1 WITH RESULT IN MANT/EXP1
                *
1D00  A5 09     LOG    LDA M1
1D02  F0 02            BEQ ERROR
```

```
1D04  10 01            BPL CONT    IF ARG>0 OK
1D06  00        ERROR  BRK         ERROR ARG<=0
                *
1D07  20 1C 1F  CONT   JSR SWAP    MOVE ARG TO EXP/MANT2
1D0A  A5 04            LDA X2      HOLD EXPONENT
1D0C  A0 80            LDY =$80
1D0E  84 04            STY X2      SET EXPONENT 2 TO 0 ($80)
1D10  49 80            EOR =$80    COMPLIMENT SIGN BIT OF ORIGINAL EXPONENT
1D12  85 0A            STA M1+1    SET EXPONENT INTO MANTISSA 1 FOR FLOAT
1D14  A9 00            LDA =0
1D16  85 09            STA M1      CLEAR MSB OF MANTISSA 1
1D18  20 2C 1F         JSR FLOAT   CONVERT TO FLOATING POINT
1D1B  A2 03            LDX =3      4 BYTE TRANSFERS
1D1D  B5 04     SEXP1  LDA X2,X
1D1F  95 10            STA Z,X     COPY MANTISSA TO Z
1D21  B5 08            LDA X1,X
1D23  95 18            STA SEXP,X  SAVE EXPONENT IN SEXP
1D25  BD D1 1D         LDA R22,X   LOAD EXP/MANT1 WITH SQRT(2)
1D28  95 08            STA X1,X
1D2A  CA               DEX
1D2B  10 F0            BPL SEXP1
1D2D  20 4A 1F         JSR FSUB    Z-SQRT(2)
1D30  A2 03            LDX =3      4 BYTE TRANSFER
1D32  B5 08     SAVET  LDA X1,X    SAVE EXP/MANT1 AS T
1D34  95 14            STA T,X
1D36  B5 10            LDA Z,X     LOAD EXP/MANT1 WITH Z
1D38  95 08            STA X1,X
1D3A  BD D1 1D         LDA R22,X   LOAD EXP/MANT2 WITH SQRT(2)
1D3D  95 04            STA X2,X
1D3F  CA               DEX
1D40  10 F0            BPL SAVET
1D42  20 50 1F         JSR FADD    Z+SQRT(2)
1D45  A2 03            LDX =3      4 BYTE TRANSFER
1D47  B5 14     TM2    LDA T,X
1D49  95 04            STA X2,X    LOAD T INTO EXP/MANT2
1D4B  CA               DEX
1D4C  10 F9            BPL TM2
1D4E  20 9D 1F         JSR FDIV    T=(Z-SQRT(2))/(Z+SQRT(2))
1D51  A2 03            LDX =3      4 BYTE TRANSFER
1D53  B5 08     MIT    LDA X1,X
1D55  95 14            STA T,X     COPY EXP/MANT1 TO T AND
1D57  95 04            STA X2,X    LOAD EXP/MANT2 WITH T
1D59  CA               DEX
1D5A  10 F7            BPL MIT
1D5C  20 77 1F         JSR FMUL    T*T
1D5F  20 1C 1F         JSR SWAP    MOVE T*T TO EXP/MANT2
1D62  A2 03            LDX =3      4 BYTE TRANSFER
1D64  BD E1 1D  MIC    LDA C,X
1D67  95 08            STA X1,X    LOAD EXP/MANT1 WITH C
1D69  CA               DEX
1D6A  10 F8            BPL MIC
1D6C  20 4A 1F         JSR FSUB    T*T-C
1D6F  A2 03            LDX =3      4 BYTE TRANSFER
1D71  BD DD 1D  M2MB   LDA MB,X
1D74  95 04            STA X2,X    LOAD EXP/MANT2 WITH MB
1D76  CA               DEX
1D77  10 F8            BPL M2MB
1D79  20 9D 1F         JSR FDIV    MB/(T*T-C)
1D7C  A2 03            LDX =3
1D7E  BD D9 1D  M2A1   LDA A1,X
1D81  95 04            STA X2,X    LOAD EXP/MANT2 WITH A1
1D83  CA               DEX
1D84  10 F8            BPL M2A1
1D86  20 50 1F         JSR FADD    MB/(T*T-C)+A1
```

```
1D89  A2 03            LDX =3      4 BYTE TRANSFER
1D8B  B5 14      M2T   LDA T,X
1D8D  95 04            STA X2,X    LOAD EXP/MANT2 WITH T
1D8F  CA               DEX
1D90  10 F9            BPL M2T
1D92  20 77 1F         JSR FMUL    (MB/(T*T-C)+A1)*T
1D95  A2 03            LDX =3      4 BYTE TRANSFER
1D97  BD E5 1D  M2MHL  LDA MHLF,X
1D9A  95 04            STA X2,X    LOAD EXP/MANT2 WITH MHLF (.5)
1D9C  CA               DEX
1D9D  10 F8            BPL M2MHL
1D9F  20 50 1F         JSR FADD    +.5
1DA2  A2 03            LDX =3      4 BYTE TRANSFER
1DA4  B5 18     LDEXP  LDA SEXP,X
1DA6  95 04            STA X2,X    LOAD EXP/MANT2 WITH ORIGINAL EXPONENT
1DA8  CA               DEX
1DA9  10 F9            BPL LDEXP
1DAB  20 50 1F         JSR FADD    +EXPN
1DAE  A2 03            LDX =3      4 BYTE TRANSFER
1DB0  BD D5 1D  MLE2   LDA LE2,X
1DB3  95 04            STA X2,X    LOAD EXP/MANT2 WITH LN(2)
1DB5  CA               DEX
1DB6  10 F8            BPL MLE2
1DB8  20 77 1F         JSR FMUL    *LN(2)
1DBB  60               RTS         RETURN RESULT IN MANT/EXP1
                *
                *     COMMON LOG OF MANT/EXP1 RESULT IN MANT/EXP1
                *
1DBC  20 00 1D  LOG10  JSR LOG     COMPUTE NATURAL LOG
1DBF  A2 03            LDX =3
1DC1  BD CD 1D  L10    LDA LN10,X
1DC4  95 04            STA X2,X    LOAD EXP/MANT2 WITH 1/LN(10)
1DC6  CA               DEX
1DC7  10 F8            BPL L10
1DC9  20 77 1F         JSR FMUL    LOG10(X)=LN(X)/LN(10)
1DCC  60               RTS
                *
1DCD  7E 6F     LN10   DCM 0.4342945
      2D ED
1DD1  80 5A     R22    DCM 1.4142136   SQRT(2)
      02 7A
1DD5  7F 58     LE2    DCM 0.69314718  LOG BASE E OF 2
      B9 0C
1DD9  80 52     A1     DCM 1.2920074
      80 40
1DDD  81 AB     MB     DCM -2.6398577
      86 49
1DE1  80 6A     C      DCM 1.6567626
      08 66
1DE5  7F 40     MHLF   DCM 0.5
      00 00
                *
1E00                   ORG $1E00   STARTING LOCATION FOR EXP
                *
                *     EXP OF MANT/EXP1 RESULT IN MANT/EXP1
                *
1E00  A2 03     EXP    LDX =3      4 BYTE TRANSFER
1E02  BD D8 1E         LDA L2E,X
1E05  95 04            STA X2,X    LOAD EXP/MANT2 WITH LOG BASE 2 OF E
1E07  CA               DEX
1E08  10 F8            BPL EXP+2
1E0A  20 77 1F         JSR FMUL    LOG2(3)*X
1E0D  A2 03            LDX =3      4 BYTE TRANSFER
1E0F  B5 08     FSA    LDA X1,X
```

