SPRA011

Floating-Point Arithmetic

with the TMS32020

Digital Signal Processing Application Report

TEXAS INSTRUMENTS

Floating-Point Arithmetic with the TMS32020

Charles Dana Crowell
Digital Signal Processing
Applications Engineering

TEXAS
INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

TI warrants performance of its semiconductor products, including SNJ and SMJ devices, to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems such testing necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor device might be or are used.

INTRODUCTION

The TMS32020 Digital Signal Processor is a fixed-point 16/32-bit microprocessor. However, it can also perform floating-point computations at a speed comparable to some dedicated floating-point processors.

The purpose of this application report is to analyze an implementation of floating-point addition, multiplication, and division on the TMS32020. The floating-point single-precision standard proposed by the IEEE will be examined. Using this standard, the TMS32020 performs a floating-point multiplication in 7.8 microseconds, a floating-point addition in 15.4 microseconds, and a floating-point division in 22.8 microseconds.

To illustrate floating-point formats and the tradeoffs involved in making a choice between different floating-point formats, a review of floating-point arithmetic notation and of addition, multiplication, and division algorithms is first presented.

FLOATING-POINT NOTATION

The floating-point number f may be written in floating-point format as

$$f = m \times b^e$$

where

m = mantissa
b = base
e = exponent

For example, 6,789,320 may be written as

$$0.6789320 \times 10^7$$

In this case,

m = 0.6789320
b = 10
e = 7

The two floating-point numbers f_1 and f_2 may be written as

$$f_1 = m_1 \times b^{e1}$$
$$f_2 = m_2 \times b^{e2}$$

Floating-point addition/subtraction, multiplication, and division for f_1 and f_2 are defined as follows:

$$f_1 \times f_2 = (m_1 \pm m_2 \times b^{-(e_1 - e_2)}) \times b^{e1} \text{ if } e_1 \geq e_2 \quad (1)$$

or

$$= (m_1 \times b^{-(e_2 - e_1)}) \pm m_2) \times b^{e2} \text{ if } e_1 < e_2$$

$$f_1 \times f_2 = m_1 \times m_2 \times b^{(e_1 + e_2)} \quad (2)$$

$$f_1/f_2 = (m_1/m_2) \times b^{(e_1 - e_2)} \quad (3)$$

A cursory examination of these expressions reveals some of the factors involved in the implementation of floating-point arithmetic. For addition, it is necessary to shift the mantissa of the floating-point number which has the smaller exponent to the right by the difference in the magnitude of the two exponents. This is shown in the multiplication by the terms

$$b^{-(e_1 - e_2)} \text{ and } b^{-(e_2 - e_1)}$$

This right shift can result in mantissa underflow. There are also possibilities for mantissa overflow. Addition and subtraction of exponents can lead to exponent underflow and overflow. To alleviate underflow and overflow, it is necessary to decide on some scheme for roundoff. For a detailed description and analysis of underflow and overflow conditions and rounding schemes, see reference 1.

It is desirable to have all numbers normalized, i.e., the mantissas of f_1 and f_2 have the most significant digit in the leftmost position. This provides the representation with the greatest accuracy possible for a fixed mantissa length. The result of any floating-point operation must also be normalized. The factors associated with normalization, overflow, and other characteristics of floating-point implementations are best illustrated with a few examples.

Consider the addition of two binary floating-point numbers f_1 and f_2 where

$$f_1 = 0.10100 \times 2^{011}$$
$$f_2 = 0.11100 \times 2^{001}$$

Both of these numbers are normalized, i.e., the first bit after the binary point is a 1. Addition requires equal exponents, so the fractions are aligned by shifting right the one with the smaller exponent and adjusting the smaller exponent. This yields

$$f_2 = 0.00111 \times 2^{011}$$

Then,

$$f_1 + f_2 = 0.10100 \times 2^{011} + 0.00111 \times 2^{011}$$
$$= 0.11011 \times 2^{011} = f_3$$

The sum may overflow the left end by one digit, thus requiring a postaddition adjustment or renormalization step. Since it is assumed that the register is only of a finite length, this renormalization will result in the loss of the lowest order bit.

Another example illustrates the overflow past the most significant bit. With an assumed register length of five, let

$$f_1 = 0.11100 \times 2^{011}$$
$$f_2 = 0.10101 \times 2^{001}$$

Then,

$$\begin{array}{rl} & 0.11100 \quad \times 2^{011} = f_1 \\ + & \underline{0.0010101 \times 2^{011} = f_2} \\ & \underline{1}.0000\underline{101} \times 2^{011} = f_3 \end{array}$$

The significance of the two digits underlined in the right part of the mantissa is suspect, since it is assumed that the corresponding bits of f_1 are zero. The left underlined digit is the overflow past the most significant bit. To finish the addition, f_3 is shifted to the right and the exponent adjusted accordingly. Thus,

$$1.0000101 \times 2^{011} = f_3$$

The shift of the fraction and the adjustment of the exponent yield

$$0.10000101 \times 2^{100} = f_3$$

The result may be rounded, giving

$$0.10001 \times 2^{100} = f_3$$

or truncated, giving

$$0.10000 \times 2^{100} = f_3$$

FLOATING-POINT ALGORITHMS

Multiplication Algorithm

The algorithm for normalized floating-point multiplication is illustrated in Figure 1. This algorithm is an implementation of Equation 2 in the section on floating-point notation. The floating-point numbers being multiplied are A and B written as

$$A = m_A \times b^{e_A} \text{ and } B = m_B \times b^{e_B}$$

The result is

$$C = m_C \times b^{e_C}$$

For the resulting m_C, there are three special cases. The m_C may be zero, in which case there is a branch to Step 10 to set $C = 0$. If $m_C \neq 0$, then the most significant bit will

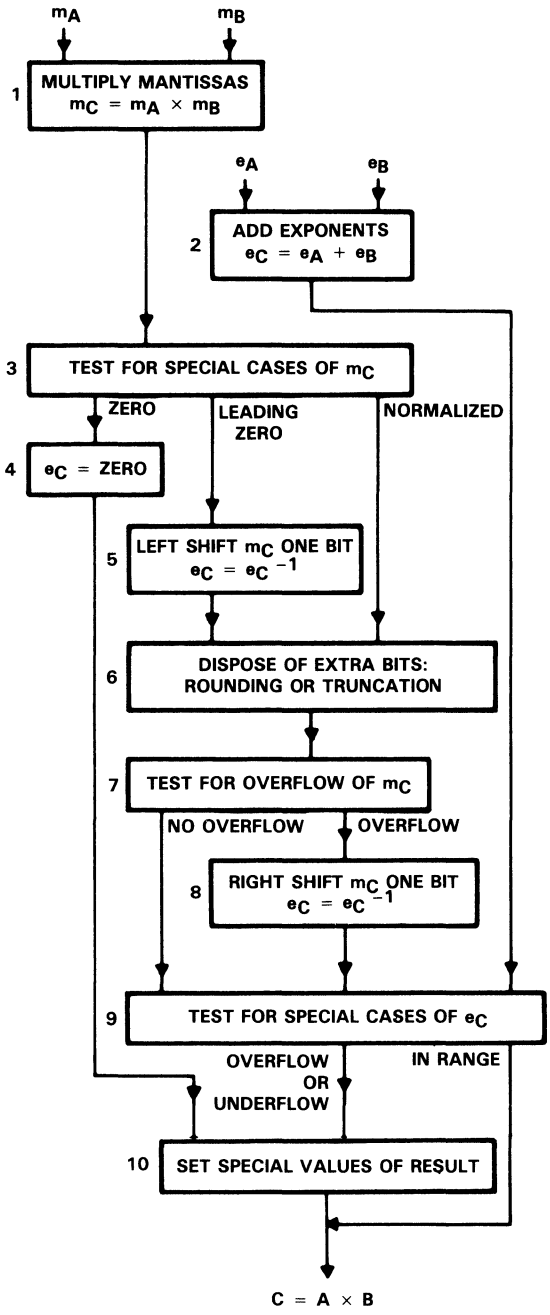

Figure 1. Floating-Point Multiplication

be in either the first or second leftmost bit. If the most significant bit is in the second leftmost bit, then a left shift of m_C is necessary (see Step 5). Otherwise, C is already in normalized form, and there is a branch to Step 6.

