To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

APPLICATION NOTE

H8/300L Super Low Power Series

Multiplication of Single-Precision Floating-Point Numbers (FMUL)

Introduction

The software FMUL performs multiplication of single-precision floating-point numbers, which are placed in general-purpose registers, and places the result of multiplication in the general purpose registers.

Target Device

H8/38024

Contents

1. Arguments 2
2. Changes to Internal Registers and Flags 2
3. Specifications 2
4. Notes 3
5. Description 3
6. Flowchart 7
7. Program List 16

About Single-Precision Floating-Point Numbers <Reference> 21

1. Arguments

Description		Memory area	Data length (bytes)
Input	Multiplicand	R0, R1	4
	Multiplier	R2, R3	4
Output	Result of multiplication	R0, R1	4

2. Changes to Internal Registers and Flags

R0	R1	R2	R3	R4	R5	R6	R7
○	○	×	×	×	×	×	—

I	U	H	U	N	Z	V	C
—	—	×	—	×	×	×	×

Legend
—: No change
×: Undefined
○: Result

3. Specifications

Program memory (bytes)
348
Data memory (bytes)
0
Stack (bytes)
16
Clock cycle count
1078
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

4. Notes

The clock cycle count (16) in the specifications is for the example shown in figure 1.

For the format of floating-point numbers, see "About Single-Precision Floating-Point Numbers <Reference>."

5. Description

5.1 Details of functions

1. The following arguments are used with the software FMUL:
   a. Input arguments:
      R0: Sets the upper 2 bytes of a single-precision floating-point as multiplicand.
      R1: Sets the lower 2 bytes of the single-precision floating-point as multiplicand.
      R2: Sets the upper 2 bytes of a single-precision floating-point as multiplier.
      R3: Sets the lower 2 bytes of the single-precision floating-point as multiplier.
   b. Output arguments:
      R0: The upper 2 bytes of a single-precision floating-point are placed here as the result of multiplication.
      R1: The lower 2 bytes of a single-precision floating-point are placed here as the result of multiplication.
2. The following figure illustrates the execution of the software FMUL. When the input arguments are set as shown in (1), the result of multiplication is placed in R0 and R1 as shown in (2).

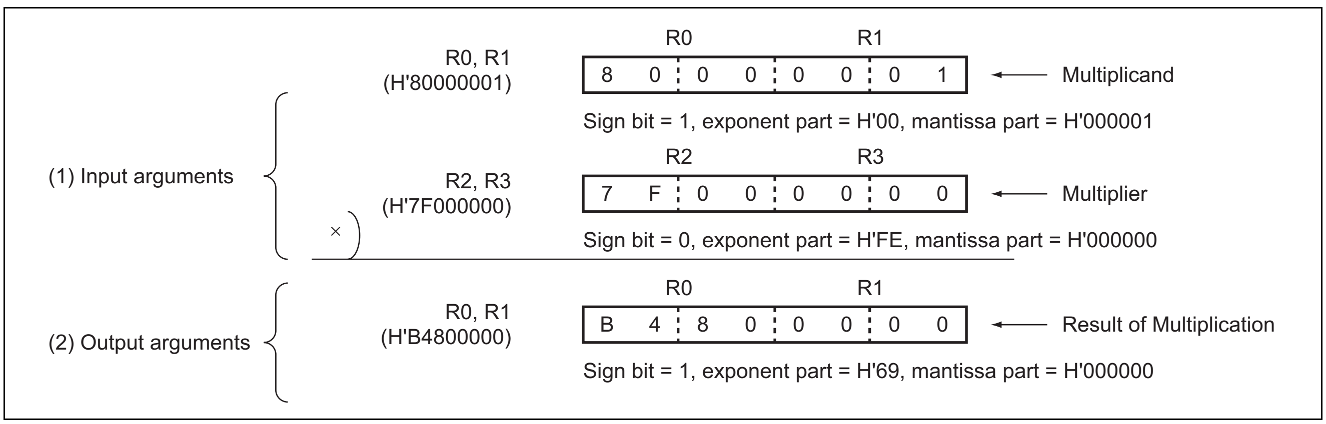


Figure 1 Example of Software FMUL Execution

5.2 Notes on usage

1. The maximum and minimum values that can be handled by the software FADD are as follows:

   Positive maximum H'7F800000
   Positive minimum H'00000001

   Negative maximum H'80000001
   Negative minimum H'FF800000

2. All positive single-precision floating-point numbers H'7F800001 to H'7FFFFFFF are treated as a maximum value (H'7F800000). All negative single-precision floating-point numbers H'FF800000 to H'FFFFFFFF are treated as a minimum value (H'FF800000).
3. As a maximum value is treated as infinity (∞),$\infty \times 100 = \infty$ or $\infty \times (-100) = -\infty$ (see table 1).

Table 1 Examples of Operation with Maximum Values Used as Arguments

Multiplicand	Multiplier	Result
>H'7F800000 (+∞)	Positive number	H'7F800000 (+∞)
	Negative number	H'FF800000 (–∞)
<H'FF800000 (–∞)	Positive number	H'FF800000 (–∞)
	Negative number	H'7F800000 (+∞)
Positive number	>H'7F800000 (+∞)	H'7F800000 (+∞)
	<H'FF800000 (–∞)	H'FF800000 (–∞)
Negative number	>H'7F800000 (+∞)	H'FF800000 (–∞)
	<H'FF800000 (–∞)	H'7F800000 (+∞)

4. H'80000000 is treated as H'00000000 (zero).
5. After execution of the software FMUL, the multiplicand and multiplier data will be lost. When the input arguments are still needed after software FMUL execution, save them in memory.

5.3 Description of data memory

The software FMUL uses no data memory.

5.4 Example of usage

Set a multiplicand and a multiplier in the general-purpose registers and call the software FMUL as a subroutine.

```
WORK1     .RES.B    2           Reserve a data memory area              a multiplicand.
WORK2     .RES.B    2           in which the user program places        a multiplier.
WORK3     .RES.B    2                                                   a result of multiplication.

          MOV.W     @WORK1,R0    Place the multiplicand set by the user
          MOV.W     @WORK1+2,R1  program in R0 and R1.

          MOV.W     @WORK2,R2    Place the multiplier set by the user
          MOV.W     @WORK2+2,R3  program in R2 and R3.

          JSR       @FMUL        Call the software FMUL as a subroutine.

          MOV.W     R0,@WORK3    Place the result of multiplication
          MOV.W     R1 @WORK3+2  set in the output argument in R0 and R1 .
              .
              .
              .
```

5.5 Operation

Multiplication of single-precision floating-point numbers is done in the following steps:

1. The software checks whether the multiplicand and multiplier are "0".
   a. If either the multiplicand or multiplier is "0", H'00000000 is output.
2. The software checks whether the multiplicand and multiplier are infinite.
   If they are infinite, the result is as given in table 1.
3. Assume that the multiplicand is R_1 (sign bit = S_1, exponent = α_1, mantissa = β_1) and the multiplier is R_2 (sign bit = S_2, exponent = α_2, mantissa = β_2). Then R_1 and R_2 are given by

   $$R_1 = (-1)^{S1} \times 2^{\alpha 1-127} \times \beta 1$$
   $$R_2 = (-1)^{S2} \times 2^{\alpha 2-127} \times \beta 2$$

   Multiplication of these two numbers is given by

   $$R_1 \times R_2 = (-1)^{S1+S2} \times 2^{\alpha 1+\alpha 2-127-127} \times \beta_1 \times \beta_2$$

   Since, in the case of the floating-point format, H'7F (D'127) is added to the result of multiplication of the exponents, the multiplication equation changes as follows:

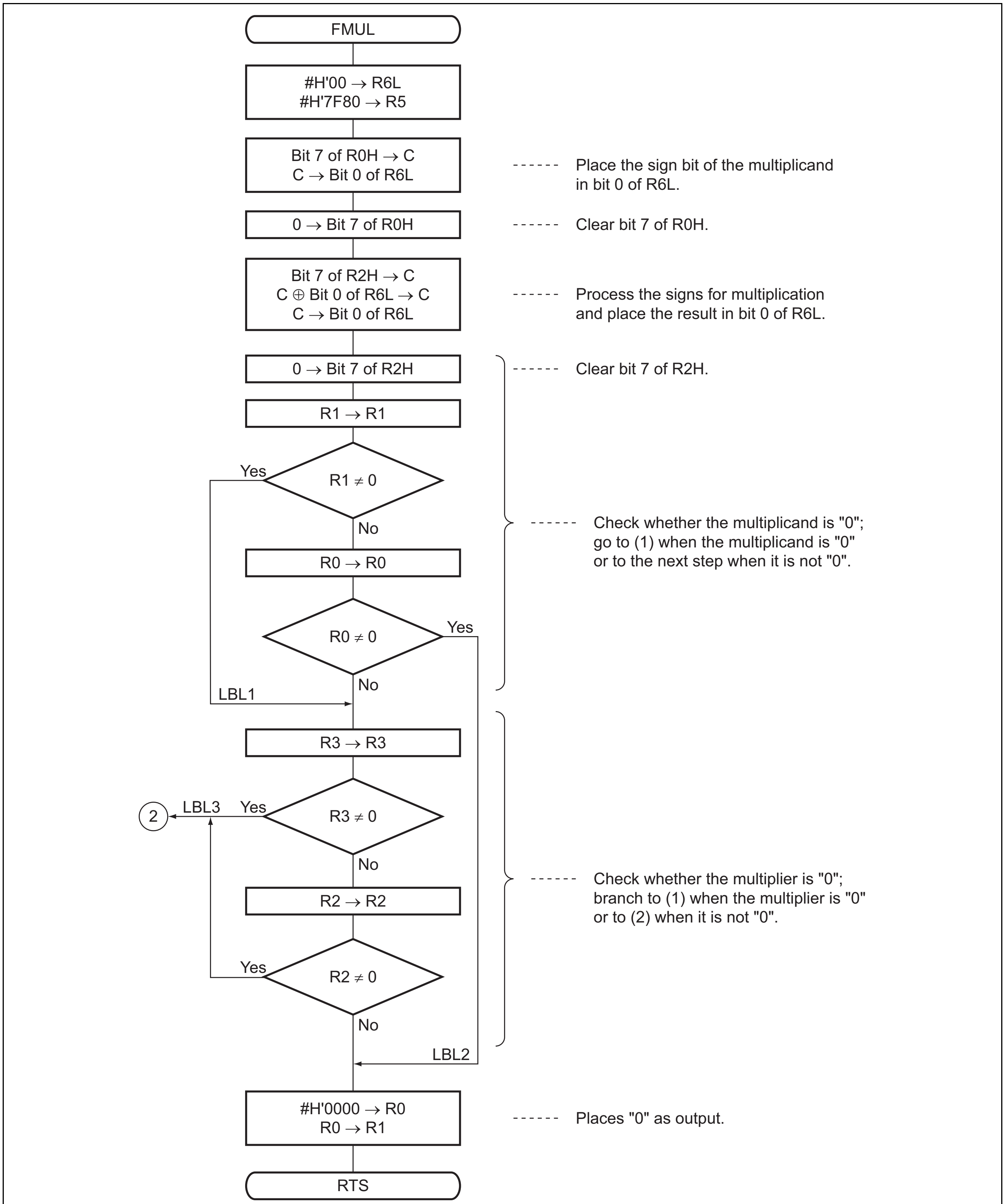
   $$R_1 \times R_2 = (-1)^{S1+S2} \times 2^{\alpha 1+\alpha 2-127} \times \beta_1 \times \beta_2$$

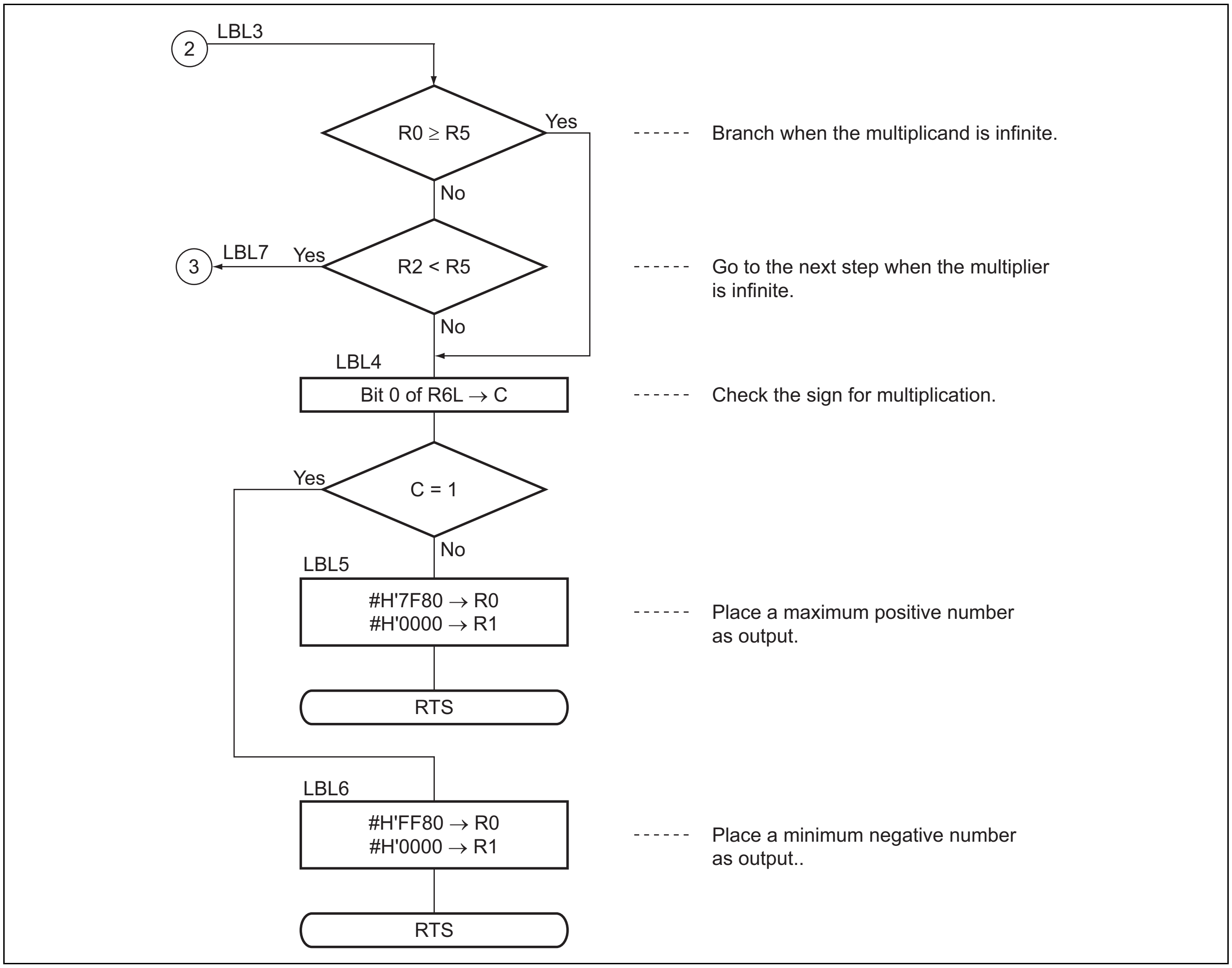
   Thus, the multiplication is performed in the steps below:
   a. The software checks the sign bits of $R_1 \times R_2$.
   b. Addition is done on the exponents.
      H'7F (D'127) is added to the actual exponent of a number in the floating-point data format; H'7F (D'127) is thus subtracted from both α_1 and α_2, and H'7F (D'127) is added to the exponent of the result. The result may thus be expressed as follows.