```
1E11  95 10            STA Z,X     STORE EXP/MANT1 IN Z
1E13  CA               DEX
1E14  10 F9            BPL FSA     SAVE Z=LN(2)*X
1E16  20 E8 1F         JSR FIX     CONVERT CONTENTS OF EXP/MANT1 TO AN INTEGER
1E19  A5 0A            LDA M1+1
1E1B  85 1C            STA INT     SAVE RESULT AS INT
1E1D  38               SEC         SET CARRY FOR SUBTRACTION
1E1E  E9 7C            SBC =124    INT-124
1E20  A5 09            LDA M1
1E22  E9 00            SBC =0
1E24  10 15            BPL OVFLW   OVERFLOW INT>=124
1E26  18               CLC         CLEAR CARRY FOR ADD
1E27  A5 0A            LDA M1+1
1E29  69 78            ADC =120    ADD 120 TO INT
1E2B  A5 09            LDA M1
1E2D  69 00            ADC =0
1E2F  10 0B            BPL CONTIN  IF RESULT POSITIVE CONTINUE
1E31  A9 00            LDA =0      INT<-120 SET RESULT TO ZERO AND RETURN
1E33  A2 03            LDX =3      4 BYTE MOVE
1E35  95 08     ZERO   STA X1,X    SET EXP/MANT1 TO ZERO
1E37  CA               DEX
1E38  10 FB            BPL ZERO
1E3A  60               RTS         RETURN
                *
1E3B  00        OVFLW  BRK         OVERFLOW
                *
1E3C  20 2C 1F  CONTIN JSR FLOAT   FLOAT INT
1E3F  A2 03            LDX =3
1E41  B5 10     ENTD   LDA Z,X
1E43  95 04            STA X2,X    LOAD EXP/MANT2 WITH Z
1E45  CA               DEX
1E46  10 F9            BPL ENTD
1E48  20 4A 1F         JSR FSUB    Z*Z-FLOAT(INT)
1E4B  A2 03            LDX =3      4 BYTE MOVE
1E4D  B5 08     ZSAV   LDA X1,X
1E4F  95 10            STA Z,X     SAVE EXP/MANT1 IN Z
1E51  95 04            STA X2,X    COPY EXP/MANT1 TO EXP/MANT2
1E53  CA               DEX
1E54  10 F7            BPL ZSAV
1E56  20 77 1F         JSR FMUL    Z*Z
1E59  A2 03            LDX =3      4 BYTE MOVE
1E5B  BD DC 1E  LA2    LDA A2,X
1E5E  95 04            STA X2,X    LOAD EXP/MANT2 WITH A2
1E60  B5 08            LDA X1,X
1E62  95 18            STA SEXP,X  SAVE EXP/MANT1 AS SEXP
1E64  CA               DEX
1E65  10 F4            BPL LA2
1E67  20 50 1F         JSR FADD    Z*Z+A2
1E6A  A2 03            LDX =3      4 BYTE MOVE
1E6C  BD E0 1E  LB2    LDA B2,X
1E6F  95 04            STA X2,X    LOAD EXP/MANT2 WITH B2
1E71  CA               DEX
1E72  10 F8            BPL LB2
1E74  20 9D 1F         JSR FDIV    T=B/(Z*Z+A2)
1E77  A2 03            LDX =3      4 BYTE MOVE
1E79  B5 08     DLOAD  LDA X1,X
1E7B  95 14            STA T,X     SAVE EXP/MANT1 AS T
1E7D  BD E4 1E         LDA C2,X
1E80  95 08            STA X1,X    LOAD EXP/MANT1 WITH C2
1E82  B5 18            LDA SEXP,X
1E84  95 04            STA X2,X    LOAD EXP/MANT2 WITH SEXP
1E86  CA               DEX
1E87  10 F0            BPL DLOAD
1E89  20 77 1F         JSR FMUL    Z*Z*C2
```

```
1E8C  20 1C 1F         JSR SWAP    MOVE EXP/MANT1 TO EXP/MANT2
1E8F  A2 03            LDX =3      4 BYTE TRANSFER
1E91  B5 14     LTMP   LDA T,X
1E93  95 08            STA X1,X    LOAD EXP/MANT1 WITH T
1E95  CA               DEX
1E96  10 F9            BPL LTMP
1E98  20 4A 1F         JSR FSUB    C2*Z*Z-B2/(Z*Z+A2)
1E9B  A2 03            LDX =3      4 BYTE TRANSFER
1E9D  BD E8 1E  LDD    LDA D,X
1EA0  95 04            STA X2,X    LOAD EXP/MANT2 WITH D
1EA2  CA               DEX
1EA3  10 F8            BPL LDD
1EA5  20 50 1F         JSR FADD    D+C2*Z*Z-B2/(Z*Z+A2)
1EA8  20 1C 1F         JSR SWAP    MOVE EXP/MANT1 TO EXP/MANT2
1EAB  A2 03            LDX =3      4 BYTE TRANSFER
1EAD  B5 10     LFA    LDA Z,X
1EAF  95 08            STA X1,X    LOAD EXP/MANT1 WITH Z
1EB1  CA               DEX
1EB2  10 F9            BPL LFA
1EB4  20 4A 1F         JSR FSUB    -Z+D+C2*Z*Z-B2/(Z*Z+A2)
1EB7  A2 03            LDX =3      4 BYTE TRANSFER
1EB9  B5 10     LF3    LDA Z,X
1EBB  95 04            STA X2,X    LOAD EXP/MANT2 WITH Z
1EBD  CA               DEX
1EBE  10 F9            BPL LF3
1EC0  20 9D 1F         JSR FDIV    Z/(**** )
1EC3  A2 03            LDX =3      4 BYTE TRANSFER
1EC5  BD E5 1D  LD12   LDA MHLF,X
1EC8  95 04            STA X2,X    LOAD EXP/MANT2 WITH .5
1ECA  CA               DEX
1ECB  10 F8            BPL LD12
1ECD  20 50 1F         JSR FADD    +Z/(***)+.5
1ED0  38               SEC         ADD INT TO EXPONENT WITH CARRY SET
1ED1  A5 1C            LDA INT     TO MULTIPLY BY
1ED3  65 08            ADC X1      2**(INT+1)
1ED5  85 08            STA X1      RETURN RESULT TO EXPONENT
1ED7  60               RTS         RETURN ANS=(.5+Z/(-Z+D+C2*Z*Z-B2/(Z*Z+A2))*2**(INT+1)
1ED8  80 5C     L2E    DCM 1.4426950409   LOG BASE 2 OF E
      55 1E
1EDC  86 57     A2     DCM 87.417497202
      6A E1
1EE0  89 4D     B2     DCM 617.9722695
      3F 1D
1EE4  7B 46     C2     DCM .03465735903
      FA 70
1EE8  83 4F     D      DCM 9.9545957821
      A3 03
                *
                *
                *      BASIC FLOATING POINT ROUTINES
                *
1F00                   ORG $1F00   START OF BASIC FLOATING POINT ROUTINES
1F00  18        ADD    CLC         CLEAR CARRY
1F01  A2 02            LDX =$02    INDEX FOR 3-BYTE ADD
1F03  B5 09     ADD1   LDA M1,X
1F05  75 05            ADC M2,X    ADD A BYTE OF MANT2 TO MANT1
1F07  95 09            STA M1,X
1F09  CA               DEX         ADVANCE INDEX TO NEXT MORE SIGNIF.BYTE
1F0A  10 F7            BPL ADD1    LOOP UNTIL DONE.
1F0C  60               RTS         RETURN
1F0D  06 03     MD1    ASL SIGN    CLEAR LSB OF SIGN
1F0F  20 12 1F         JSR ABSWAP  ABS VAL OF MANT1, THEN SWAP MANT2
1F12  24 09     ABSWAP BIT M1      MANT1 NEG?
1F14  10 05            BPL ABSWP1  NO,SWAP WITH MANT2 AND RETURN
```