In Step 6, the desired rounding scheme is implemented. After this rounding, it is possible that m_C will overflow (see Step 7). In this case, it is necessary to right-shift m_C one bit (see Step 8). Special cases of e_C, are tested for in Step 9.

If there is an overflow or underflow of e_C, it is corrected in Step 10. Otherwise, the result is in range, and the calculation is complete.

Addition Algorithm

The implementation of normalized floating-point addition is more involved than for multiplication. This addition algorithm, outlined in Figure 2, is an implementation of Equation 1 in the section on floating-point notation.

In Step 1, e_A and e_B are compared to determine e_C. For this illustration of the algorithm, it is assumed that $e_A \leq e_B$. The right shift (d) required to align m_A is determined in Step 2. The procedure in Step 3 implements the right shift of m_A. In Step 4, the extra bits of m_A are discarded by using the desired rounding technique. The mantissas of A and B are then added in Step 5.

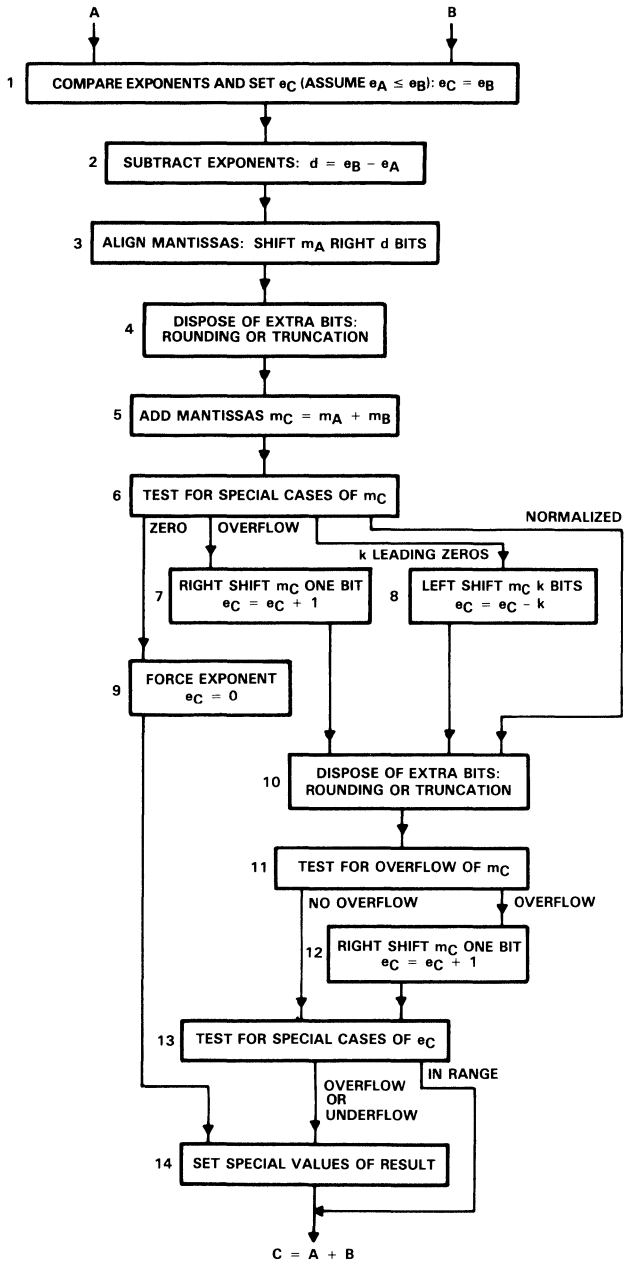

Figure 2. Floating-Point Addition

Now, the procedure becomes somewhat more involved. The m_C may be zero, in which case there is a branch to Step 9 which sets $e_C = 0$; a branch to Step 14 sets the special value of the result. The m_C may overflow, making a right shift of one necessary (see Step 7). The m_C may have k leading zeroes; therefore, a left shift of k is required. This normalization step is generally the most involved and time-consuming step to perform. The procedures in Steps 10, 11, and 12 round the m_C, test for a possible overflow due to the rounding, and adjust e_C accordingly. The special case of e_C is determined in Step 13. Finally, after Step 14, the sum C = A + B is formed.

Division Agorithm

Floating-point division is more sophisticated than multiplication and addition since fixed-point processors such as the TMS32020 are not inherently capable of performing division. For example, 1/3 = 0.3333...; only an approximation can be calculated since 1/3 must be represented in a finite number of terms. Several algorithms can be implemented to find good approximations of such numbers. The algorithm implemented in this report is shown in Figure 3.

Step 1 shows the equivalent of A/B. In Step 2, the latter term is expanded using a power series of 1/(1 + X), where ϵ (BLO/BHI) is X (ϵ simply denotes that the term is right-shifted 16 bits forming the least significant bits of a 32-bit number). The third term in the power series only affects the LSB of a 32-bit result; therefore, this term and all the following terms can be dropped, as shown in Step 3.

The equation in Step 3 can be implemented on the TMS32020 in two steps. Assuming that the result is a 32-bit number Q and that it is composed of a 16-bit QHI and a 16-bit QLO, think of the equation in Step 3 in the following manner: A/B = Q − ϵX. The first term is a fair approximation of the result Q, and the second term is a correction term to obtain a better approximation. With this in mind, it can be shown that (AHI + ϵALO)/BHI will give a 16-bit quotient and a 16-bit remainder. Due to the architecture of the TMS32020, the 16-bit quotient will be in the low word of the accumulator and the remainder will be in the high word of the accumulator after the division. Since it is desirable

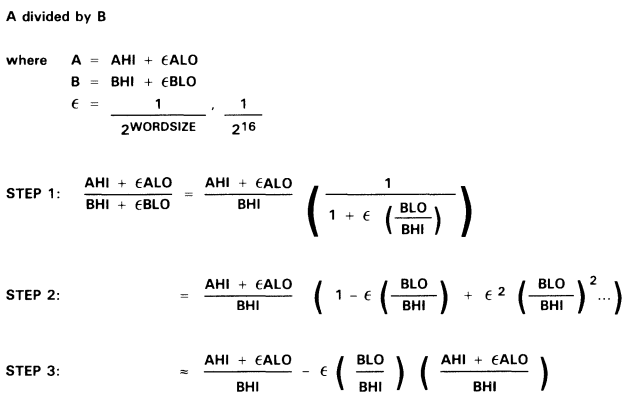

Figure 3. Division Equation

to have a floating-point result, the remainder must be divided by BHI to obtain the low word of the quotient. Now QHI and QLO have been calculated. When placing Q into the correction term (equation in Step 3), note that Q is equal to QHI + QLO. It can be shown that QLO will have no effect on the result since the correction term is multiplied by ϵ. Therefore, to calculate A divided by B, simply implement the following equation:

$$\frac{A}{B} = \frac{A}{BHI} - \epsilon\left(\frac{BLO}{BHI} \times QHI\right)$$

where the division is fixed binary (left-shifts and subtracts).

Figure 4 shows the implementation of the division algorithm that was outlined in Figure 3.