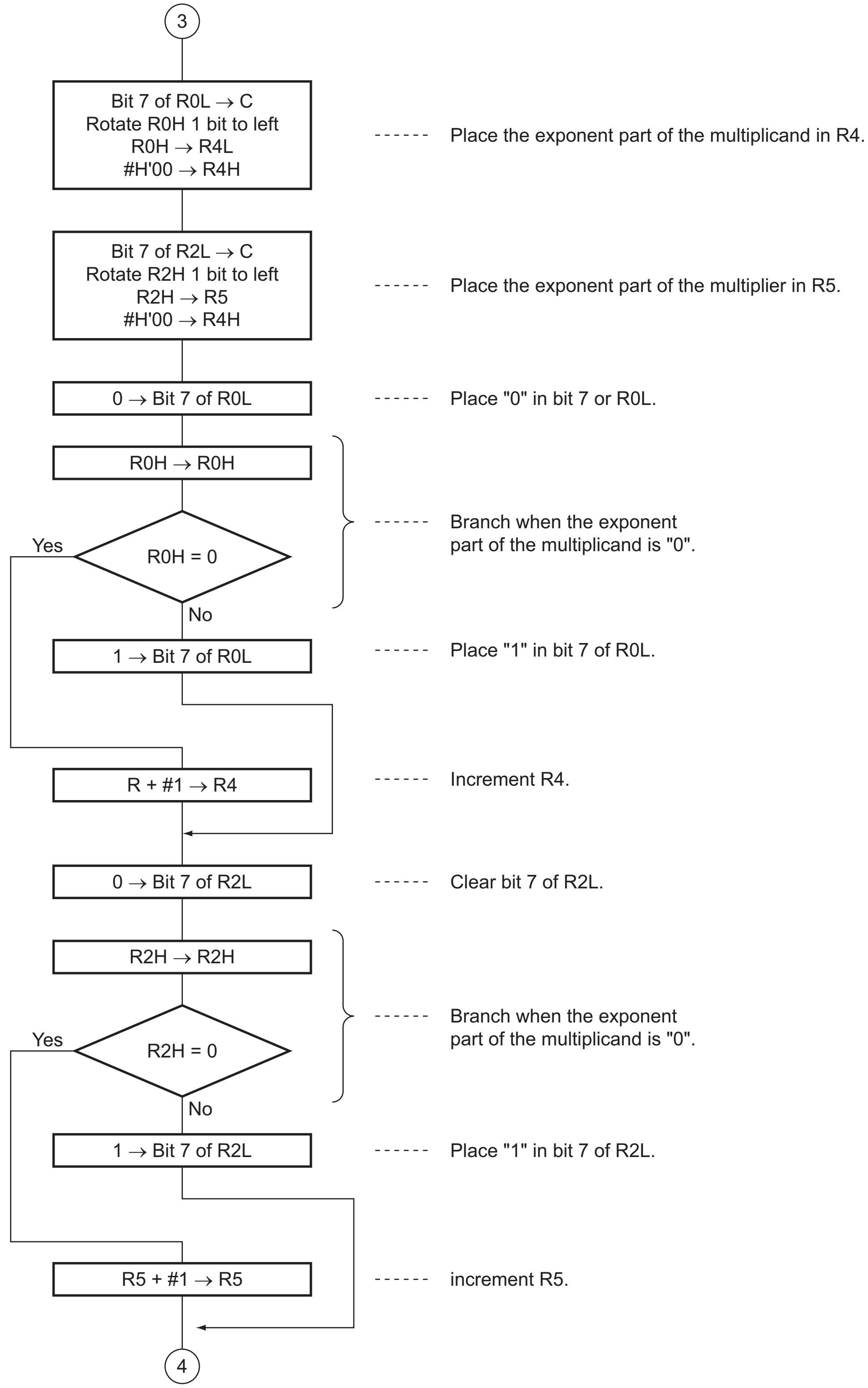
      $$(\alpha_1 - H'7F) + (\alpha_2 - H'7F) + H'7F = \alpha_1 + \alpha_2 - H'7F$$

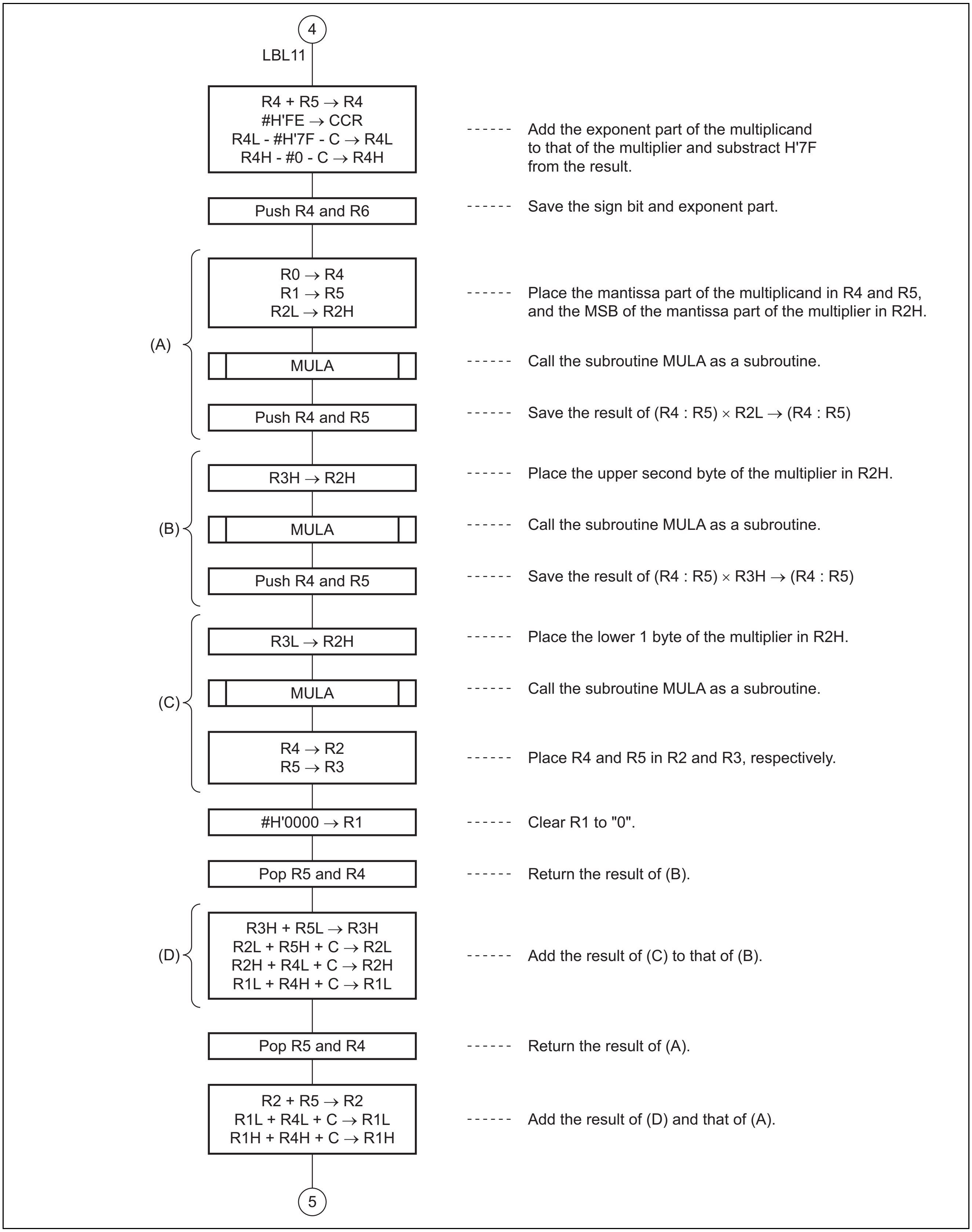
      (In the case of the denormalized format, 1 is added to the exponent before the calculation.)
   c. Multiplication is done on the mantissas.
      The implicit MSB is included in this operation.
      (In the case of the denormalized format, the implicit MSB of the mantissa is treated as "0".)
   d. The result of multiplication is corrected to produce a number in the floating-point data format.