```
1F16  20 8F 1F         JSR FCOMPL  YES, COMPLIMENT IT.
1F19  E6 03            INC SIGN    INCR SIGN, COMPLEMENTING LSB
1F1B  38        ABSWP1 SEC         SET CARRY FOR RETURN TO MUL/DIV
                *
                *     SWAP EXP/MANT1 WITH EXP/MANT2
                *
1F1C  A2 04     SWAP   LDX =$04    INDEX FOR 4-BYTE SWAP.
1F1E  94 0B     SWAP1  STY E-1,X
1F20  B5 07            LDA X1-1,X  SWAP A BYTE OF EXP/MANT1 WITH
1F22  B4 03            LDY X2-1,X  EXP/MANT2 AND LEAVEA COPY OF
1F24  94 07            STY X1-1,X  MANT1 IN E(3BYTES). E+3 USED.
1F26  95 03            STA X2-1,X
1F28  CA               DEX         ADVANCE INDEX TO NEXT BYTE
1F29  D0 F3            BNE SWAP1   LOOP UNTIL DONE.
1F2B  60               RTS
                *
                *
                *
                *     CONVERT 16 BIT INTEGER IN M1(HIGH) AND M1+1(LOW) TO F.P.
                *     RESULT IN EXP/MANT1.  EXP/MANT2 UNEFFECTED
                *
                *
1F2C  A9 8E     FLOAT  LDA =$8E
1F2E  85 08            STA X1      SET EXPN TO 14 DEC
1F30  A9 00            LDA =0      CLEAR LOW ORDER BYTE
1F32  85 0B            STA M1+2
1F34  F0 08            BEQ NORM    NORMALIZE RESULT
1F36  C6 08     NORM1  DEC X1      DECREMENT EXP1
1F38  06 0B            ASL M1+2
1F3A  26 0A            ROL M1+1    SHIFT MANT1 (3 BYTES) LEFT
1F3C  26 09            ROL M1
1F3E  A5 09     NORM   LDA M1      HIGH ORDER MANT1 BYTE
1F40  0A               ASL         UPPER TWO BITS UNEQUAL?
1F41  45 09            EOR M1
1F43  30 04            BMI RTS1    YES,RETURN WITH MANT1 NORMALIZED
1F45  A5 08            LDA X1      EXP1 ZERO?
1F47  D0 ED            BNE NORM1   NO, CONTINUE NORMALIZING
1F49  60        RTS1   RTS         RETURN
                *
                *
                *     EXP/MANT2-EXP/MANT1 RESULT IN EXP/MANT1
                *
1F4A  20 8F 1F  FSUB   JSR FCOMPL  CMPL MANT1 CLEARS CARRY UNLESS ZERO
1F4D  20 5D 1F  SWPALG JSR ALGNSW  RIGHT SHIFT MANT1 OR SWAP WITH MANT2 ON CARRY
                *
                *     ADD EXP/MANT1 AND EXP/MANT2 RESULT IN EXP/MANT1
                *
1F50  A5 04     FADD   LDA X2
1F52  C5 08            CMP X1      COMPARE EXP1 WITH EXP2
1F54  D0 F7            BNE SWPALG  IF UNEQUAL, SWAP ADDENDS OR ALIGN MANTISSAS
1F56  20 00 1F         JSR ADD     ADD ALIGNED MANTISSAS
1F59  50 E3     ADDEND BVC NORM    NO OVERFLOW, NORMALIZE RESULTS
1F5B  70 05            BVS RTLOG   OV: SHIFT MANT1 RIGHT. NOTE CARRY IS CORRECT SIGN
1F5D  90 BD     ALGNSW BCC SWAP    SWAP IF CARRY CLEAR, ELSE SHIFT RIGHT ARITH.
1F5F  A5 09     RTAR   LDA M1      SIGN OF MANT1 INTO CARRY FOR
1F61  0A               ASL         RIGHT ARITH SHIFT
1F62  E6 08     RTLOG  INC X1      INCR EXP1 TO COMPENSATE FOR RT SHIFT
1F64  F0 7E            BEQ OVFL    EXP1 OUT OF RANGE.
1F66  A2 FA     RTLOG1 LDX =$FA    INDEX FOR 6 BYTE RIGHT SHIFT
1F68  A9 80     ROR1   LDA =$80
1F6A  B0 01            BCS ROR2
1F6C  0A               ASL
1F6D  56 0F     ROR2   LSR E+3,X   SIMULATE ROR E+3,X
1F6F  15 0F            ORA E+3,X
```

```
1F71  95 0F            STA E+3,X
1F73  E8               INX         NEXT BYTE OF SHIFT
1F74  D0 F2            BNE ROR1    LOOP UNTIL DONE
1F76  60               RTS         RETURN
                *
                *
                *     EXP/MANT1 X EXP/MANT2 RESULT IN EXP/MANT1
                *
1F77  20 0D 1F  FMUL   JSR MD1     ABS. VAL OF MANT1, MANT2
1F7A  65 08            ADC X1      ADD EXP1 TO EXP2 FOR PRODUCT EXPONENT
1F7C  20 CD 1F         JSR MD2     CHECK PRODUCT EXP AND PREPARE FOR MUL
1F7F  18               CLC         CLEAR CARRY
1F80  20 66 1F  MUL1   JSR RTLOG1  MANT1 AND E RIGHT.(PRODUCT AND MPLIER)
1F83  90 03            BCC MUL2    IF CARRY CLEAR, SKIP PARTIAL PRODUCT
1F85  20 00 1F         JSR ADD     ADD MULTIPLICAN TO PRODUCT
1F88  88        MUL2   DEY         NEXT MUL ITERATION
1F89  10 F5            BPL MUL1    LOOP UNTIL DONE
1F8B  46 03     MDEND  LSR SIGN    TEST SIGN (EVEN/ODD)
1F8D  90 AF     NORMX  BCC NORM    IF EXEN, NORMALIZE PRODUCT, ELSE COMPLEMENT
1F8F  38        FCOMPL SEC         SET CARRY FOR SUBTRACT
1F90  A2 03            LDX =$03    INDEX FOR 3 BYTE SUBTRACTION
1F92  A9 00     COMPL1 LDA =$00    CLEAR A
1F94  F5 08            SBC X1,X    SUBTRACT BYTE OF EXP1
1F96  95 08            STA X1,X    RESTORE IT
1F98  CA               DEX         NEXT MORE SIGNIFICANT BYTE
1F99  D0 F7            BNE COMPL1  LOOP UNTIL DONE
1F9B  F0 BC            BEQ ADDEND  NORMALIZE (OR SHIFT RIGHT IF OVERFLOW)
                *
                *
                *     EXP/MANT2 / EXP/MANT1 RESULT IN EXP/MANT1
                *
1F9D  20 0D 1F  FDIV   JSR MD1     TAKE ABS VAL OF MANT1, MANT2
1FA0  E5 08            SBC X1      SUBTRACT EXP1 FROM EXP2
1FA2  20 CD 1F         JSR MD2     SAVE AS QUOTIENT EXP
1FA5  38        DIV1   SEC         SET CARRY FOR SUBTRACT
1FA6  A2 02            LDX =$02    INDEX FOR 3-BYTE INSTRUCTION
1FA8  B5 05     DIV2   LDA M2,X
1FAA  F5 0C            SBC E,X     SUBTRACT A BYTE OF E FROM MANT2
1FAC  48               PHA         SAVE ON STACK
1FAD  CA               DEX         NEXT MORE SIGNIF BYTE
1FAE  10 F8            BPL DIV2    LOOP UNTIL DONE
1FB0  A2 FD            LDX =$FD    INDEX FOR 3-BYTE CONDITIONAL MOVE
1FB2  68        DIV3   PLA         PULL A BYTE OF DIFFERENCE OFF STACK
1FB3  90 02            BCC DIV4    IF MANT2<E THEN DONT RESTORE MANT2
1FB5  95 08            STA M2+3,X
1FB7  E8        DIV4   INX         NEXT LESS SIGNIF BYTE
1FB8  D0 F8            BNE DIV3    LOOP UNTIL DONE
1FBA  26 0B            ROL M1+2
1FBC  26 0A            ROL M1+1    ROLL QUOTIENT LEFT, CARRY INTO LSB
1FBE  26 09            ROL M1
1FC0  06 07            ASL M2+2
1FC2  26 06            ROL M2+1    SHIFT DIVIDEND LEFT
1FC4  26 05            ROL M2
1FC6  B0 1C            BCS OVFL    OVERFLOW IS DUE TO UNNORMALIZED DIVISOR
1FC8  88               DEY         NEXT DIVIDE ITERATION
1FC9  D0 DA            BNE DIV1    LOOP UNTIL DONE 23 ITERATIONS
1FCB  F0 BE            BEQ MDEND   NORMALIZE QUOTIENT AND CORRECT SIGN
1FCD  86 0B     MD2    STX M1+2
1FCF  86 0A            STX M1+1    CLR MANT1 (3 BYTES) FOR MUL/DIV
1FD1  86 09            STX M1
1FD3  B0 0D            BCS OVCHK   IF EXP CALC SET CARRY, CHECK FOR OVFL
1FD5  30 04            BMI MD3     IF NEG NO UNDERFLOW
1FD7  68               PLA         POP ONE
1FD8  68               PLA         RETURN LEVEL
```