In Step 1, the dividend is right-shifted four times to prevent an overflow. Note that the result is not shifted left to compensate for this shift, because the normalization routine automatically does this. The shift causes the dividend to be limited to 27 significant bits instead of 31. In Step 2, a binary divide (left-shifts and subtracts) is implemented on the dividend by the high 16 bits of the divisor. The 32-bit result contains a quotient in the low 16 bits of the accumulator, and a remainder (R1) in the high 16 bits of the accumulator. R1 is left-shifted fifteen places in Step 3. The new R1 is divided by BHI in Step 4 to calculate the lower 16 bits of the quotient.

The quotient has now been approximated. The 32-bit result is composed of QHI and QLO, as shown in Figure 3. To obtain a better approximation, one term in the power series expansion must be added to the quotient. Therefore, the procedure in Step 5 calculates a 16-bit correction term, which is then added (or subtracted since it is the term following the "1" in the power series) to the 32-bit quotient.

Testing for an overflow of the resulting mantissa is necessary. Since the dividend was left-shifted four places, the resulting quotient will not be negative if an overflow occurred. To detect an overflow, bit 28 in the quotient must be tested. If this bit is a 1, an overflow occurred; if it is a 0, no overflow occurred. If an overflow has occurred, the exponent must be incremented. Finally, it is necessary to normalize the quotient and output the results.

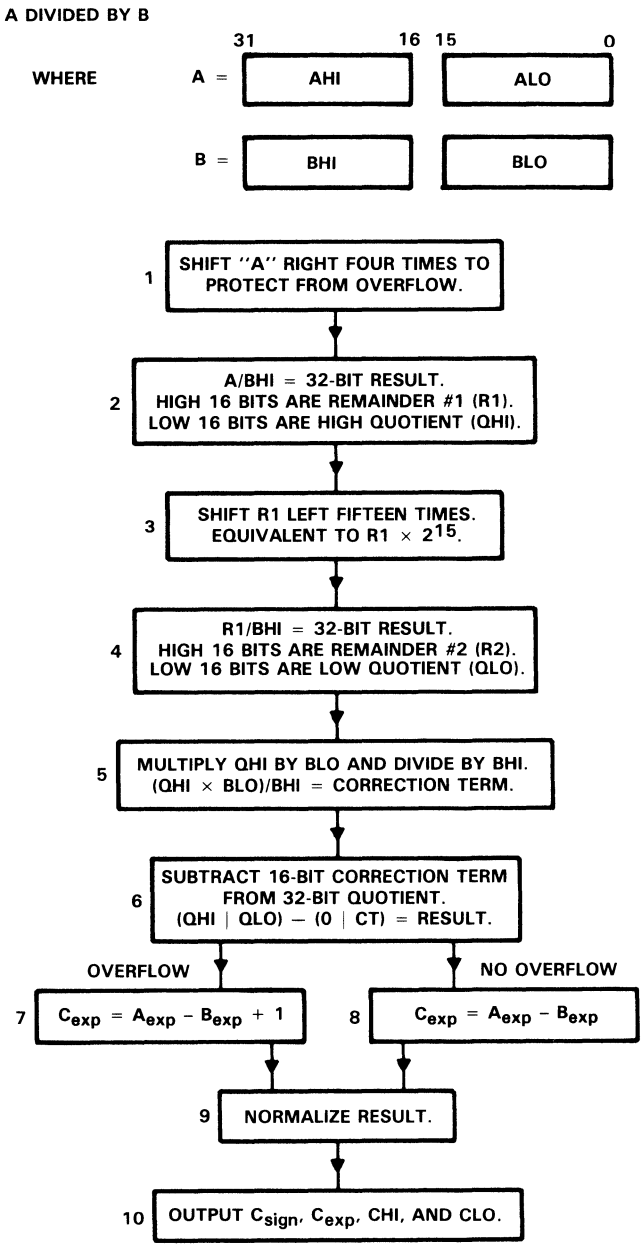

Figure 4. Floating-Point Division

IEEE FLOATING-POINT SINGLE-PRECISION FORMAT

Of interest is a set of formats known as the IEEE standard. This IEEE recommended format consists of a variety of precision formats (single, double, single-extended, and double-extended). The IEEE has also proposed several techniques for handling special cases such as overflow, underflow, $\pm \infty$, and rounding. For complete details, the reader is referred to the proposed IEEE standard.[2]

The single-precision format is a 32-bit format consisting of a 1-bit sign field s, an 8-bit biased exponent e, and a 23-bit fraction f (see Figure 5). The value of a binary floating-point number X is determined as follows:

$$X = (-1)^s \times 2^{(e-127)} \times 1.f$$

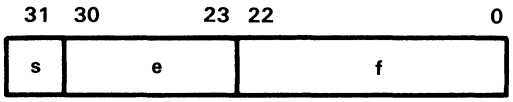

Figure 5. IEEE Floating-Point Single-Precision Format

The advantage of this format is that it is structured in such a way as to provide easy storage and straightforward input/output operations on 8-, 16- and 32-bit processors. The disadvantage with this format is that the large mantissa will generally span several words of memory.

FLOATING-POINT IMPLEMENTATION

IEEE Implementation

The IEEE single-precision format is described here as it applies to the addition, multiplication, and division algorithms. In these floating-point routines written for the TMS32020, all results are truncated to 31 bits to provide more flexibility in the user's development of a rounding scheme suitable for his application. The representations of $\pm \infty$ are ignored so that the user can decide how to handle these exceptions in a manner that is appropriate for his particular application.

I/O Considerations

The first consideration is the internal representation of the binary floating-point number. If the number is read into the TMS32020 as two 16-bit words, some processing is then necessary to put the floating-point number into a representation which is easier to process. The representation used in the TMS32020 programs in the appendices is shown in Figure 6. This internal representation may be arrived at by a simple manipulation of the IEEE bit fields. For this particular algorithm, it is assumed that the floating-point number is input to the TMS32020 as the four 16-bit fields shown in Figure 6. However, the user can easily supply his own routine to arrive at this format from two 16-bit inputs to the TMS32020 where the inputs contain the IEEE single-precision format.

The format in Figure 6 was chosen to minimize the execution time of the floating-point addition, multiplication, and division routines. The format of the result is shown in Figure 7. Notice that it is identical to the format in Figure 5 except for CLO. CLO has its 16 most significant bits valid for both the addition, multiplication, and division routines.

Normalization

Since the floating-point routines require normalization, a partial binary search algorithm is implemented in the addition and division routines in the appendices. To begin the normalization routine, note that all mantissas can be considered to be positive with the format used for the result shown in Figure 7. The binary search for the most significant bit (the leftmost 1 since the mantissa is positive) is illustrated in Figure 8.

The first move is to split the result into CHI and CLO. If CHI $\neq$ 0, the most significant bit (MSB) is the CHI; otherwise, it is the CLO. For this example, it is in CLO.