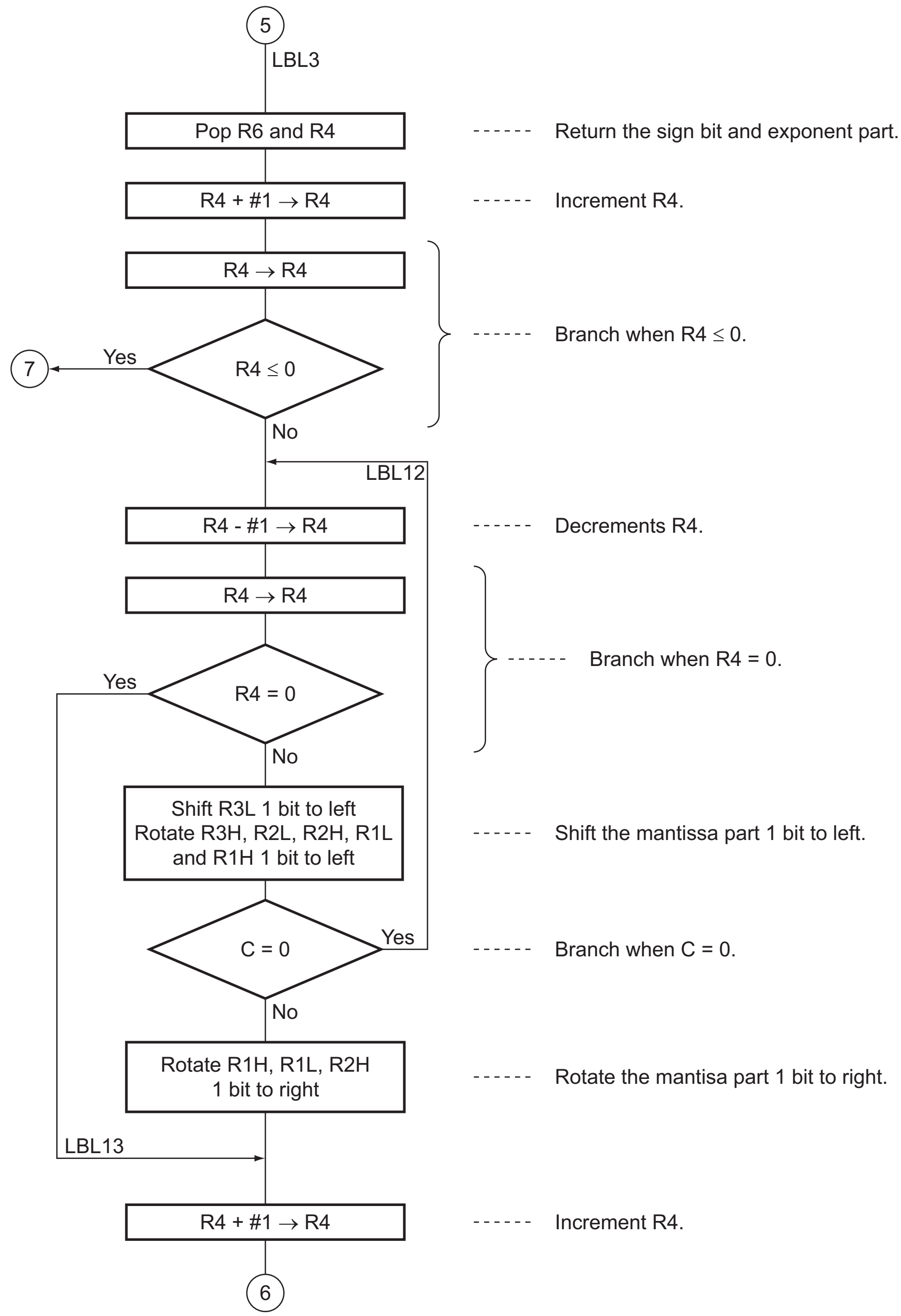
6. Flowchart

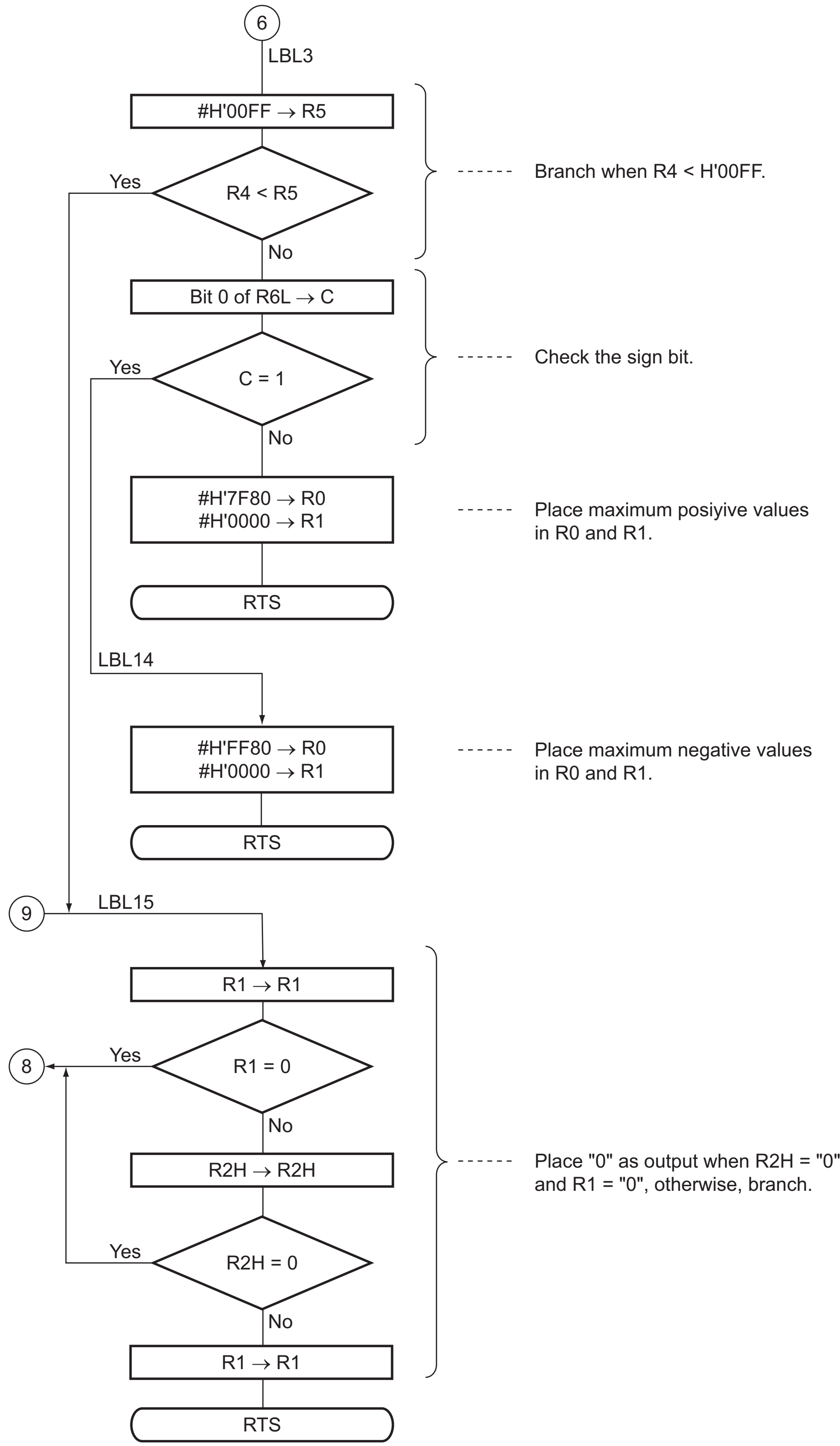

FMUL
#H'00 → R6L
#H'7F80 → R5
Bit 7 of R0H → C
C → Bit 0 of R6L
Place the sign bit of the multiplicand in bit 0 of R6L.
0 → Bit 7 of R0H
Clear bit 7 of R0H.
Bit 7 of R2H → C
C ⊕ Bit 0 of R6L → C
C → Bit 0 of R6L
Process the signs for multiplication and place the result in bit 0 of R6L.
0 → Bit 7 of R2H
Clear bit 7 of R2H.
R1 → R1
R1 ≠ 0
Yes
No
R0 → R0
R0 ≠ 0
Yes
No
LBL1
Check whether the multiplicand is "0"; go to (1) when the multiplicand is "0" or to the next step when it is not "0".
R3 → R3
R3 ≠ 0
2
LBL3
Yes
No
R2 → R2
R2 ≠ 0
Yes
No
LBL2
Check whether the multiplier is "0"; branch to (1) when the multiplier is "0" or to (2) when it is not "0".
#H'0000 → R0
R0 → R1
Places "0" as output.
RTS