```
1FD9  90 B2            BCC NORMX   CLEAR X1 AND RETURN
1FDB  49 80     MD3    EOR =$80    COMPLIMENT SIGN BIT OF EXP
1FDD  85 08            STA X1      STORE IT
1FDF  A0 17            LDY =$17    COUNT FOR 24 MUL OR 23 DIV ITERATIONS
1FE1  60               RTS         RETURN
1FE2  10 F7     OVCHK  BPL MD3     IF POS EXP THEN NO OVERFLOW
1FE4  00        OVFL   BRK
                *
                *
                *     CONVERT EXP/MANT1 TO INTEGER IN M1 (HIGH) AND M1+1(LOW)
                *      EXP/MANT2 UNEFFECTED
                *
1FE5  20 5F 1F         JSR RTAR    SHIFT MANT1 RT AND INCREMENT EXPNT
1FE8  A5 08     FIX    LDA X1      CHECK EXPONENT
1FEA  C9 8E            CMP =$8E    IS EXPONENT 14?
1FEC  D0 F7            BNE FIX-3   NO, SHIFT
1FEE  60        RTRN   RTS         RETURN
                       END
```

==

Errata for Rankin's 6502 Floating Point Routines by Roy Rankin

In the November/December issue of Dr. Dobb's Journal Roy Rankin published three error corrections to the Floating Point Routines presented above.

Dr. Dobb's Journal, November/December 1976, page 57.

ERRATA FOR RANKIN'S 6502
FLOATING POINT ROUTINES

Sept. 22, 1976

Dear Jim,

Subsequent to the publication of "Floating Point Routines for the 6502" (Vol.1, No.7) an error which I made in the LOG routine came to light which causes improper results if the argument is less than 1. The following changes will correct the error.

```
1.  After:            CONT JSR SWAP (1D07)
    Add:    A2 00          LDX =0    LOAD X FOR HIGH BYTE OF EXPONENT

2.  After:                 STA M1+1 (1D12)
    Delete:                LDA =0
                           STA M1
    Add:    10 01          BPL *+3   IS EXPONENT NEGATIVE
            CA             DEX       YES, SET X TO $FF
            86 09          STX M1    SET UPPER BYTE OF EXPONENT
```

3. Changes 1 and 2 shift the code by 3 bytes so add 3 to the addresses of the constants LN10 through MHLF whenever they are referenced. For example the address of LN10 changes from 1DCD to 1DD0. Note also that the entry point for LOG10 becomes 1DBF. The routines stays within the page and hence the following routines (EXP etc.) are not affected.

Yours truly,

Roy Rankin
Dep. of Mech. Eng.
Stanford University

==

Floating Point Implementation in the Apple II by Steve Wozniak

An almost identical set of the above routines appeared in the original manual for the Apple II (the Red Book, January 1978). Documentation for these routines appeared in another book, the Wozpak II, in November 1979.

Woz 6502 Floating Point Routines

Apple II Reference Manual (Red Book), January 1978, pages 94-95.

```
                ***********************
                *                     *
                *  APPLE-II FLOATING  *
                *   POINT ROUTINES    *
                *                     *
                *  COPYRIGHT 1977 BY  *
                * APPLE COMPUTER INC. *
                *                     *
                * ALL RIGHTS RESERVED *
                *                     *
                *     S. WOZNIAK      *
                *                     *
                ***********************
                 TITLE "FLOATING POINT ROUTINES"
                SIGN      EPZ  $F3
                X2        EPZ  $F4
                M2        EPZ  $F5
                X1        EPZ  $F8
                M1        EPZ  $F9
                E         EPZ  $FC
                OVLOC     EQU  $3F5
                          ORG  $F425
F425: 18        ADD       CLC            CLEAR CARRY
F426: A2 02               LDX  #$2       INDEX FOR 3-BYTE ADD.
F428: B5 F9     ADD1      LDA  M1,X
F42A: 75 F5               ADC  M2,X      ADD A BYTE OF MANT2 TO MANT1
F42C: 95 F9               STA  M1,X
F42E: CA                  DEX            INDEX TO NEXT MORE SIGNIF. BYTE.
F42F: 10 F7               BPL  ADD1      LOOP UNTIL DONE.
F431: 60                  RTS            RETURN
F432: 06 F3     MD1       ASL  SIGN      CLEAR LSB OF SIGN.
F434: 20 37 F4            JSR  ABSWAP    ABS VAL OF M1, THEN SWAP WITH M2
F437: 24 F9     ABSWAP    BIT  M1        MANT1 NEGATIVE?
F439: 10 05               BPL  ABSWAP1   NO, SWAP WITH MANT2 AND RETURN.
F43B: 20 A4 F4            JSR  FCOMPL    YES, COMPLEMENT IT.
F43E: E6 F3               INC  SIGN      INCR SIGN, COMPLEMENTING LSB.
F440: 38        ABSWAP1   SEC            SET CARRY FOR RETURN TO MUL/DIV.
F441: A2 04     SWAP      LDX  #$4       INDEX FOR 4 BYTE SWAP.
F443: 94 FB     SWAP1     STY  E-1,X
F445: B5 F7               LDA  X1-1,X    SWAP A BYTE OF EXP/MANT1 WITH
F447: B4 F3               LDY  X2-1,X    EXP/MANT2 AND LEAVE A COPY OF
F449: 94 F7               STY  X1-1,X    MANT1 IN E (3 BYTES).  E+3 USED
F44B: 95 F3               STA  X2-1,X
F44D: CA                  DEX            ADVANCE INDEX TO NEXT BYTE
```