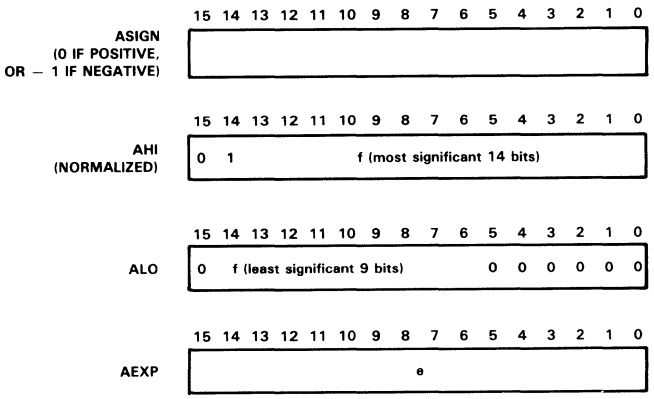

Figure 6. Floating-Point Representation

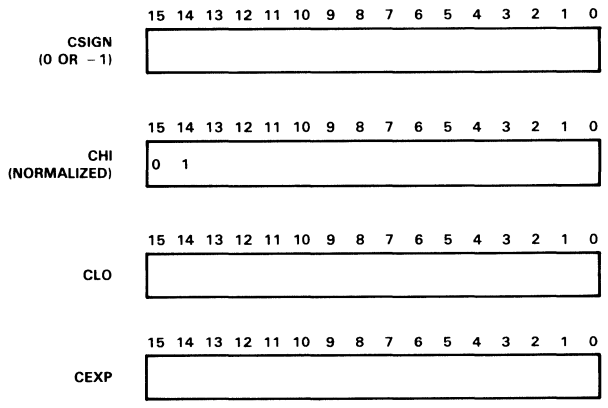

Figure 7. Result Representation

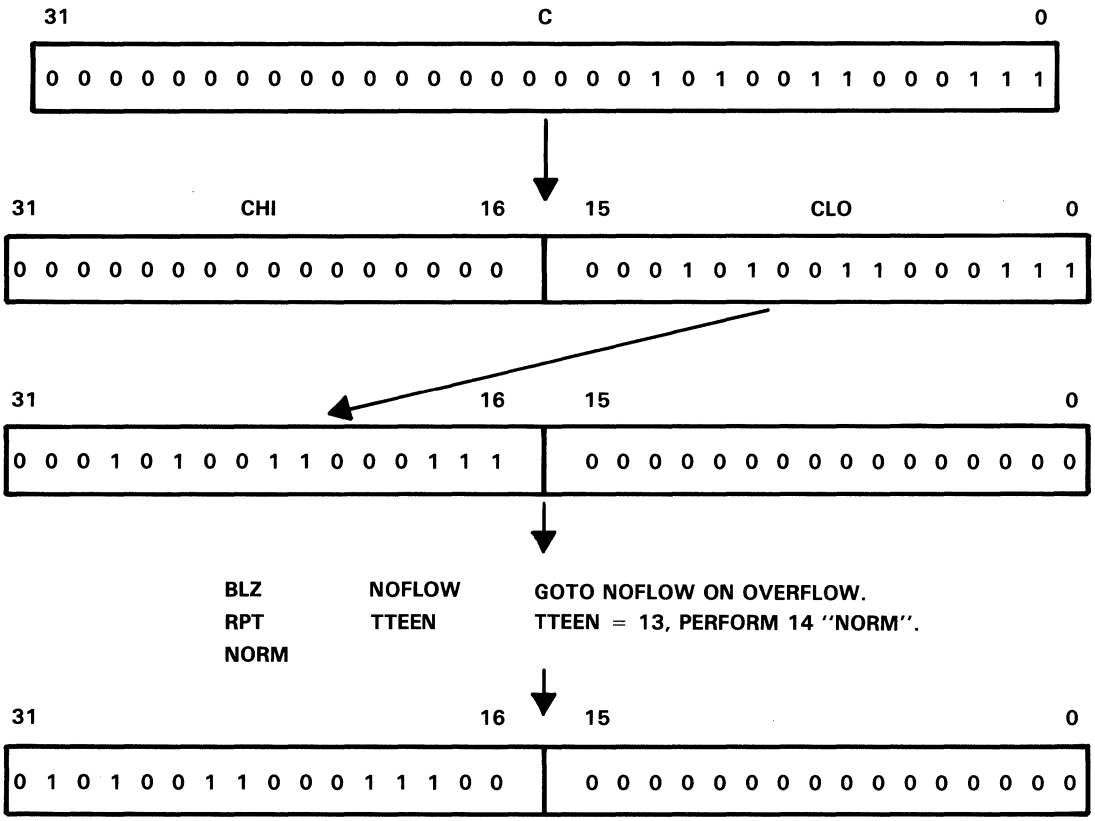

Figure 8. Partial Binary Search

The next step is to form a 32-bit result with CLO in the most significant word position. It is now possible for the MSB to be in the highest bit location since CLO has been left-shifted 16 times. If this is the case, an overflow has occurred, and the result must be right-shifted once. The normalization routine tests this by branching to NOFLOW if the result is negative. If the number is not negative, the normalization can continue.

The NORM instruction is used in the repeat mode to complete the normalization. Note that this whole normalization routine can be replaced by the following two instructions: RPTK 29 and NORM. The RPTK instruction causes the NORM instruction to be repeated 30 times, thus normalizing a 32-bit number. This method is not implemented here due to the timing. These two instructions always take 31 cycles to normalize a 32-bit number. The normalization routine here takes only 22 cycles (worst case) for normalizing a 32-bit number. Therefore, if program space is more important than timing efficiency, it is best to replace the normalization routine with these two instructions.

Added Precision

As illustrated in Figure 7, the 16 most significant bits of CLO are valid, i.e., C is valid for 31 places beyond the binary point. Oftentimes the user is not as concerned with the IEEE standard as in being certain that he has enough accuracy for his particular application. Since the TMS32020 uses 16-bit words, the routines in the appendices implicitly maintain a 30-bit mantissa. They also implicitly use a 16-bit exponent. If the user desires this added accuracy and dynamic range, then it is readily implementable with no additional cost in execution time. The normalization for the addition, as mentioned previously, operates over the entire 32-bit accumulator. For the strict IEEE format, the user will only want to normalize over the 25 most significant bits of the accumulator. The structure of the normalization routine makes this modification simple.

The routines in the appendices make no provision for the representation of $\pm$ ∞ and exponent underflow and overflow. The user of the routines should consider the degree of significance of these results and the way they should be handled for his particular application. Since these routines are written to operate at maximum speed, truncation of results is used. If the user desires to implement a rounding scheme, then he will also need to check for the possibility of overflow due to the rounding scheme. This step is shown in the multiplication, addition, and division flowcharts (see Figures 1, 2, and 3).

SUMMARY

The TMS32020 may be used to perform floating-point operations with great accuracy, wide dynamic range, and high-speed execution. The design engineer has the responsibility of deciding what type of floating-point format is best for his application. To aid in understanding floating-point operations, several examples have been given that illustrate the manipulations necessary to implement floating-point addition, multiplication, and division algorithms. Flowcharts for these algorithms are also included. The appendices contain the TMS32020 code for the IEEE floating-point single-precision format used in addition, multiplication, and division. The addition and multiplication routines may also be used without modification to implement a format with up to a 30-bit mantissa and a 16-bit exponent without any increase in execution time.

ACKNOWLEDGEMENTS

Major portions of this application report were taken from "Floating-Point Arithmetic with the TMS32010," an application report written by Ray Simar, Jr. The author would also like to thank Gwyn Guidy for her assistance with the floating-point division algorithm.

REFERENCES

1. D.J. Kuck, *The Structure of Computers and Computations*, **Vol 1**, John Wiley & Sons (1978).
2. J. Coonen et al, "A Proposed Standard for Binary Floating-Point Arithmetic," *ACM Signum Newsletter*, 4-12 (October 1979).
3. Donald E. Knuth, *Seminumerical Algorithms*, **Vol 2**, 2nd Edition, Addison-Wesley (1981).

