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

H8/300H Tiny Series

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

Introduction

Multiplies single-precision floating-point numbers set in general registers and stores the result in general registers.

Target Device

H8/300H Tiny Series

Contents

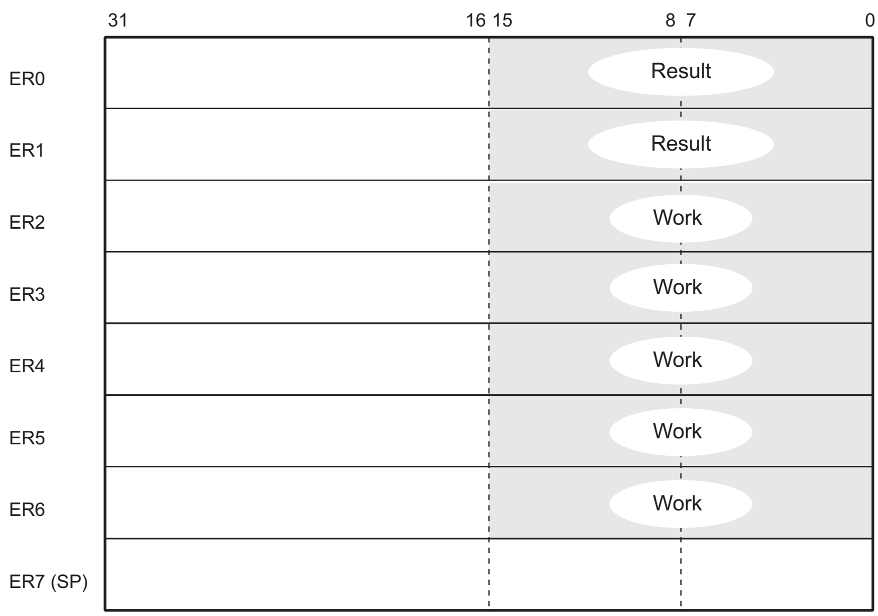
1. Function

1. Multiplies single-precision floating-point numbers set in general registers and stores the result in general registers.
2. The arguments are all in the single-precision floating-point data format.

2. Arguments

Description		Storage Location	Data Length (Bytes)
Input	Multiplicand	R0, R1	4
	Multiplier	R2, R3	4
Output	Result	R0, R1	4

3. Changes to Internal Registers and Flags



I	UI	H	U	N	Z	V	C
—	—	↕	—	↕	↕	↕	↕

—: No change
↕ : Varies
0 : Fixed to 0
1 : Fixed to 1

4. Programming Specifications

Program memory (bytes)	348
Data memory (bytes)	0
Stack (bytes)	16
Number of cycles	1078
Re-entrant	Yes
Relocatalbe	Yes
Interrupts during execution	Yes

5. Notes

The number of cycles in the programming specifications is the value for execution of the example in figure 1. For details on the floating-point data format, refer to Reference: Description of Single-Precision Floating-Point Formats.

6. Descriptions

6.1 Descriptions of Functions

1. The arguments are as follows.
   1) Set the input arguments as follows.
      R0: higher-order two bytes of the multiplicand
      R1: lower-order two bytes of the multiplicand
      R2: higher-order two bytes of the multiplier
      R3: lower-order two bytes of the multiplier
   2) The FMUL subroutine sets the following output arguments.
      R0: higher-order two bytes of the multiplication result
      R1: lower-order two bytes of the multiplication result

2. The following figure illustrates the execution of the FMUL subroutine. When the input arguments are set as shown below, the subroutine places the result of multiplication in R0 and R1.