```
F44E: D0 F3               BNE  SWAP1    LOOP UNTIL DONE.
F450: 60                  RTS           RETURN
F451: A9 8E     FLOAT     LDA  #$8E     INIT EXP1 TO 14,
F453: 85 F8               STA  X1       THEN NORMALIZE TO FLOAT.
F455: A5 F9     NORM1     LDA  M1       HIGH-ORDER MANT1 BYTE.
F457: C9 C0               CMP  #$C0     UPPER TWO BITS UNEQUAL?
F459: 30 0C               BMI  RTS1     YES, RETURN WITH MANT1 NORMALIZED
F45B: C6 F8               DEC  X1       DECREMENT EXP1.
F45D: 06 FB               ASL  M1+2
F45F: 26 FA               ROL  M1+1     SHIFT MANT1 (3 BYTES) LEFT.
F461: 26 F9               ROL  M1
F463: A5 F8     NORM      LDA  X1       EXP1 ZERO?
F465: D0 EE               BNE  NORM1    NO, CONTINUE NORMALIZING.
F467: 60        RTS1      RTS           RETURN.
F468: 20 A4 F4  FSUB      JSR  FCOMPL   CMPL MANT1,CLEARS CARRY UNLESS 0
F46B: 20 7B F4  SWPALGN   JSR  ALGNSWP  RIGHT SHIFT MANT1 OR SWAP WITH
F46E: A5 F4     FADD      LDA  X2
F470: C5 F8               CMP  X1       COMPARE EXP1 WITH EXP2.
F472: D0 F7               BNE  SWPALGN  IF #,SWAP ADDENDS OR ALIGN MANTS.
F474: 20 25 F4            JSR  ADD      ADD ALIGNED MANTISSAS.
F477: 50 EA     ADDEND    BVC  NORM     NO OVERFLOW, NORMALIZE RESULT.
F479: 70 05               BVS  RTLOG    OV: SHIFT M1 RIGHT, CARRY INTO SIGN
F47B: 90 C4     ALGNSWP   BCC  SWAP     SWAP IF CARRY CLEAR,
                *       ELSE SHIFT RIGHT ARITH.
F47D: A5 F9     RTAR      LDA  M1       SIGN OF MANT1 INTO CARRY FOR
F47F: 0A                  ASL           RIGHT ARITH SHIFT.
F480: E6 F8     RTLOG     INC  X1       INCR X1 TO ADJUST FOR RIGHT SHIFT
F482: F0 75               BEQ  OVFL     EXP1 OUT OF RANGE.
F484: A2 FA     RTLOG1    LDX  #$FA     INDEX FOR 6:BYTE RIGHT SHIFT.
F486: 76 FF     ROR1      ROR  E+3,X
F488: E8                  INX           NEXT BYTE OF SHIFT.
F489: D0 FB               BNE  ROR1     LOOP UNTIL DONE.
F48B: 60                  RTS           RETURN.
F48C: 20 32 F4  FMUL      JSR  MD1      ABS VAL OF MANT1, MANT2
F48F: 65 F8               ADC  X1       ADD EXP1 TO EXP2 FOR PRODUCT EXP
F491: 20 E2 F4            JSR  MD2      CHECK PROD. EXP AND PREP. FOR MUL
F494: 18                  CLC           CLEAR CARRY FOR FIRST BIT.
F495: 20 84 F4  MUL1      JSR  RTLOG1   M1 AND E RIGHT (PROD AND MPLIER)
F498: 90 03               BCC  MUL2     IF CARRY CLEAR, SKIP PARTIAL PROD
F49A: 20 25 F4            JSR  ADD      ADD MULTIPLICAND TO PRODUCT.
F49D: 88        MUL2      DEY           NEXT MUL ITERATION.
F49E: 10 F5               BPL  MUL1     LOOP UNTIL DONE.
F4A0: 46 F3     MDEND     LSR  SIGN     TEST SIGN LSB.
F4A2: 90 BF     NORMX     BCC  NORM     IF EVEN,NORMALIZE PROD,ELSE COMP
F4A4: 38        FCOMPL    SEC           SET CARRY FOR SUBTRACT.
F4A5: A2 03               LDX  #$3      INDEX FOR 3 BYTE SUBTRACT.
F4A7: A9 00     COMPL1    LDA  #$0      CLEAR A.
F4A9: F5 F8               SBC  X1,X     SUBTRACT BYTE OF EXP1.
F4AB: 95 F8               STA  X1,X     RESTORE IT.
F4AD: CA                  DEX           NEXT MORE SIGNIFICANT BYTE.
F4AE: D0 F7               BNE  COMPL1   LOOP UNTIL DONE.
F4B0: F0 C5               BEQ  ADDEND   NORMALIZE (OR SHIFT RT IF OVFL).
F4B2: 20 32 F4  FDIV      JSR  MD1      TAKE ABS VAL OF MANT1, MANT2.
F4B5: E5 F8               SBC  X1       SUBTRACT EXP1 FROM EXP2.
F4B7: 20 E2 F4            JSR  MD2      SAVE AS QUOTIENT EXP.
F4BA: 38        DIV1      SEC           SET CARRY FOR SUBTRACT.
F4BB: A2 02               LDX  #$2      INDEX FOR 3-BYTE SUBTRACTION.
F4BD: B5 F5     DIV2      LDA  M2,X
F4BF: F5 FC               SBC  E,X      SUBTRACT A BYTE OF E FROM MANT2.
F4C1: 48                  PHA           SAVE ON STACK.
F4C2: CA                  DEX           NEXT MORE SIGNIFICANT BYTE.
F4C3: 10 F8               BPL  DIV2     LOOP UNTIL DONE.
F4C5: A2 FD               LDX  #$FD     INDEX FOR 3-BYTE CONDITIONAL MOVE
F4C7: 68        DIV3      PLA           PULL BYTE OF DIFFERENCE OFF STACK
```

```
F4C8: 90 02              BCC  DIV4     IF M2<E THEN DON'T RESTORE M2.
F4CA: 95 F8              STA  M2+3,X
F4CC: E8        DIV4     INX           NEXT LESS SIGNIFICANT BYTE.
F4CD: D0 F8              BNE  DIV3     LOOP UNTIL DONE.
F4CF: 26 FB              ROL  M1+2
F4D1: 26 FA              ROL  M1+1     ROLL QUOTIENT LEFT, CARRY INTO LSB
F4D3: 26 F9              ROL  M1
F4D5: 06 F7              ASL  M2+2
F4D7: 26 F6              ROL  M2+1     SHIFT DIVIDEND LEFT
F4D9: 26 F5              ROL  M2
F4DB: B0 1C              BCS  OVFL     OVFL IS DUE TO UNNORMED DIVISOR
F4DD: 88                 DEY           NEXT DIVIDE ITERATION.
F4DE: D0 DA              BNE  DIV1     LOOP UNTIL DONE 23 ITERATIONS.
F4E0: F0 BE              BEQ  MDEND    NORM. QUOTIENT AND CORRECT SIGN.
F4E2: 86 FB     MD2      STX  M1+2
F4E4: 86 FA              STX  M1+1     CLEAR MANT1 (3 BYTES) FOR MUL/DIV.
F4E6: 86 F9              STX  M1
F4E8: B0 0D              BCS  OVCHK    IF CALC. SET CARRY,CHECK FOR OVFL
F4EA: 30 04              BMI  MD3      IF NEG THEN NO UNDERFLOW.
F4EC: 68                 PLA           POP ONE RETURN LEVEL.
F4ED: 68                 PLA
F4EE: 90 B2              BCC  NORMX    CLEAR X1 AND RETURN.
F4F0: 49 80     MD3      EOR  #$80     COMPLEMENT SIGN BIT OF EXPONENT.
F4F2: 85 F8              STA  X1       STORE IT.
F4F4: A0 17              LDY  #$17     COUNT 24 MUL/23 DIV ITERATIONS.
F4F6: 60                 RTS           RETURN.
F4F7: 10 F7     OVCHK    BPL  MD3      IF POSITIVE EXP THEN NO OVFL.
F4F9: 4C F5 03  OVFL     JMP  OVLOC
                         ORG  $F63D
F63D: 20 7D F4  FIX1     JSR  RTAR
F640: A5 F8     FIX      LDA  X1
F642: 10 13              BPL  UNDFL
F644: C9 8E              CMP  #$8E
F646: D0 F5              BNE  FIX1
F648: 24 F9              BIT  M1
F64A: 10 0A              BPL  FIXRTS
F64C: A5 FB              LDA  M1+2
F64E: F0 06              BEQ  FIXRTS
F650: E6 FA              INC  M1+1
F652: D0 02              BNE  FIXRTS
F654: E6 F9              INC  M1
F656: 60        FIXRTS   RTS
F657: A9 00     UNDFL    LDA  #$0
F659: 85 F9              STA  M1
F65B: 85 FA              STA  M1+1
F65D: 60                 RTS
```

****

Wozpak][, November 1979, pages 109-115.

FLOATING POINT PACKAGE

The mantissa-exponent, or 'floating point' numerical representation is widely used by computers to express values with a wide dynamic range. With floating point representation, the number 7.5 x 10^22 requires no more memory to store than the number 75 does. We have allowed for binary floating point arithmetic on the APPLE][computer by providing a useful subroutine package in ROM, which performs the common arithmetic functions. Maximum precision is retained by these routines and overflow conditions such as 'divide by zero' are trapped for the user. The 4-byte floating point number representation is compatible with future APPLE products such as floating point BASIC.

A small amount of memory in Page Zero is dedicated to the floating point workspace, including the two floating-point accumulators, FP1 and FP2. After placing operands in these accumulators, the user calls subroutines in the ROM which perform the desired arithmetic operations, leaving results in FP1. Should an overflow condition occur, a jump to location $3F5 is executed, allowing a user routine to take appropriate action.

FLOATING POINT REPRESENTATION