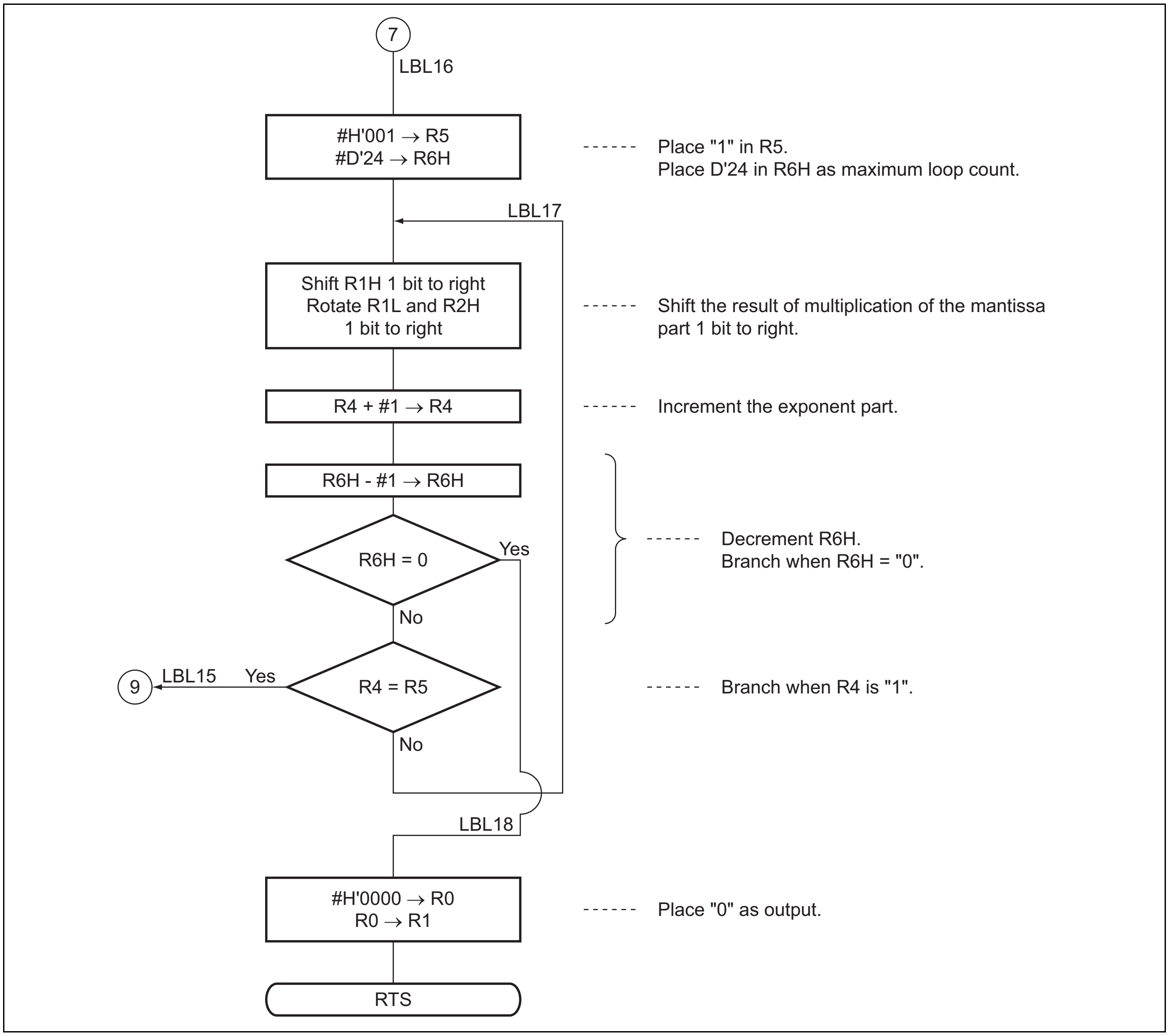

2
LBL3
R0 ≥ R5
Yes
No
Branch when the multiplicand is infinite.
3
LBL7
Yes
R2 < R5
No
Go to the next step when the multiplier is infinite.
LBL4
Bit 0 of R6L → C
Check the sign for multiplication.
Yes
C = 1
No
LBL5
#H'7F80 → R0
#H'0000 → R1
Place a maximum positive number as output.
RTS
LBL6
#H'FF80 → R0
#H'0000 → R1
Place a minimum negative number as output..
RTS

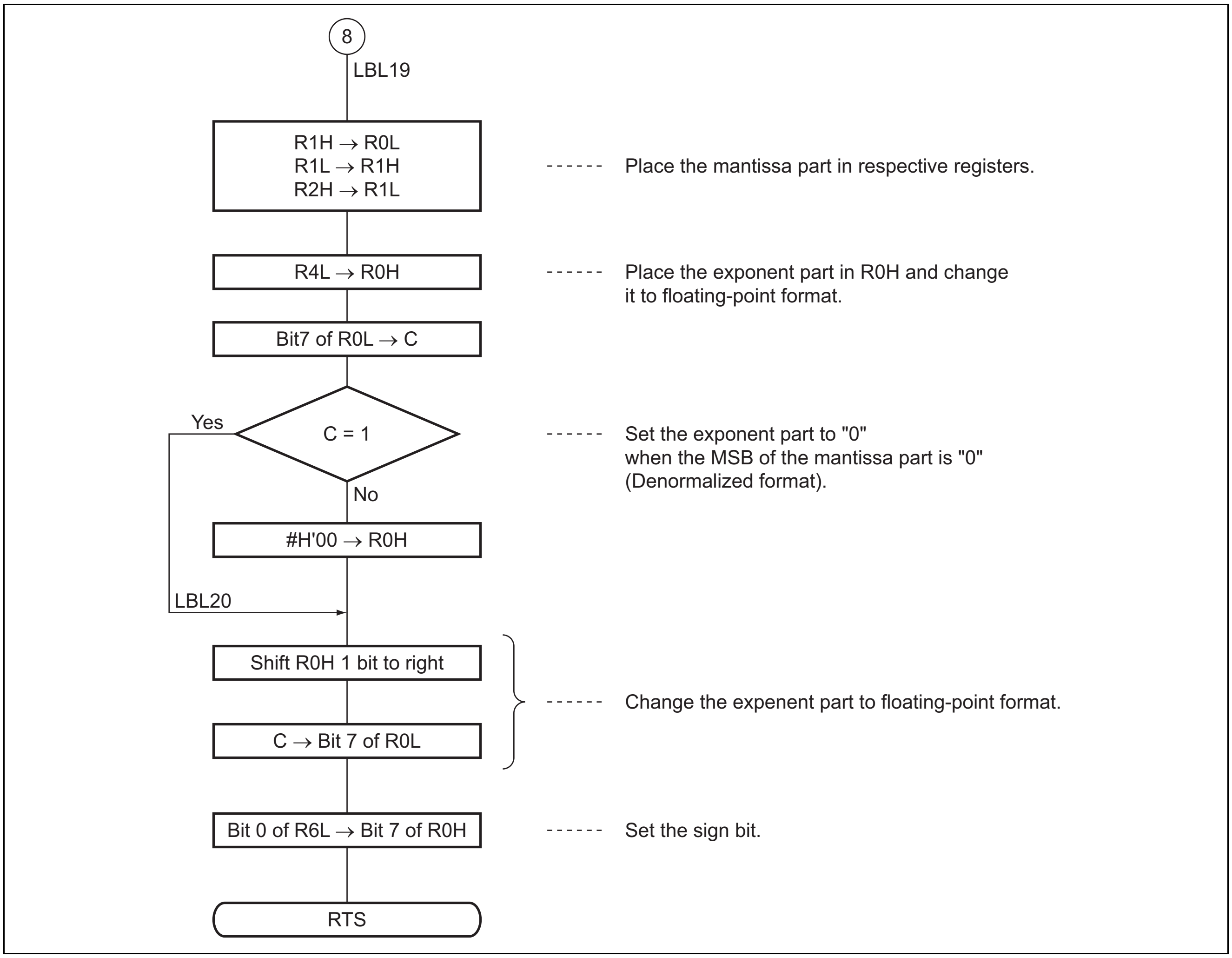

3
Bit 7 of R0L → C
Rotate R0H 1 bit to left
R0H → R4L
#H'00 → R4H
Place the exponent part of the multiplicand in R4.
Bit 7 of R2L → C
Rotate R2H 1 bit to left
R2H → R5
#H'00 → R4H
Place the exponent part of the multiplier in R5.
0 → Bit 7 of R0L
Place "0" in bit 7 or R0L.
R0H → R0H
R0H = 0
Yes
No
Branch when the exponent
part of the multiplicand is "0".
1 → Bit 7 of R0L
Place "1" in bit 7 of R0L.
R + #1 → R4
Increment R4.
0 → Bit 7 of R2L
Clear bit 7 of R2L.
R2H → R2H
R2H = 0
Yes
No
Branch when the exponent
part of the multiplicand is "0".
1 → Bit 7 of R2L
Place "1" in bit 7 of R2L.
R5 + #1 → R5
increment R5.
4