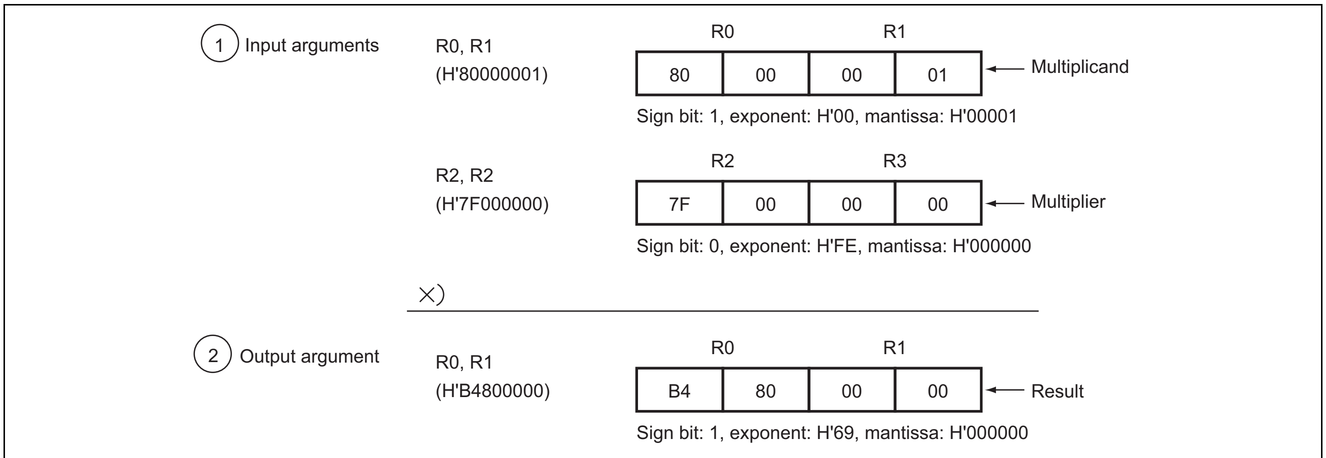


Figure 1 Example of FMUL Execution

6.2 Usage Notes

1. The maximum and minimum values handled by the FMUL subroutine are given below.
   Maximum positive value: H'7F80000
   Minimum positive value: H'00000001
   Maximum negative value: H'80000001
   Minimum negative value: H'FF800000
2. Positive single-precision floating-point numbers from H'7F800001 to H'7FFFFFFF are regarded as having the maximum value, H'7F800000. Negative single-precision floating-point numbers from H'FF800000 to H'FFFFFFF are regarded as having the minimum value, H'FF800000.
3. The maximum value is handled as infinity (∞), that is, the result of multiplying other numbers by the maximum value is the maximum value. For example, in multiplication by 100, ∞ × 100 = ∞ and ∞ × (–100) = –∞ (see table 1).

Table 1 Examples of Results when Maximum Values are Specified as Arguments

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

4. H'80000000 is handled as H'00000000 (zero).
5. The multiplicand and multiplier stored in the general registers are lost through execution of FMUL. When you will still require the input arguments, save them elsewhere in memory beforehand.

6.3 Description of Data Memory

No data memory is used by the FMUL subroutine.

6.4 Example of Usage

After setting the multiplicand and multiplier in the general registers, call the FADD subroutine.