APPENDIX A

```
NO$IDT      32020 FAMILY MACRO ASSEMBLER   PC0.7 84.348    15:24:29  03-27-85
             *** PRERELEASE ***
                                                                  PAGE 0001

0001            ************************************************************
0002            *                                                          *
0003            *    THIS IS A FLOATING-POINT ADDITION ROUTINE WHICH       *
0004            *    IMPLEMENTS THE IEEE PROPOSED FLOATING-POINT FORMAT    *
0005            *    ON THE TMS32020.                                      *
0006            *                                                          *
0007            ************************************************************
0008            *
0009            *    INITIAL FORMAT (ALL 16-BIT WORDS)
0010            *    ---------------
0011            *    |   ALL 0 OR 1   |       ASIGN (0 OR -1)
0012            *    ---------------
0013            *
0014            *    ---------------
0015            *    |0|.   15 BITS   |       AHI (NORMALIZED)
0016            *    ---------------
0017            *
0018            *    ---------------
0019            *    |0|  9 BITS |--0-|       ALO
0020            *    ---------------
0021            *
0022            *    ---------------
0023            *    |               |       AEXP (-127 TO 128)
0024            *    ---------------
0025            *
0026            *    TO CORRESPOND WITH IEEE FORMAT,
0027            *    INPUT 0.1F * 2 ** (E + 1)
0028            *    INSTEAD OF 1.F * 2 **E, AND SUBTRACT 127 FROM E.
0029            *
0030            *    THE FINAL FORMAT IS THE SAME AS THE INITIAL FORMAT
0031            *    EXCEPT THAT FOR CLO WE HAVE:
0032            *
0033            *    ---------------
0034            *    |   16 BITS     |       CLO
0035            *    ---------------
0036            *
0037            *    ALL 16 BITS OF CLO ARE VALID. ANYTHING PAST THESE HAS
0038            *    BEEN TRUNCATED.
0039            *
0040            ************************************************************
0041            *                                                          *
0042            *    WORST CASE (EXCLUDING INITIALIZATION AND I/O):        *
0043            *    15.4 MICROSECONDS.                                    *
0044            *    THIS TIMING INCLUDES THE NORMALIZATION.               *
0045            *    WORDS OF PROGRAM MEMORY: 170                          *
0046            *                                                          *
0047            ************************************************************
0048            *
0049 0000                  AORG
0050       0000  ASIGN     EQU     0
0051       0001  AEXP      EQU     1
0052       0002  AHI       EQU     2
0053       0003  ALO       EQU     3
0054       0004  BSIGN     EQU     4
0055       0005  BEXP      EQU     5
0056       0006  BHI       EQU     6
```

```
0057        0007  BLO     EQU     7
0058        0008  CSIGN   EQU     8
0059        0009  CEXP    EQU     9
0060        000A  CHI     EQU     10
0061        000B  CLO     EQU     11
0062        000C  D       EQU     12
0063        000D  ONE     EQU     13
0064        000E  TEMP    EQU     14
0065        000F  THREE   EQU     15
0066        0010  SIXT    EQU     16
0067        0011  RESID   EQU     17
0068        0012  TTEEN   EQU     18
0069              *
0070              *
0071              *       INITIALIZATION
0072              *
0073              *
0074              *
0075 0000 C804           LDPK    4               BEGIN ON PAGE 4.
0076 0001 CE07           SSXM                    SET SIGN EXTENSION.
0077 0002 5589           LARP    1
0078 0003 D100           LRLK    AR1,>200
     0004 0200
0079 0005 CB07           RPTK    7
0080 0006 80A0           IN      *+,PA0
0081 0007 5588           LARP    0
0082 0008 C000           LARK    AR0,0           CLEAR EXPONENT REGISTER.
0083 0009 CA01           LACK    1
0084 000A 600D           SACL    ONE             ONE = 1
0085 000B CA10           LACK    16
0086 000C 6010           SACL    SIXT
0087 000D CA03           LACK    3
0088 000E 600F           SACL    THREE
0089 000F CA0D           LACK    13
0090 0010 6012           SACL    TTEEN
0091              *
0092              *
0093              *       BEGIN FLOATING POINT ADD
0094              *
0095              *
0096 0011 2001  UP       LAC     AEXP            FIND LARGEST NUMBER.
0097 0012 1005           SUB     BEXP
0098 0013 600C           SACL    D
0099 0014 F680           BZ      AEQB            IF EXPONENTS ARE THE SAME, JUMP TO AEQB.
     0015 002D
0100 0016 F380           BLZ     ALTB            IF A IS LESS THAN B, JUMP TO ALTB.
     0017 0037
0101              *
0102 0018 CE23  AGTB     NEG
0103 0019 0010           ADD     SIXT            D = (16-D)
0104 001A 600C           SACL    D
0105 001B 3C0C           LT      D
0106 001C 2000           LAC     ASIGN           A IS LARGER THAN B.
0107 001D 6008           SACL    CSIGN           THEREFORE, CSIGN = ASIGN.
0108 001E 2001           LAC     AEXP            ALIGN THE B MANTISSA.
0109 001F 6009           SACL    CEXP
```

```
0110 0020 4206          LACT    BHI         BHI IS SHIFTED RIGHT "D" TIMES.
0111 0021 6806          SACH    BHI
0112 0022 6011          SACL    RESID       RESIDUAL BITS MUST BE MAINTAINED.
0113 0023 4207          LACT    BLO         BLO IS SHIFTED RIGHT "D" TIMES.
0114 0024 CE18          SFL                 MSB (THE 0) IS SHIFTED AWAY.
0115 0025 6807          SACH    BLO
0116 0026 2007          LAC     BLO
0117 0027 4D11          OR      RESID       GET BITS THAT WERE SHIFTED FROM BHI.
0118 0028 6007          SACL    BLO
0119 0029 2103          LAC     ALO,1       GET RID OF EXTRA BIT.
0120 002A 6003          SACL    ALO
0121 002B FF80          B       CHKSGN      DO BOTH NUMBERS HAVE THE SAME SIGN?
     002C 0049
0122              *
0123 002D 2000  AEQB    LAC     ASIGN       IF SIGNS ARE THE SAME, CSIGN = ASIGN
0124 002E 6008          SACL    CSIGN
0125 002F 2103          LAC     ALO,1       ALIGN MANTISSAS.
0126 0030 6003          SACL    ALO
0127 0031 2107          LAC     BLO,1
0128 0032 6007          SACL    BLO