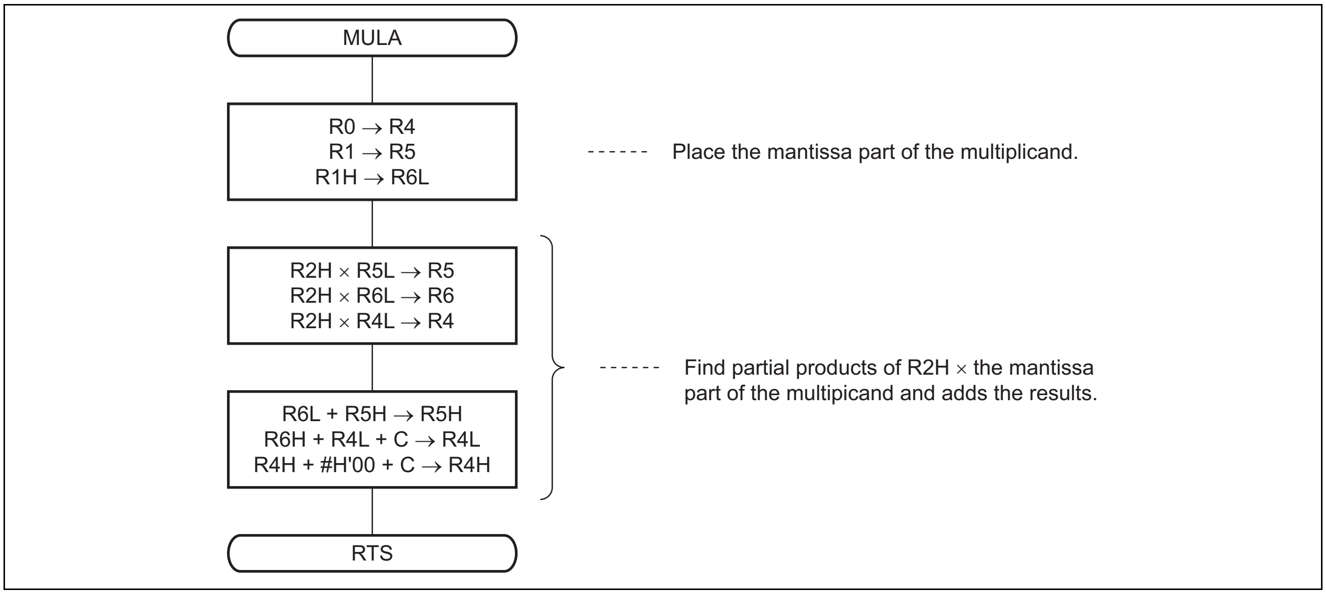

4
LBL11
R4 + R5 → R4
#H'FE → CCR
R4L - #H'7F - C → R4L
R4H - #0 - C → R4H
------ Add the exponent part of the multiplicand to that of the multiplier and substract H'7F from the result.
Push R4 and R6
------ Save the sign bit and exponent part.
(A)
R0 → R4
R1 → R5
R2L → R2H
------ Place the mantissa part of the multiplicand in R4 and R5, and the MSB of the mantissa part of the multiplier in R2H.
MULA
------ Call the subroutine MULA as a subroutine.
Push R4 and R5
------ Save the result of (R4 : R5) × R2L → (R4 : R5)
(B)
R3H → R2H
------ Place the upper second byte of the multiplier in R2H.
MULA
------ Call the subroutine MULA as a subroutine.
Push R4 and R5
------ Save the result of (R4 : R5) × R3H → (R4 : R5)
(C)
R3L → R2H
------ Place the lower 1 byte of the multiplier in R2H.
MULA
------ Call the subroutine MULA as a subroutine.
R4 → R2
R5 → R3
------ Place R4 and R5 in R2 and R3, respectively.
#H'0000 → R1
------ Clear R1 to "0".
Pop R5 and R4
------ Return the result of (B).
(D)
R3H + R5L → R3H
R2L + R5H + C → R2L
R2H + R4L + C → R2H
R1L + R4H + C → R1L
------ Add the result of (C) to that of (B).
Pop R5 and R4
------ Return the result of (A).
R2 + R5 → R2
R1L + R4L + C → R1L
R1H + R4H + C → R1H
------ Add the result of (D) and that of (A).
5


5
LBL3
Pop R6 and R4
Return the sign bit and exponent part.
R4 + #1 → R4
Increment R4.
R4 → R4
Branch when R4 ≤ 0.
R4 ≤ 0
Yes
7
No
LBL12
R4 - #1 → R4
Decrements R4.
R4 → R4
Branch when R4 = 0.
R4 = 0
Yes
No
Shift R3L 1 bit to left
Rotate R3H, R2L, R2H, R1L
and R1H 1 bit to left
Shift the mantissa part 1 bit to left.
C = 0
Yes
Branch when C = 0.
No
Rotate R1H, R1L, R2H
1 bit to right
Rotate the mantisa part 1 bit to right.
LBL13
R4 + #1 → R4
Increment R4.
6


6
LBL3
#H'00FF → R5
R4 < R5
Yes
No
Branch when R4 < H'00FF.
Bit 0 of R6L → C
C = 1
Yes
No
Check the sign bit.
#H'7F80 → R0
#H'0000 → R1
Place maximum posiyive values in R0 and R1.
RTS
LBL14
#H'FF80 → R0
#H'0000 → R1
Place maximum negative values in R0 and R1.
RTS
9
LBL15
R1 → R1
R1 = 0
8
Yes
No
R2H → R2H
R2H = 0
Yes
No
Place "0" as output when R2H = "0" and R1 = "0", otherwise, branch.
R1 → R1
RTS


7
LBL16
#H'001 → R5
#D'24 → R6H
Place "1" in R5.
Place D'24 in R6H as maximum loop count.
LBL17
Shift R1H 1 bit to right
Rotate R1L and R2H
1 bit to right
Shift the result of multiplication of the mantissa
part 1 bit to right.
R4 + #1 → R4
Increment the exponent part.
R6H - #1 → R6H
R6H = 0
Yes
No
Decrement R6H.
Branch when R6H = "0".
9
LBL15
Yes
R4 = R5
No
Branch when R4 is "1".
LBL18
#H'0000 → R0
R0 → R1
Place "0" as output.
RTS


8
LBL19
R1H → R0L
R1L → R1H
R2H → R1L
Place the mantissa part in respective registers.
R4L → R0H
Place the exponent part in R0H and change it to floating-point format.
Bit7 of R0L → C
C = 1
Yes
No
Set the exponent part to "0" when the MSB of the mantissa part is "0" (Denormalized format).
#H'00 → R0H
LBL20
Shift R0H 1 bit to right
C → Bit 7 of R0L
Change the expenent part to floating-point format.
Bit 0 of R6L → Bit 7 of R0H
Set the sign bit.
RTS


MULA
R0 → R4
R1 → R5
R1H → R6L
Place the mantissa part of the multiplicand.
R2H × R5L → R5
R2H × R6L → R6
R2H × R4L → R4
R6L + R5H → R5H
R6H + R4L + C → R4L
R4H + #H'00 + C → R4H
Find partial products of R2H × the mantissa
part of the multipicand and adds the results.
RTS