```
  _____    _____    _____    _____
 |     |  |     |  |     |  |     |
 |     |  | HI  |  |     |  | LOW |
 |_____|  |_____|  |_____|  |_____|

Exponent       Signed Mantissa
```

1. Mantissa

The floating point mantissa is stored in two's complement representation with the sign at the most significant bit (MSB) position of the high-order mantissa byte. The mantissa provides 24 bits of precision, including sign, and can represent 24-bit integers precisely. Extending precision is simply a matter of adding bytes at the low order end of the mantissa.

Except for magnitudes less than 2^-128 (which lose precision) mantissa are normalized by the floating point routines to retain maximum precision. That is, the numbers are adjusted so that the upper two high-order mantissa bits are unequal.

HIGH-ORDER MANTISSA BYTE	
01.XXXXXX	Positive mantissa.
10.XXXXXX	Negative mantissa.
00.XXXXXX	Unnormalized mantissa.
11.XXXXXX	Exponent = -128.

2. Exponent.

The exponent is a binary scaling factor (power of two) which is applied to the mantissa. Ranging from -128 to +127, the exponent is stored in standard two's complement representation except for the sign bit which is complemented. This representation allows direct comparison of exponents, since they are stored in increasing numerical sequence. The most negative exponent, corresponding to the smallest magnItude, -128, is stored as $00 ($ means hexidecimal) and the most positive, +127, is stored as $FF (all ones).

EXPONENT	STORED AS
+127	11111111 ($FF)
+3	10000011 ($83)
+2	10000010 ($82)

+1	10000001	($81)
0	10000000	($80)
-1	01111111	($7F)
-2	01111110	($7E)
-3	01111101	($7D)
-128	00000000	($00)

The smallest magnitude which can be represented is 2^-150.

```
  _____    _____    _____    _____
 |     |  |     |  |     |  |     |
 |  0  |  |  0  |  |  0  |  |  1  |
 |_____|  |_____|  |_____|  |_____|

           HIGH                LOW
   EXP            MANTISSA
```

The largest positive magnitude which can be represented is +2^128-1.

```
  _____    _____    _____    _____
 |     |  |     |  |     |  |     |
 | $7F |  | $7F |  | $FF |  | $FF |
 |_____|  |_____|  |_____|  |_____|

   EXP            MANTISSA
```

FLOATING POINT REPRESENTATION EXAMPLES

DECIMAL NUMBER	HEX EXPONENT	HEX MANTISSA
+ 3	81	60 00 00
+ 4	82	40 00 00
+ 5	82	50 00 00
+ 7	82	70 00 00
+12	83	60 00 00
+15	83	78 00 00
+17	84	44 00 00
+20	84	50 00 00
+60	85	78 00 00
- 3	81	A0 00 00
- 4	81	80 00 00
- 5	82	B0 00 00
- 7	82	90 00 00
-12	83	A0 00 00
-15	83	88 00 00
-17	84	BC 00 00
-20	84	B0 00 00
-60	85	88 00 00

FLOATING POINT SUBROUTINE DESCRIPTIONS

FCOMPL subroutine (address $F4A4)

Purpose: FCOMPL is used to negate floating point numbers.

Entry: A normalized or unnormalized value is in FP1 (floating point accumulator 1).

Uses: NORM, RTLOG.

Exit: The value in FP1 is negated and then normalized to retain precision. The 3-byte FP1 extension, E, may also be altered but FP2 and SIGN are not disturbed. The 6502 A-REG is altered and the X-REG is cleared. The Y-REG is not disturbed.

Caution: Attempting to negate -2^128 will result in an overflow since +2^128 is not representable, and a jump to location $3F5 will be executed, with the following contents in FP1.

```
      _____   _____   _____   _____
     |     | |     | |     | |     |
FP1: |  0  | | $80 | |  0  | |  0  |
     |_____| |_____| |_____| |_____|

       X1      M1
```

Example: Prior to calling FCOMPL, FP1 contains +15.

```
      _____   _____   _____   _____
     |     | |     | |     | |     |
FP1: | $83 | | $78 | |  0  | |  0  |   (+15)
     |_____| |_____| |_____| |_____|

       X1      M1
```

After calling FCOMPL as a subroutine, FP1 contains -15.

```
      _____   _____   _____   _____
     |     | |     | |     | |     |
FP1: | $83 | | $88 | |  0  | |  0  |   (+15)
     |_____| |_____| |_____| |_____|

       X1      M1
```

FADD subroutine (address $F46E)

Purpose: To add two numbers in floating point form.

Entry: The two addends are in FP1 and FP2 respectively. For maximum precision, both should be normalized.

Uses: SWPALGN, ADD, NORM, RTLOG.

Exit: The normalized sum is left in FP1. FP2 contains the addend of greatest magnitude. E is altered but sign is not. The A-REG is altered and the X-REG is cleared. The sum mantissa is truncated to 24 bits.

Caution: Overflow may result if the sum is less that -2^128 or greater than +2^128-1. If so, a jump to location $3F5 is executed leaving 0 in X1, and twice the proper sum in the mantissa M1. The sign bit is left in the carry, 0 for positive, 1 for negative.

```
       _____    __________
      |     |  |          |
FP1:  |  0  |  | X.YYY... |
      |_____|  |__________|

        X1       M1
```

(For carry=0, true sum=+X.YYY x 2^128)

Example: Prior to calling FADD, FP1 contains +12 and FP2 contains -5.

```
       _____    _____    _____    _____
      |     |  |     |  |     |  |     |
FP1:  | $83 |  | $60 |  |  0  |  |  0  |   (+12)
      |_____|  |_____|  |_____|  |_____|

        X1       M1

       _____    _____    _____    _____
      |     |  |     |  |     |  |     |
FP2:  | $82 |  | $B0 |  |  0  |  |  0  |   (-5)
      |_____|  |_____|  |_____|  |_____|

        X2       M2
```

After calling FADD, FP1 contains +7 (FP2 contains +12).