0129 0033 2001          LAC     AEXP        SET C EXPONENT =  A EXPONENT.
0130 0034 6009          SACL    CEXP
0131 0035 FF80          B       CHKSGN      DO BOTH NUMBERS HAVE THE SAME SIGN?
     0036 0049
0132              *
0133 0037 0010  ALTB    ADD     SIXT
0134 0038 600C          SACL    D           D = (16-D)
0135 0039 3C0C          LT      D
0136 003A 2004          LAC     BSIGN       B IS THE BIGGEST NUMBER.
0137 003B 6008          SACL    CSIGN       THEREFORE, LET THE SIGN OF C = BSIGN.
0138 003C 2005          LAC     BEXP        SET C EXPONENT = B EXPONENT.
0139 003D 6009          SACL    CEXP
0140 003E 4202          LACT    AHI         AHI GETS SHIFTED "D" TIMES.
0141 003F 6802          SACH    AHI
0142 0040 6011          SACL    RESID       MAINTAIN EXTRA BITS.
0143 0041 4203          LACT    ALO         ALO GETS SHIFTED "D" TIMES.
0144 0042 CE18          SFL                 MSB (THE 0) IS SHIFTED AWAY.
0145 0043 6803          SACH    ALO
0146 0044 2003          LAC     ALO
0147 0045 4D11          OR      RESID       GET RESIDUAL BITS.
0148 0046 6003          SACL    ALO
0149 0047 2107          LAC     BLO,1       GET RID OF EXTRA BIT.
0150 0048 6007          SACL    BLO
0151              *
0152 0049 2000  CHKSGN   LAC     ASIGN       CHECK THE SIGNS.
0153 004A 1004          SUB     BSIGN
0154 004B F680          BZ      ADNOW       IF THEY ARE THE SAME, JUST ADD.
     004C 007A
0155 004D F380          BLZ     AISNEG
     004E 005D
0156 004F 4002  BISNEG  ZALH    AHI         DO (|A| - |B|),
0157 0050 4903          ADDS    ALO         SINCE B < 0  AND  A > 0.
0158 0051 4507          SUBS    BLO
0159 0052 4406          SUBH    BHI
0160 0053 F680          BZ      CZERO
     0054 006B
```

```
0161 0055 F380          BLZ     CNEG
     0056 0072
0162 0057 680A          SACH    CHI
0163 0058 600B          SACL    CLO
0164 0059 CA00          ZAC
0165 005A 6008          SACL    CSIGN
0166 005B FF80          B       NORMAL      GO AND NORMALIZE RESULT.
     005C 0084
0167 005D 4006  AISNEG  ZALH    BHI         DO (!B! - !A!),
0168 005E 4907          ADDS    BLO         SINCE A < 0  AND  B > 0.
0169 005F 4503          SUBS    ALO
0170 0060 4402          SUBH    AHI
0171 0061 F680          BZ      CZERO
     0062 006B
0172 0063 F380          BLZ     CNEG
     0064 0072
0173 0065 680A          SACH    CHI
0174 0066 600B          SACL    CLO
0175 0067 CA00          ZAC
0176 0068 6008          SACL    CSIGN
0177 0069 FF80          B       NORMAL      GO AND NORMALIZE RESULTS.
     006A 0084
0178              *
0179 006B CA00  CZERO   ZAC                 HERE, ONLY IF RESULT = 0.
0180 006C 6009          SACL    CEXP
0181 006D 6008          SACL    CSIGN
0182 006E 600A          SACL    CHI
0183 006F 600B          SACL    CLO
0184 0070 FF80          B       AROUND      OUTPUT A ZERO.
     0071 00A7
0185              *
0186 0072 CE1B  CNEG    ABS                 HERE, IF RESULT IS NEGATIVE.
0187 0073 680A          SACH    CHI
0188 0074 600B          SACL    CLO
0189 0075 D001          LALK    >FFFF
     0076 FFFF
0190 0077 6008          SACL    CSIGN
0191 0078 FF80          B       NORMAL      GO NORMALIZE RESULT.
     0079 0084
0192              *
0193 007A 4002  ADNOW   ZALH    AHI         IF SIGNS ARE THE SAME, JUST ADD.
0194 007B 4903          ADDS    ALO
0195 007C 4907          ADDS    BLO
0196 007D 4806          ADDH    BHI
0197 007E 680A          SACH    CHI
0198 007F 600B          SACL    CLO
0199 0080 F080          BV      OVFLOW      DID AN OVERFLOW OCCUR?
     0081 0095
0200 0082 F680          BZ      CZERO       IS RESULT = 0 ?
     0083 006B
0201              *
0202              *       NORMALIZE
0203              *
0204 0084 200A  NORMAL  LAC     CHI         DOES CHI HAVE THE MSB?
0205 0085 F680          BZ      LOi
     0086 008D
```

```
0206 0087 400A          ZALH    CHI         IF YES, NORMALIZE RESULT.
0207 0088 490B          ADDS    CLO
0208 0089 4B12          RPT     TTEEN       WILL PERFORM 14 "NORMS"
0209 008A CEA2          NORM
0210 008B FF80          B       OUTPUT      GO OUTPUT RESULTS.
     008C 00A1
0211 008D 400B  L01     ZALH    CLO         HERE IF CLO HAS MSB.
0212 008E C010          LARK    AR0,16      OFFSET EXPONENT BY 16.
0213 008F F380          BLZ     NOFLOW      DID BIT SEARCH CAUSE OVERFLOW?
     0090 009E
0214 0091 4B12          RPT     TTEEN       IF NOT, NORMALIZE RESULT.
0215 0092 CEA2          NORM
0216 0093 FF80          B       OUTPUT      GO OUTPUT RESULT.
     0094 00A1
0217              *
0218              *
0219              *       FINISHED WITH NORMALIZATION
0220              *
0221              *       HERE ONLY IF OVERFLOW OCCURRED DURING ADDITION
0222              *
0223              *
0224 0095 CE06  OVFLOW  RSXM                RESET SIGN EXTENSION TO SHIFT RIGHT.
0225 0096 CE19          SFR                 SHIFT RIGHT.
0226 0097 680A          SACH    CHI         STORE NORMALIZED MANTISSA.
0227 0098 600B          SACL    CLO
0228 0099 2009          LAC     CEXP        DECREMENT EXPONENT.
0229 009A 000D          ADD     ONE
0230 009B 6009          SACL    CEXP
0231 009C FF80          B       AROUND      GO OUTPUT RESULTS.
     009D 00A7
0232              *
0233              *       OVERLOW OCCURRED DURING BIT SEARCH
0234              *
0235 009E 5590  NOFLOW  MAR     *-          DECREMENT EXPONENT.
0236 009F CE06          RSXM                RSXM FOR LOGICAL RIGHT SHIFT.
0237 00A0 CE19          SFR                 PERFORM RIGHT SHIFT.
0238              *
0239              *
0240              *       TAKE CARE OF EXPONENT & NORMALIZED MANTISSA,
0241              *       THEN OUTPUT RESULTS.
0242              *
0243              *
0244 00A1 700E  OUTPUT  SAR     AR0,TEMP    HERE AFTER NORMALIZATION.
0245 00A2 680A          SACH    CHI         SAVE NORMALIZED MANTISSA.
0246 00A3 600B          SACL    CLO
0247 00A4 2009          LAC     CEXP        ADJUST EXPONENT.
0248 00A5 100E          SUB     TEMP
0249 00A6 6009          SACL    CEXP
0250              *
0251 00A7 5589  AROUND  LARP    1           RESET POINTER.
0252 00A8 4B0F          RPT     THREE
0253 00A9 E0A0          OUT     *+,PA0
0254 00AA CE1F          IDLE                WAIT FOR INTERRUPT.
NO ERRORS, NO WARNINGS
```