7. Program List

```
*** H8/300 ASSEMBLER VER 1.0B ** 08/18/92 10:22:23
PROGRAM NAME =
                                       ;****************************************************************
                                       ;*
                                       ;*     00 - NAME  :FLOATING POINT MULTIPLICATION (FMUL)
                                       ;*
                                       ;****************************************************************
                                       ;*
                                       ;*     ENTRY      :R0 (UPPER WORD OF MULTI PLICAND)
                                       ;*                  R1 (LOWER WORD OF MULTI PLICAND)
                                       ;*                  R2 (UPPER WORD OF MULTIPLIER)
                                       ;*                  R3 (LOWER WORD OF MULTIPLIER)
                                       ;*
                                       ;*     RETURNS    :R0 (UPPER WORD OF RESULT)
                                       ;*                  R1 (LOWER WORD OF RESULT)
                                       ;*
                                       ;****************************************************************
                                       ;
  FMUL_cod C  0000                            .SECTION              FMUL_code,CODE,ALIGN=2
                                              .EXPORT FMUL
                                       ;
  FMUL_cod C        00000000   FMUL    .EQU $                ;Entry point
  FMUL_cod C  0000  FE00                      MOV.B    #H'00,R6L     ;Clear R6L
  FMUL_cod C  0002  79057F80                  MOV.W    #H'7F80,R5    ;Set "H'7F80"
                                       ;
  FMUL_cod C  0006  7770                      BLD      #7,R0H        ;Set sign bit of multiplicand
  FMUL_cod C  0008  670E                      BST      #0,R6L        ; to bit 0 of R6L
  FMUL_cod C  000A  7270                      BCLR     #7,R0H        ;Bit clear bit 7 of R0H
                                       ;
  FMUL_cod C  000C  7772                      BLD      #7,R2H        ;
  FMUL_cod C  000E  750E                      BXOR     #0,R6L        ;Set sign bit of result
  FMUL_cod C  0010  670E                      BST      #0,R6L        ; to bit 0 of R6L
  FMUL_cod C  0012  7272                      BCLR     #7,R2H        ;Bit clear bit 7 of R2H
                                       ;
  FMUL_cod C  0014  0D11                      MOV.W    R1,R1
  FMUL_cod C  0016  4604                      BNE      LBL1
  FMUL_cod C  0018  0D00                      MOV.W    R0,R0
  FMUL_cod C  001A  4708                      BEQ      LBL2          ;Branch if R1=R0=0
  FMUL_cod C  001C             LBL1
  FMUL_cod C  001C  0D33                      MOV.W    R3,R3
  FMUL_cod C  001E  460C                      BNE      LBL3          ;Branch if not R3=0
  FMUL_cod C  0020  0D22                      MOV.W    R2,R2
  FMUL_cod C  0022  4608                      BNE      LBL3          ;Branch if not R2=0
                                       ;
  FMUL_cod C  0024             LBL2
  FMUL_cod C  0024  79000000                  MOV.W    #H'0000,R0    ;Set 0 to result
  FMUL_cod C  0028  0D01                      MOV.W    R0,R1
  FMUL_cod C  002A  5470                      RTS
                                       ;
```

```
FMUL_cod C  002C            LBL3
FMUL_cod C  002C  1D05              CMP.W     R0,R5
FMUL_cod C  002E  4304              BLS       LBL4          ;Branch if R0>=R5
FMUL_cod C  0030  1D25              CMP.W     R2,R5
FMUL_cod C  0032  4218              BHI       LBL7          ;Branch if R2>=R5
FMUL_cod C  0034            LBL4
FMUL_cod C  0034  770E              BLD       #0,R6L        ;Load sign bit
FMUL_cod C  0036  450A              BCS       LBL6          ;Branch if C = 1
FMUL_cod C  0038            LBL5
FMUL_cod C  0038  79007F80          MOV.W     #H'7F80,R0    ;Set #H'7F800000 to result
FMUL_cod C  003C  79010000          MOV.W     #H'0000,R1
FMUL_cod C  0040  5470              RTS
FMUL_cod C  0042            LBL6
FMUL_cod C  0042  7900FF80          MOV.W     #H'FF80,R0    ;Set #H'FF800000 to result
FMUL_cod C  0046  79010000          MOV.W     #H'0000,R1
FMUL_cod C  004A  5470              RTS
                            ;
FMUL_cod C  004C            LBL7
FMUL_cod C  004C  7778              BLD       #7,R0L        ;
FMUL_cod C  004E  1200              ROTXL     R0H           ;
FMUL_cod C  0050  0C0C              MOV.B     R0H,R4L       ;Set exponent of multiplicand to R4
FMUL_cod C  0052  F400              MOV.B     #H'00,R4H
                            ;
FMUL_cod C  0054  777A              BLD       #7,R2L
FMUL_cod C  0056  1202              ROTXL     R2H
FMUL_cod C  0058  0C2D              MOV.B     R2H,R5L       ;Set exponent of multiplier to R5
FMUL_cod C  005A  F500              MOV.B     #H'00,R5H
                            ;
FMUL_cod C  005C  7278              BCLR      #7,R0L        ;Clear bit 7 of R0L
FMUL_cod C  005E  0C00              MOV.B     R0H,R0H
FMUL_cod C  0060  4704              BEQ       LBL8          ;Branch if multiplicand is
                                                             denormalized
FMUL_cod C  0062  7078              BSET      #7,R0L        ;Set implicit MSB
FMUL_cod C  0064  4002              BRA       LBL9          ;Branch always
FMUL_cod C  0066            LBL8
FMUL_cod C  0066  0B04              ADDS.W #1,R4
                            ;
FMUL_cod C  0068            LBL9
FMUL_cod C  0068  727A              BCLR      #7,R2L        ;Clear bit 7 of R2L
FMUL_cod C  006A  0C22              MOV.B     R2H,R2H
FMUL_cod C  006C  4704              BEQ       LBL10         ;Branch if multiplier is
                                                             denormalized
FMUL_cod C  006E  707A              BSET      #7,R2L        ;Set implicit MSB
FMUL_cod C  0070  4002              BRA       LBL11         ;Branch always
FMUL_cod C  0072            LBL10
FMUL_cod C  0072  0B05              ADDS.W #1,R5
                            ;
FMUL_cod C  0074            LBL11
FMUL_cod C  0074  0954              ADD.W     R5,R4         ;addition exponents
FMUL_cod C  0076  06FE              ANDC      #H'FE,CCR     ;Clear C flag of CCR
FMUL_cod C  0078  BC7F              SUBX.B    #H'7F,R4L     ;R4L - #H'7F - C -> R4L
FMUL_cod C  007A  B400              SUBX.B    #H'00,R4H
```

```
                                 ;
FMUL_cod C  007C  6DF4                  PUSH     R4            ;Push R4
FMUL_cod C  007E  6DF6                  PUSH     R6            ;Push R6
                                 ;
FMUL_cod C  0080  0D04                  MOV.W    R0,R4         ;
FMUL_cod C  0082  0D15                  MOV.W    R1,R5
                                 ;
FMUL_cod C  0084  0CA2                  MOV.B    R2L,R2H
FMUL_cod C  0086  5E000000              JSR      @MULA         ;R2L * (R0L:R1) -> (R4:R5)
FMUL_cod C  008A  6DF4                  PUSH     R4            ;Push R4
FMUL_cod C  008C  6DF5                  PUSH     R5            ;Push R5
                                 ;
FMUL_cod C  008E  0C32                  MOV.B    R3H,R2H
FMUL_cod C  0090  5E000000              JSR      @MULA         ;R3L * (R0L:R1) -> (R4:R5)
FMUL_cod C  0094  6DF4                  PUSH     R4            ;Push R4
FMUL_cod C  0096  6DF5                  PUSH     R5            ;Push R5
                                 ;
FMUL_cod C  0098  0CB2                  MOV.B    R3L,R2H       ;
FMUL_cod C  009A  5E000000              JSR      @MULA         ;R3L * (R0L:R1) -> (R4:R5)
FMUL_cod C  009E  0D42                  MOV.W    R4,R2         ;Push R4
FMUL_cod C  00A0  0D53                  MOV.W    R5,R3         ;Push R5
                                 ;
FMUL_cod C  00A2  79010000              MOV.W    #H'0000,R1    ;Clear R1
FMUL_cod C  00A6  6D75                  POP      R5            ;Pop R5
FMUL_cod C  00A8  6D74                  POP      R4            ;Pop R4
                                 ;
FMUL_cod C  00AA  08D3                  ADD.B    R5L,R3H       ;R3H + R5L -> R3H
FMUL_cod C  00AC  0E5A                  ADDX.B   R5H,R2L       ;R2L + R5H + C -> R2L
FMUL_cod C  00AE  0EC2                  ADDX.B   R4L,R2H       ;R2H + R4L + C -> R2H
FMUL_cod C  00B0  0E49                  ADDX.B   R4H,R1L       ;R1L + R4H + C -> R1L
                                 ;
FMUL_cod C  00B2  6D75                  POP      R5            ;Pop R5
FMUL_cod C  00B4  6D74                  POP      R4            ;Pop R4
FMUL_cod C  00B6  0952                  ADD.W    R5,R2         ;R2 + R5 -> R2
FMUL_cod C  00B8  0EC9                  ADDX.B   R4L,R1L       ;R1L + R4L + C -> R1L
FMUL_cod C  00BA  0E41                  ADDX.B   R4H,R1H       ;R1H + R4H + C -> R1H
                                 ;
FMUL_cod C  00BC  6D76                  POP      R6            ;Pop R6
FMUL_cod C  00BE  6D74                  POP      R4            ;Pop R4
FMUL_cod C  00C0  0B04                  ADDS.W   #1,R4
FMUL_cod C  00C2  0D44                  MOV.W    R4,R4
                                 ;
FMUL_cod C  00C4  474A                  BEQ      LBL16         ;Branch if R4=0
FMUL_cod C  00C6  4B48                  BMI      LBL16         ;Branch if R4<0
FMUL_cod C  00C8                 LBL12
FMUL_cod C  00C8  1B04                  SUBS.W   #1,R4
FMUL_cod C  00CA  0D44                  MOV.W    R4,R4
FMUL_cod C  00CC  4714                  BEQ      LBL13         ;Branch if R4=0
FMUL_cod C  00CE  100B                  SHLL     R3L           ;Shift mantissa 1 bit left
FMUL_cod C  00D0  1203                  ROTXL    R3H
FMUL_cod C  00D2  120A                  ROTXL    R2L
FMUL_cod C  00D4  1202                  ROTXL    R2H
FMUL_cod C  00D6  1209                  ROTXL    R1L
FMUL_cod C  00D8  1201                  ROTXL    R1H
```

```
FMUL_cod C  00DA  44EC              BCC      LBL12         ;Branch if C = 0
FMUL_cod C  00DC  1301              ROTXR    R1H           ;Rotate mantissa 1 bit right
FMUL_cod C  00DE  1309              ROTXR    R1L
FMUL_cod C  00E0  1302              ROTXR    R2H
FMUL_cod C  00E2           LBL13
FMUL_cod C  00E2  0B04              ADDS.W   #1,R4
                           ;
FMUL_cod C  00E4  790500FF          MOV.W    #H'00FF,R5    ;
FMUL_cod C  00E8  1D45              CMP.W    R4,R5
FMUL_cod C  00EA  4418              BCC      LBL15         ;Branch if R5>R4
FMUL_cod C  00EC  770E              BLD      #0,R6L        ;Load sign bit
FMUL_cod C  00EE  450A              BCS      LBL14         ;Branch if C = 1
FMUL_cod C  00F0  79007F80          MOV.W    #H'7F80,R0    ;Set H'7F800000 to result
FMUL_cod C  00F4  79010000          MOV.W    #H'0000,R1
FMUL_cod C  00F8  5470              RTS
                           ;
FMUL_cod C  00FA           LBL14
FMUL_cod C  00FA  7900FF80          MOV.W    #H'FF80,R0    ;Set H'FF800000 to product
FMUL_cod C  00FE  79010000          MOV.W    #H'0000,R1
FMUL_cod C  0102  5470              RTS
                           ;
FMUL_cod C  0104           LBL15
FMUL_cod C  0104  0D11              MOV.W    R1,R1
FMUL_cod C  0106  4628              BNE      LBL19         ;Branch if not R1=0
FMUL_cod C  0108  0C22              MOV.B    R2H,R2H
FMUL_cod C  010A  4624              BNE      LBL19         ;Branch if not R2H=0
FMUL_cod C  010C  0D10              MOV.W    R1,R0
FMUL_cod C  010E  5470              RTS
                           ;
FMUL_cod C  0110           LBL16
FMUL_cod C  0110  79050001          MOV.W    #H'0001,R5    ;Set #H'0001 to R5
FMUL_cod C  0114  F618              MOV.B    #D'24,R6H     ;Se bit counter
FMUL_cod C  0116           LBL17
FMUL_cod C  0116  1101              SHLR     R1H           ;Shift mantissa 1 bit right
FMUL_cod C  0118  1309              ROTXR    R1L
FMUL_cod C  011A  1302              ROTXR    R2H
FMUL_cod C  011C  0B04              ADDS.W   #1,R4         ;Increment exponent
FMUL_cod C  011E  1A06              DEC.B    R6H           ;Decrement bit counter
FMUL_cod C  0120  4706              BEQ      LBL18         ;Branch if Z=1
FMUL_cod C  0122  1D54              CMP.W    R5,R4
FMUL_cod C  0124  47DE              BEQ      LBL15         ;Branch if R5=R4
FMUL_cod C  0126  40EE              BRA      LBL17         ;Branch always
FMUL_cod C  0128           LBL18
FMUL_cod C  0128  79000000          MOV.W    #H'0000,R0    ;Clear result
FMUL_cod C  012C  0D01              MOV.W    R0,R1
FMUL_cod C  012E  5470              RTS
                           ;
FMUL_cod C  0130           LBL19
FMUL_cod C  0130  0C18              MOV.B    R1H,R0L
FMUL_cod C  0132  0C91              MOV.B    R1L,R1H
FMUL_cod C  0134  0C29              MOV.B    R2H,R1L
```

```
                                  ;
      FMUL_cod C  0136  0CC0              MOV.B    R4L,R0H
      FMUL_cod C  0138  7778              BLD      #7,R0L
      FMUL_cod C  013A  4502              BCS      LBL20        ;Branch if C = 1
      FMUL_cod C  013C  F000              MOV.B    #H'00,R0H
      FMUL_cod C  013E                LBL20                     ;Change floating point format
      FMUL_cod C  013E  1100              SHLR     R0H
      FMUL_cod C  0140  6778              BST      #7,R0L
      FMUL_cod C  0142  770E              BLD      #0,R6L
      FMUL_cod C  0144  6770              BST      #7,R0H
      FMUL_cod C  0146  5470              RTS
                                  ;
                                  ;---------------------------------------------------------------
                                  ;
      FMUL_cod C  0148                MULA                      ;R2H * (R0L:R1) -> (R4:R5)
      FMUL_cod C  0148  0D04              MOV.W    R0,R4        ;R0 -> R4
      FMUL_cod C  014A  0D15              MOV.W    R1,R5        ;R1 -> R5
      FMUL_cod C  014C  0C1E              MOV.B    R1H,R6L      ;R1H -> R6L
                                  ;
      FMUL_cod C  014E  5025              MULXU    R2H,R5       ;R2H * R5L -> R5
      FMUL_cod C  0150  5026              MULXU    R2H,R6       ;R2H * R6L -> R6
      FMUL_cod C  0152  5024              MULXU    R2H,R4       ;R2H * R4L -> R4
                                  ;
      FMUL_cod C  0154  08E5              ADD.B    R6L,R5H      ;R5H + R6L -> R5H
      FMUL_cod C  0156  0E6C              ADDX.B   R6H,R4L      ;R4L + R6H + C -> R4L
      FMUL_cod C  0158  9400              ADDX.B   #H'00,R4H    ;R4H + #H'00 + C -> R4H
*** H8/300 ASSEMBLER VER 1.0B ** 08/18/92 10:22:23 PAGE
5
PROGRAM NAME =
      FMUL_cod C  015A  5470               RTS
                                  ;
                                          .END
*****TOTAL ERRORS 0
*****TOTAL WARNINGS 0
```