```
       _____    _____    _____    _____
      |     |  |     |  |     |  |     |
FP1   | $82 |  | $70 |  |  0  |  |  0  |   (+7)
      |_____|  |_____|  |_____|  |_____|

        X1       M1
```

FSUB subroutine (address $F468)

Purpose: To subtract two floating point numbers.

Entry: The minuend is in FP1 and the subtrahend is in FP2. Both should be normalized to retain maximum precision prior to calling FSUB.

Uses: FCOMPL, ALGNSWP, FADD, ADD, NORM, RTLOG.

Exit: The normalized difference is in FP1 with the mantissa truncated to 24 bits. FP2 holds either the minued or the negated subtrahend, whichever is of greater magnitude. E is altered but SIGN and SCR are not. the A-REG is altered and the X-REG is cleared. The Y-REG is not disturbed.

Cautions: An exit to location S3F5 is taken if the result is less than -2^128 or greater than +2^128-1. or if the subtrahend is -2^128.

Example: Prior to calling FSUB, FP1 contains +7 (minuend) and FP2 contalns -5 (subtrahend).

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $82 |  | $70 |  |  0  |  |  0  |   (+12)
     |_____|  |_____|  |_____|  |_____|

       X1       M1

      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP2: | $82 |  | $B0 |  |  0  |  |  0  |   (- 5)
     |_____|  |_____|  |_____|  |_____|

       X2       M2
```

After calling FSUB, FP1 contains +12 and FP2 contains +7.

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $83 |  | $60 |  |  0  |  |  0  |   (+12)
     |_____|  |_____|  |_____|  |_____|

       X1       M1
```

FMUL subroutine (address $F48C)

Purpose: To multiply floating point numbers.

Entry: The multiplicand and multiplier must reside in FP1 and FP2 respectively. Both should be normalized prior to calling FMUL to retain maximum precision.

Uses: MD1, MD2, RTLOG1, ADD, MDEND.

Exit: The signed normalized floating point product is left in FP1. M1 is truncated to contain the 24 most significant mantissa bits (including sign). The absolute value of the multiplier mantissa (M2) is left in FP2. E, SIGN, and SCR are altered. The A- and X-REGs are altered and the Y-REG contains $FF upon exit.

Cautions: An exit to location $3F5 is taken if the product is less than -2^128 or greater than +2^128-1.

Notes: FMUL will run faster if the absolute value of the multiplier mantissa contains fewer '1's than the absolute value of the multiplicand mantissa.

Example: Prior to calling FMUL, FP1 contains +12 and FP2 contains -5.

```
           _____    _____    _____    _____
          |     |  |     |  |     |  |     |
    FP1:  | $83 |  | $60 |  |  0  |  |  0  |   (+12)
          |_____|  |_____|  |_____|  |_____|

             X1       M1

           _____    _____    _____    _____
          |     |  |     |  |     |  |     |
    FP2:  | $82 |  | $B0 |  |  0  |  |  0  |   (- 5)
          |_____|  |_____|  |_____|  |_____|

             X2       M2
```

After calling FMUL, FP1 contains -60 and FP2 contains +5.

```
           _____    _____    _____    _____
          |     |  |     |  |     |  |     |
    FP1:  | $85 |  | $88 |  |  0  |  |  0  |   (-60)
          |_____|  |_____|  |_____|  |_____|

             X1       M1

           _____    _____    _____    _____
          |     |  |     |  |     |  |     |
    FP2:  | $82 |  | $50 |  |  0  |  |  0  |   (+ 5)
          |_____|  |_____|  |_____|  |_____|

             X2       M2
```

FDIV subroutine (addr $F4B2)

Purpose: To perform division of floating point numbers.

Entry: The normalized dividend is in FP2 and the normalized divisor is in FP1.

Exit: The signed normalized floating point quotient is left in FP1. The mantissa (M1) is truncated to 24 bits. The 3-bit M1 extension (E) contains the absolute value of the divisor mantissa. MD2, SIGN, and SCR are altered. The A- and X-REGs are altered and the Y-REG is cleared.

Uses: MD1, MD2, MDEND.

Cautions: An exit to location $3F5 is taken if the quotient is less than -2^128 or greater than +2^128-1

Notes: MD2 contains the remainder mantissa (equivalent to the MOD function). The remainder exponent is the same as the quotient exponent, or 1 less if the dividend mantissa magnitude is less than the divisor mantissa magnitude.

Example: Prior to calling FDIV, FP1 contains -60 (dividend), and FP2 contains +12 (divisor).

```
            _____    _____    _____    _____
          |      |  |      |  |      |  |      |
FP1:  | $85 |  | $80 |  |  0   |  |  0   |   (-60)
          |_____|  |_____|  |_____|  |_____|

            X1        M1

            _____    _____    _____    _____
          |      |  |      |  |      |  |      |
FP2   | $83 |  | $60 |  |  0   |  |  0   |   (+12)
          |_____|  |_____|  |_____|  |_____|

            X1        M1
```

After calling FMUL, FP1 contains -5 and M2 contains 0.

```
            _____    _____    _____    _____
          |      |  |      |  |      |  |      |
FP1:  | $82 |  | $B0 |  |  0   |  |  0   |   (-5)
          |_____|  |_____|  |_____|  |_____|

            X1        M1
```

FLOAT Subroutine (address $F451)

Purpose: To convert integers to floating point representation.

Entry: A signed (two's complement) 2-byte integer is stored in M1 (high-order byte) and M1+1 (low-order byte). M1+2 must be cleared by user prior to entry.

Uses: NORM1.

Exit: The normalized floating point equivalent is left in FP1. E, FP2, SIGN, and SCR are not disturbed. The A-REG contains a copy of the high-order mantissa byte upon exit but the X- and Y-REGs are not disturbed. The carry is cleared.

Notes: To float a 1-byte integer, place it in M1+1 and clear M1 as well as M1+2 prior to calling FLOAT.

FLOAT takes approximately 3 msec. lonqer to convert zero to floating point form than other arguments. The user may check for zero prior to calling FLOAT and increase throughput.

```
           *
           *  LOW-ORDER INT. BYTE IN A-REG
           * HIGH-ORDER BYTE IN Y-REG
           *
85 FA      XFLOAT  STA  M1+1
```

```
84 F9              STY  M1    INIT MANT1
A0 00              LDY  #$0
84 FB              STY  M1+2
05 D9              ORA  M1    CHK BOTH
                              BYTES FOR
D0 03              BNE  TOFLOAT  ZERO
85 F8              STA  X1    IF SO CLR X1
60                 RTS        AND RETURN
4C 51 F4  TOFLOAT  JMP  FLOAT ELSE FLOAT
                              INTEGER
```

Example: Float +274 ($0112 hex)

```
             CALLING SEQUENCE

A0 01              LDY  #$01  HIGH-ORDER
                              INTEGER BYTE
A9 12              LDA  #$12  LOW-ORDER
                              INTEGER BYTE
84 F9              STY M1
85 FA              STA M1+1
A9 00              LDA #$00
85 F8              STA M1+2
20 51 F4           JSR FLOAT
```

Upon returning from FLOAT, FP1 contains the floating point representation of +274.

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1  | $88 |  | $44 |  | $80 |  |  0  |   (+274)
     |_____|  |_____|  |_____|  |_____|

       X1       M1
```

FIX subroutine (address $F640)

Purpose: To extract the integer portion of a floating point number with truncation (ENTIER function).

Entry: A floating point value is in FP1. It need not be normalized.

Uses: RTAR.

Exit: The two-byte signed two's complement representation of the integer portion is left in M1 (high-order byte) and M1+1 (low-order byte). The floating point values +24.63 and -61.2 are converted to the integers +24 and -61 respectively. FP1 and E are altered but FP2, E, SIGN, and SCR are not. The A- and X-REGs are altered but the Y-REG is not.

Example: The floating point value +274 is in FP1 prior to calling FIX.