APPENDIX B

```
NO$IDT      32020 FAMILY MACRO ASSEMBLER   PC0.7 84.348     15:24:53  03-27-85
             *** PRERELEASE ***
                                                                  PAGE 0001

0001            ****************************************************************
0002            *                                                              *
0003            *    THIS IS A FLOATING-POINT MULTIPLICATION ROUTINE WHICH     *
0004            *    IMPLEMENTS THE IEEE PROPOSED FLOATING-POINT FORMAT        *
0005            *    ON THE TMS32020.                                          *
0006            *                                                              *
0007            ****************************************************************
0008            *
0009            *    INITIAL FORMAT (ALL 16-BIT WORDS)
0010            *    ---------------
0011            *    !   ALL 0 OR 1   !       ASIGN (0 OR -1)
0012            *    ---------------
0013            *
0014            *    ---------------
0015            *    !0!.   15 BITS   !       AHI (NORMALIZED)
0016            *    ---------------
0017            *
0018            *    ---------------
0019            *    !0!  9 BITS !--0-!       ALO
0020            *    ---------------
0021            *
0022            *    ---------------
0023            *    !               !       AEXP (-127 TO 128)
0024            *    ---------------
0025            *
0026            *    TO CORRESPOND WITH IEEE FORMAT,
0027            *    INPUT 0.1F * 2 ** (E + 1)
0028            *    INSTEAD OF 1.F * 2 **E, AND SUBTRACT 127 FROM E.
0029            *
0030            *    THE FINAL FORMAT IS THE SAME AS THE INITIAL FORMAT
0031            *    EXCEPT THAT FOR CLO WE HAVE:
0032            *
0033            *    ---------------
0034            *    !   16 BITS     !       CLO
0035            *    ---------------
0036            *
0037            *    ALL 16 BITS OF CLO ARE VALID. ANYTHING PAST THESE HAS
0038            *    BEEN TRUNCATED.
0039            *
0040            ****************************************************************
0041            *                                                              *
0042            *    WORST CASE (EXCLUDING INITIALIZATION AND I/O):            *
0043            *    7.8 MICROSECONDS.                                         *
0044            *    THIS TIMING INCLUDES THE NORMALIZATION.                   *
0045            *    WORDS OF PROGRAM MEMORY: 60                               *
0046            *                                                              *
0047            ****************************************************************
0048            *
0049 0000                  AORG
0050       0000  ASIGN     EQU     0
0051       0001  AEXP      EQU     1
0052       0002  AHI       EQU     2
0053       0003  ALO       EQU     3
0054       0004  BSIGN     EQU     4
0055       0005  BEXP      EQU     5
0056       0006  BHI       EQU     6
```

```
0057        0007  BLO     EQU     7
0058        0008  CSIGN   EQU     8
0059        0009  CEXP    EQU     9
0060        000A  CHI     EQU     10
0061        000B  CLO     EQU     11
0062        000C  THI     EQU     12
0063        000D  NEGONE  EQU     13
0064        000E  TLO     EQU     14
0065        000F  TEMP    EQU     15
0066              *
0067              *
0068              *       INITIALIZATION
0069              *
0070              *
0071              *
0072 0000 C804            LDPK    4             BEGIN ON PAGE 4.
0073 0001 CE07            SSXM                  SET SIGN EXTENSION.
0074 0002 5589            LARP    1
0075 0003 D100            LRLK    AR1,>200
     0004 0200
0076 0005 CB07            RPTK    7             READ NUMBERS INTO BLOCK B0.
0077 0006 80A0            IN      *+,PA0
0078 0007 C000            LARK    AR0,0         CLEAR EXPONENT REGISTER.
0079 0008 5588            LARP    0
0080 0009 D001            LALK    >FFFF
     000A FFFF
0081 000B 600D            SACL    NEGONE        NEGONE = -1
0082              *
0083              *
0084              *       BEGIN FLOATING-POINT MULTIPLICATION.
0085              *
0086              *
0087 000C 2001  UP        LAC     AEXP          ADD EXPONENTS.
0088 000D 0005            ADD     BEXP
0089 000E 6009            SACL    CEXP
0090              *
0091 000F 3C03            LT      ALO           FIRST PRODUCT (ALO * BHI)
0092 0010 3806            MPY     BHI
0093 0011 CE14            PAC
0094 0012 680C            SACH    THI
0095 0013 600E            SACL    TLO
0096              *
0097 0014 3C02            LT      AHI           SECOND PRODUCT (AHI * BLO)
0098 0015 3807            MPY     BLO
0099              *
0100 0016 CE15            APAC                  HAS EFFECT OF (AHI * BLO + ALO * BHI) * 2 ** -15.
0101 0017 CE15            APAC
0102              *
0103 0018 480C            ADDH    THI
0104 0019 490E            ADDS    TLO
0105 001A 680C            SACH    THI
0106              *
0107 001B 3806            MPY     BHI           (AHI * BHI)
0108 001C CE14            PAC
0109 001D 490C            ADDS    THI
0110              *
```

```
 0111 001E 690A         SACH    CHI,1       GET RID OF EXTRA SIGN BITS.
 0112 001F 610B         SACL    CLO,1
 0113              *
 0114 0020 F580         BNZ     OK          IS RESULT ZERO?
      0021 0026
 0115 0022 CA00         ZAC
 0116 0023 6009         SACL    CEXP
 0117 0024 FF80         B       SETSIN
      0025 002F
 0118              *
 0119 0026 400A  OK     ZALH    CHI         NORMALIZE AND WRAP UP.
 0120 0027 490B         ADDS    CLO
 0121 0028 CEA2         NORM
 0122 0029 680A         SACH    CHI
 0123 002A 600B         SACL    CLO
 0124 002B 700F         SAR     AR0,TEMP
 0125 002C 2009         LAC     CEXP
 0126 002D 100F         SUB     TEMP
 0127 002E 6009         SACL    CEXP
 0128              *
 0129 002F 4100  SETSIN ZALS    ASIGN       WHAT IS SIGN OF RESULT?
 0130 0030 4C04         XOR     BSIGN
 0131 0031 F580         BNZ     NEG
      0032 0037
 0132 0033 CA00         ZAC
 0133 0034 6008         SACL    CSIGN
 0134 0035 FF80         B       OUTPUT
      0036 0039
 0135 0037 200D  NEG    LAC     NEGONE
 0136 0038 6008         SACL    CSIGN
 0137 0039 5589  OUTPUT LARP    1           OUTPUT RESULTS.
 0138 003A CB03         RPTK    3
 0139 003B E0A0         OUT     *+,PA0
 0140 003C CE1F         IDLE
NO ERRORS, NO WARNINGS
```