About Single-Precision Floating-Point Numbers <Reference>

Single-Precision Floating-Point Formats:

1. Internal representation of single-precision floating-point numbers

   In this Application Note, the following formats are applied to single-precision floating-point numbers depending on their values (R = real number):

   A. Internal representation for R = 0

31	30	29	......	2	1	0
0	0	0	......	0	0	0

All of the 32 bits are 0's.

   B. Normalized format

31	30 ... 23	22 ...	0
S	α	β	0

α is an exponent whose field is 8 bits long. β is a mantissa whose field is 23 bits long. The value of R can be represented by the following equation (on conditions that $1 \leq \alpha \leq 254$) :

$$R = 2^{S} \times 2^{\alpha-127} \times (1 + 2^{-1} \times \beta_{22} + 2^{-2} \times \beta_{21} + \ldots\ldots + 2^{-23} \times \beta_{0})$$

where βi is the value of the i-th bit ($0 \leq i \leq 22$) and S is the sign bit.

   C. Denormalized format

31	30 ... 23	22 ... 0
S	0 0 0 0 0 0 0 0	β

where β is a mantissa whose field is 23 bits long. This format is used to represent a real number too small to be represented in the normal format. In this format, R can be represented by the following equation:

$$R = 2^{S} \times 2^{-126} \times (2^{-1} \times \beta_{22} + 2^{-2} \times \beta_{21} + \ldots\ldots + 2^{-23} \times \beta_{0})$$

   D. Infinity

31	30 ... 23	22 ... 0
S	1 1 1 1 1 1 1 1	β

where β is a mantissa whose field is 23 bits long. In this Application Note, however, the following rules apply if all exponents are 1's;

Positive infinity when S = 0

$R = +\infty$

Negative infinity when S = 1

$R = -\infty$

2. Example of internal representation

If S = B'0 (binary)

α = B'10000011 (binary)

β = B'1011100……0 (binary)

Then the corresponding real number is as follows:

$R = 2^{0} \times 2^{131-127} \times (1 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5})$

$= 16 + 8 + 2 + 1 + 0.5 = 27.5$

A. Maximum and minimum values

The maximum value (R_{MAX}) and minimum value (R_{MIN}), in terms of the absolute value, are as follows:

$RMAX = 2^{254 - 127} \times (1 + 2^{-1} + 2^{-2} + 2^{-3} \ldots\ldots + 2^{-23})$

$= 3.37 \times 10^{38}$

$RMIN = 2^{-126} \times 2^{-23} = 2^{-140} = 1.40 \times 10^{-45}$

The absolute values within the above range can be represented.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.18.03	—	First edition issued
2.00	Nov.30.06	All pages	Content correction