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $88 |  | $44 |  | $80 |  |  0  |   (+274)
```

```
     |_____|  |_____|  |_____|  |_____|

       X1       M1
```

After calling FIX, M1 (high-order byte) and M1+1 (low-order byte) contain the integer representation of +274 ($0112).

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $8E |  | $01 |  | $12 |  |  0  |
     |_____|  |_____|  |_____|  |_____|

       X1       M1
```

Note: FP1 contains an unnormalized representation of +274 upon exit.

NORM Subroutine (address $F463)

Purpose: To normalize the value in FP1, thus insuring maximum precision.

Entry: A normalized or unnormalized value is in FP1.

Exit: The value in FP1 is normalized. A zero mantissa will exit with X1=0 (2 exponent). If the exponent on exit is -128 (X1=0) then the mantissa (M1) is not necessarily normalized (with the two high-order mantissa bits unequal). E, FP2, SIGN, AND SCR are not distubed. The A-REG is disturbed but the X- and Y-REGs are not. The carry is set.

Example: FP1 contains +12 in unnormalized form (as .0011 x 2).

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $86 |  | $0C |  |  0  |  |  0  |   (+12)
     |_____|  |_____|  |_____|  |_____|

       x1       M1
```

Upon exit from NORM, FP1 contains +12 in normalized form (as 1.1 x 2).

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $83 |  | $60 |  |  0  |  |  0  |   (+12)
     |_____|  |_____|  |_____|  |_____|

       X1       M1
```

NORM1 subroutine (address $F455)

Purpose: To normalize a floating point value in FP1 when it is known the exponent is not -128 (X1=0) upon entry.

Entry: An unnormalized number is in FP1. The exponent byte should not be 0 for normal use.

Exit: The normalized value is in FP1. E, FP2, SIGN, and SCR are not not disturbed. The A-REG is altered but the X- and Y-REGs are not.

ADD Subroutine (address $F425)

Purpose: To add the two mantissas (M1 and M2) as 3-byte integers.

Entry: Two mantissas are in M1 (through M1+2) and M2 (through M2+2). They should be aligned, that is with identical exponents, for use in the FADD and FSUB subroutines.

Exit: the 24-bit integer sum is in M1 (high-order byte in M1, low-order byte in M1+2). FP2, X1, E, SIGN and SCR are not disturbed. The A-REG contains the high-order byte of the sum, the X-REG contains $FF and the Y-REG is not altered. The carry is the '25th' sum bit.

Example: FP1 contains +5 and FP2 contains +7 prior to calling ADD.

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP1: | $82 |  | $50 |  |  0  |  |  0  |   (+5)
     |_____|  |_____|  |_____|  |_____|

       X1       M1

      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP2: | $82 |  | $70 |  |  0  |  |  0  |   (+7)
     |_____|  |_____|  |_____|  |_____|
```

Upon exit, M1 contains the overflow value for +12. Note that the sign bit is incorrect. This is taken care of with a call to the right shift routine.

```
      _____    _____    _____    _____
     |     |  |     |  |     |  |     |
FP:  | $82 |  | $C0 |  |  0  |  |  0  |   (+12)
     |_____|  |_____|  |_____|  |_____|
```

ABSWAP Subroutine (address $F437)

Purpose: To take the absolute value of FP1 and then swap FP1 with FP2. Note that two sequential calls to ABSWAP will take the absolute values of both FP1 and FP2 in preparation for a multiply or divide.

Entry: FP1 and FP2 contain floating point values.

Exit: The absolute value of the original FP1 contents are in FP2 and the original FP2 contents are in FP1. The least significant bit of SIGN is complemented if a negation takes place (if the original FP1 contents are negative) by means of an increment. SCR and E are used. The A-REG contains a copy of X2, the X-REG is cleared, and the Y-REG is not altered.

RTAR Subroutine (address $F47D)

Purpose: To shift M1 right one bit position while incrementing X1 to compensate for scale. This is roughly the opposite of the NORM subroutine.

Entry: A normalized or unnormalized floating point value is in FP1.

Exit: The 6-byte field MANT1 and E is shifted right one bit arithmetically and X1 is incremented by 1 to retain proper scale. The sign bit of MANT1 (MSB of M1) is unchanged. FP2, SIGN, and SCR are not disturbed. The A-REG contains the least significant byte of E (E+2), the X-REG is cleared, and the Y-REG is not disturbed.

Caution: If X1 increments of 0 (overflow) then an exit to location $3F5 is taken, the A-REG contains the high-order MANT1 byte, M1 and X1 is cleared. FP2, SIGN, SCR, and the X- and Y-REGs are not disturbed.

Uses: RTLOG

Example: Prior to calling RTAR, FP1 contains the normalized value -7.

```
            _____    _____    _____    _____
           |     |  |     |  |     |  |     |
     FP1   | $83 |  | $A0 |  |  0  |  |  0  |   (-7)
           |_____|  |_____|  |_____|  |_____|

             X1       M1
```

After calling RTAR, FP1 contains the unnormalized value -7 (note that precision is lost off the low-order end of M1).

```
            _____    _____    _____    _____
           |     |  |     |  |     |  |     |
     FP1   | $84 |  | $D0 |  |  0  |  |  0  |   (-7)
           |_____|  |_____|  |_____|  |_____|

             X1       M1
```

Note: M1 sign bit is unchanged.

RTLOG subroutine (address $F480)

Purpose: To shift the 6-byte field MANT1 and E one bit to the right (toward the least significant bit). The 6502 carry bit is shifted into the high-order M1 bit. This is useful in correcting binary sum overflows.

Entry: A normalized or unnormalized floating point value is in FP1. The carry must be cleared or set by the user since it is shifted Into the sign bit of M1.

Exit: Same as RTAR except that the sign of M1 is not preserved (it is set to the value of the carry bit on entry)

Caution: Same as RTAR.

Example: Prior to calling RTLOG, FP1 contains the normalized value -12 and the carry is clear.

```
        _____   _____   _____   _____
       |     | |     | |     | |     |
 FP1:  | $83 | | $A0 | |  0  | |  0  |   (-12)
       |_____| |_____| |_____| |_____|

         X1      M1
```

After calling RTLOG, M1 is shifted one bit to the right and the sign bit is clear. X1 is incremented by 1.

```
        _____   _____   _____   _____
       |     | |     | |     | |     |
 FP1:  | $84 | | $50 | |  0  | |  0  |   (+20)
       |_____| |_____| |_____| |_____|

         X1      M1
```

Note: The bit shifted off the end of MANT1 is rotated into the high-order bit of the 3-byte extension E. The 3-byte E field is also shifted one bit to the right.

RTLOG1 subroutine (address $F484)

Purpose: To shift MANT1 and E right one bit without adjusting X1. This is used by the multiply loop. The carry is shifted into the sign bit of MANT1.

Entry: M1 and E contain a 6-byte unsigned field. E is the 3-byte low-order extension of MANT1.

Exit: Same as RTLOG except that X1 is not altered and an overflow exit cannot occur.

MD2 subroutine (address $F4E2)

Purpose: To clear the 3-byte MANT1 field for FMUL and FDIV, check for inital result exponent overflow (and underflow), and initialize the X-REG to $17 for loop counting.

Entry: the X-REG is cleared by the user since it is placed in the 3 bytes of MANT1. The A-REG contains the result of an exponent addition (FMUL) or subtraction (FDIV). The carry and sign status bits should be set according to this addition or subtraction for overflow and underflow determination.

Exit: The 3 bytes of M1 are cleared (or all set to the contents of the X-REG on Entry) and the Y-REG is loaded with $17. The sign bit of the

A-REG is complemented and a copy of the A-REG is stored in X1. FP2, SIGN, SCR, and the X-REG are not disturbed.

Uses: NORM.

Caution: Exponent overflow results in an exit to location $3F5. Exponent underflow results in an early return from the calling subroutine (FDIV or FMUL) with a floating point zero in FP1. Because MD2 pops a return address off the stack, it may only be called by another subroutine.

###