APPENDIX C

```
NO$IDT      32020 FAMILY MACRO ASSEMBLER   PC0.7 84.348     15:25:17  03-27-85
             *** PRERELEASE ***
                                                                  PAGE 0001

0001            ************************************************************
0002            *                                                          *
0003            *    THIS IS A FLOATING-POINT DIVISION ROUTINE WHICH       *
0004            *    IMPLEMENTS THE IEEE PROPOSED FLOATING-POINT FORMAT    *
0005            *    ON THE TMS32020.                                      *
0006            *                                                          *
0007            ************************************************************
0008            *
0009            *    INITIAL FORMAT (ALL 16-BIT WORDS)
0010            *    ---------------
0011            *    !   ALL 0 OR 1  !       ASIGN (0 OR -1)
0012            *    ---------------
0013            *
0014            *    ---------------
0015            *    !0!.   15 BITS  !       AHI (NORMALIZED)
0016            *    ---------------
0017            *
0018            *    ---------------
0019            *    !0!  9 BITS !--0-!       ALO
0020            *    ---------------
0021            *
0022            *    ---------------
0023            *    !               !       AEXP (-127 TO 128)
0024            *    ---------------
0025            *
0026            *    TO CORRESPOND WITH IEEE FORMAT,
0027            *    INPUT 0.1F * 2 ** (E + 1)
0028            *    INSTEAD OF 1.F * 2 **E, AND SUBTRACT 127 FROM E.
0029            *
0030            *    THE FINAL FORMAT IS THE SAME AS THE INITIAL FORMAT
0031            *    EXCEPT THAT FOR CLO WE HAVE:
0032            *
0033            *    ---------------
0034            *    !   16 BITS     !       CLO
0035            *    ---------------
0036            *
0037            *    ALL 16 BITS OF CLO ARE VALID. ANYTHING PAST THESE HAS
0038            *    BEEN TRUNCATED.
0039            *
0040            ************************************************************
0041            *                                                          *
0042            *    WORST CASE (EXCLUDING INITIALIZATION AND I/O):        *
0043            *    22.8 MICROSECONDS.                                    *
0044            *    THIS TIMING INCLUDES THE NORMALIZATION.               *
0045            *    WORDS OF PROGRAM MEMORY: 92                           *
0046            *                                                          *
0047            ************************************************************
0048            *
0049 0000                  AORG    0
0050       0000 ASIGN     EQU     0
0051       0001 AEXP      EQU     1
0052       0002 AHI       EQU     2
0053       0003 ALO       EQU     3
0054       0004 BSIGN     EQU     4
0055       0005 BEXP      EQU     5
0056       0006 BHI       EQU     6
```

```
0057        0007  BLO     EQU     7
0058        0008  CSIGN   EQU     8
0059        0009  CEXP    EQU     9
0060        000A  CHI     EQU     10
0061        000B  CLO     EQU     11
0062        000C  NEGONE  EQU     12
0063        000D  TEMP    EQU     13
0064        000E  FOUR    EQU     14
0065        000F  QM      EQU     15
0066        0010  QL      EQU     16
0067        0011  R1      EQU     17
0068        0012  R2      EQU     18
0069        0013  CL      EQU     19
0070        0014  M1000   EQU     20
0071        0015  ONE     EQU     21
0072        0016  THREE   EQU     22
0073        0017  FITEEN  EQU     23
0074        0018  THIRTY  EQU     24
0075        0019  TTEEN   EQU     25
0076              *
0077              *
0078              *       INITIALIZATION
0079              *
0080              *
0081              *
0082 0000 C804           LDPK    4               BEGIN ON PAGE 4.
0083 0001 CE07           SSXM                    SET SIGN EXTENSION.
0084 0002 5589           LARP    1
0085 0003 D100           LRLK    AR1,>200
     0004 0200
0086 0005 CB07           RPTK    7               READ NUMBERS INTO BLOCK B0.
0087 0006 80A0           IN      *+,PA0
0088 0007 5588           LARP    0
0089 0008 C000           LARK    AR0,0           CLEAR EXPONENT REGISTER.
0090 0009 D001           LALK    >FFFF
     000A FFFF
0091 000B 600C           SACL    NEGONE          NEGONE = -1
0092 000C D001           LALK    >1000
     000D 1000
0093 000E 6014           SACL    M1000           M1000 = >1000
0094 000F CA04           LACK    4
0095 0010 600E           SACL    FOUR            FOUR = 4
0096 0011 CA01           LACK    1
0097 0012 6015           SACL    ONE             ONE = 1
0098 0013 CA03           LACK    3
0099 0014 6016           SACL    THREE           THREE = 3
0100 0015 CA0F           LACK    15
0101 0016 6017           SACL    FITEEN          FITEEN = 15
0102 0017 CA1E           LACK    30
0103 0018 6018           SACL    THIRTY          THIRTY = 30
0104 0019 CA0D           LACK    13
0105 001A 6019           SACL    TTEEN           TTEEN = 13
0106 001B CA00           ZAC
0107 001C 6009           SACL    CEXP            CLEAR CEXP
0108              *
0109              *
```

```
0110                *       FINISHED WITH INITIALIZATION
0111                *
0112                *
0113 001D 2000              LAC     ASIGN       CSIGN = ASIGN, IF ASIGN = BSIGN.
0114 001E 6008              SACL    CSIGN
0115 001F 1004              SUB     BSIGN
0116 0020 F680              BZ      OK          ELSE, CSIGN = -1.
     0021 0023
0117 0022 200C              LAC     NEGONE
0118                *       SACL    CSIGN
0119 0023 4002  OK          ZALH    AHI         SHIFT DIVIDEND TO PROTECT FROM OVERFLOW.
0120 0024 4903              ADDS    ALO
0121 0025 4B16              RPT     THREE
0122 0026 CE19              SFR
0123                *
0124 0027 4B17              RPT     FITEEN      QM = AHI¦ALO / BHI, R1 = REMAINDER.
0125 0028 4706              SUBC    BHI
0126 0029 6811              SACH    R1          HIGH ACCUMULATOR RETAINS REMAINDER.
0127 002A 600F              SACL    QM
0128 002B 2F11              LAC     R1,15       (R1 * 2**15) / BHI   GIVES QL, AND R2.
0129 002C 4B17              RPT     FITEEN      COMPUTES (R1 * 2**15) / BHI.
0130 002D 4706              SUBC    BHI
0131 002E 6812              SACH    R2          HIGH ACCUMULATOR RETAINS REMAINDER.
0132 002F 6010              SACL    QL
0133                *
0134 0030 3C0F              LT      QM          CORRECTION TERM = (QM * BLO) / BHI.
0135 0031 3807              MPY     BLO         COMPUTES (QM * BLO).
0136 0032 CE14              PAC
0137 0033 4B17              RPT     FITEEN      COMPUTES (QM * BLO) / BHI.
0138 0034 4706              SUBC    BHI
0139 0035 6013              SACL    CL
0140 0036 400F              ZALH    QM          QM¦QL - 0¦CL = CHI¦CLO
0141 0037 4910              ADDS    QL
0142 0038 1013              SUB     CL
0143 0039 600B              SACL    CLO
0144 003A 680A              SACH    CHI
0145 003B 200A              LAC     CHI         DID AN OVERFLOW OCCUR?
0146 003C 4E14              AND     M1000
0147 003D F680              BZ      NOOVF       IF NOT, GOTO NOOVF.
     003E 0041
0148 003F 2015              LAC     ONE         ELSE, INCREMENT CEXP.
0149 0040 6009              SACL    CEXP
0150 0041 2001  NOOVF       LAC     AEXP        COMPUTE RESULTING EXPONENT.
0151 0042 1005              SUB     BEXP
0152 0043 0009              ADD     CEXP
0153 0044 6009              SACL    CEXP
0154                *
0155                *
0156                *       NORMALIZE
0157                *
0158 0045 200A  NORMAL      LAC     CHI          DOES CHI HAVE THE MSB?
0159 0046 F680              BZ      LO1
     0047 004E
0160 0048 400A              ZALH    CHI          IF YES, NORMALIZE RESULT.
0161 0049 490B              ADDS    CLO
0162 004A 4B19              RPT     TTEEN        WILL PERFORM 14 "NORMS".
```

```
 0163 004B CEA2          NORM
 0164 004C FF80          B       OUTPUT      GO OUTPUT RESULTS.
      004D 0057
 0165 004E 400B  LO1     ZALH    CLO         HERE, IF CLO HAS MSB.
 0166 004F F380          BLZ     NOFLOW      DID BIT SEARCH CAUSE OVERFLOW?
      0050 0055
 0167 0051 4B19          RPT     TTEEN       IF NOT, NORMALIZE RESULT.
 0168 0052 CEA2          NORM
 0169 0053 FF80          B       OUTPUT      GO OUTPUT RESULT.
      0054 0057
 0170              *
 0171              *
 0172              *       FINISHED WITH NORMALIZATION
 0173              *
 0174              *       OVERFLOW OCCURRED DURING BIT SEARCH
 0175              *
 0176 0055 CE06  NOFLOW  RSXM                RSXM FOR LOGICAL RIGHT SHIFT.
 0177 0056 CE19          SFR                 PERFORM RIGHT SHIFT.
 0178              *
 0179              *
 0180              *       TAKE CARE OF EXPONENT & NORMALIZED MANTISSA,
 0181              *       THEN OUTPUT RESULTS.
 0182              *
 0183              *
 0184 0057 680A  OUTPUT  SACH    CHI         SAVE NORMALIZED MANTISSA.
 0185 0058 600B          SACL    CLO
 0186 0059 5589          LARP    1           RESET POINTER.
 0187 005A 4B16          RPT     THREE       OUTPUT RESULTS, CSIGN, CEXP, CHI, AND CLO.
 0188 005B E0A0          OUT     *+,PA0
 0189 005C CE1F          IDLE                WAIT FOR INTERRUPT.
NO ERRORS, NO WARNINGS
```

TEXAS INSTRUMENTS

Creating useful products
and services for you.

Printed in U.S.A.

SPRA011
1602265-9701