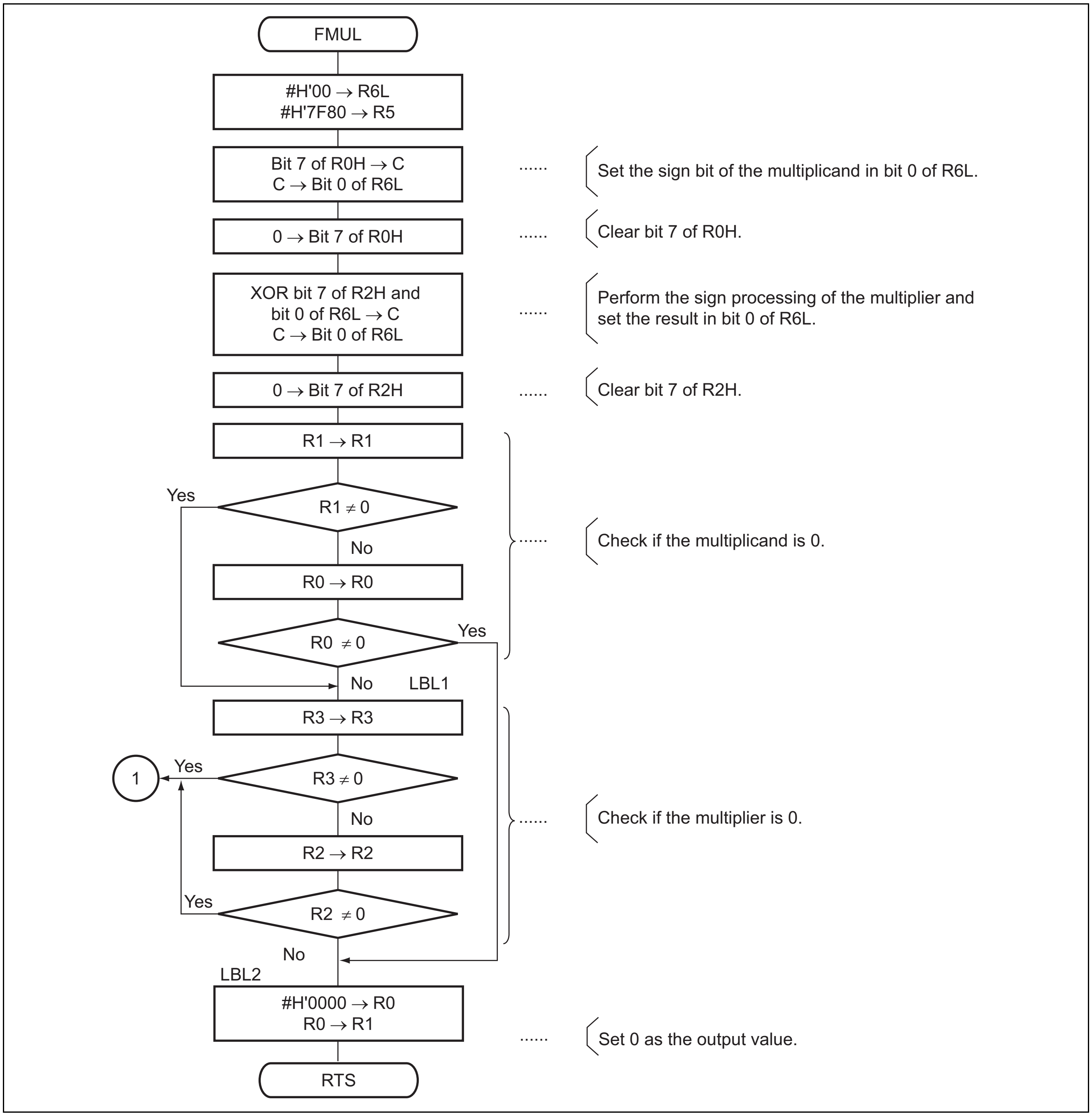
```
WORK1    . RES. W 2          ········· Reservation of  the data memory area for setting of the multiplicand by the user program.
WORK2    . RES. W 2          ········· Reservation of  the data memory area for setting of the multiplier by the user program.
WORK3    . RES. W 2          ········· Reservation of the data memory area where the product of multiplication will be stored by the user
             .                         program.
             .
             .
         MOV. W @WORK1, R0   ········· Sets the multiplicand specified by the user program as an input argument.
         MOV. W @WORK1+2, R1
         MOV. W @WORK2, R2   ········· Sets the multiplier specified by the user program as an input argument.
         MOV. W @WORK2+2, R3
         JSR    @FMUL        ········· Subroutine call of FMUL
         MOV. W R0, @WORK3   ········· Transfers the product set as the output argument to the data memory area of the user program.
         MOV. W R0, @WORK3+2
```

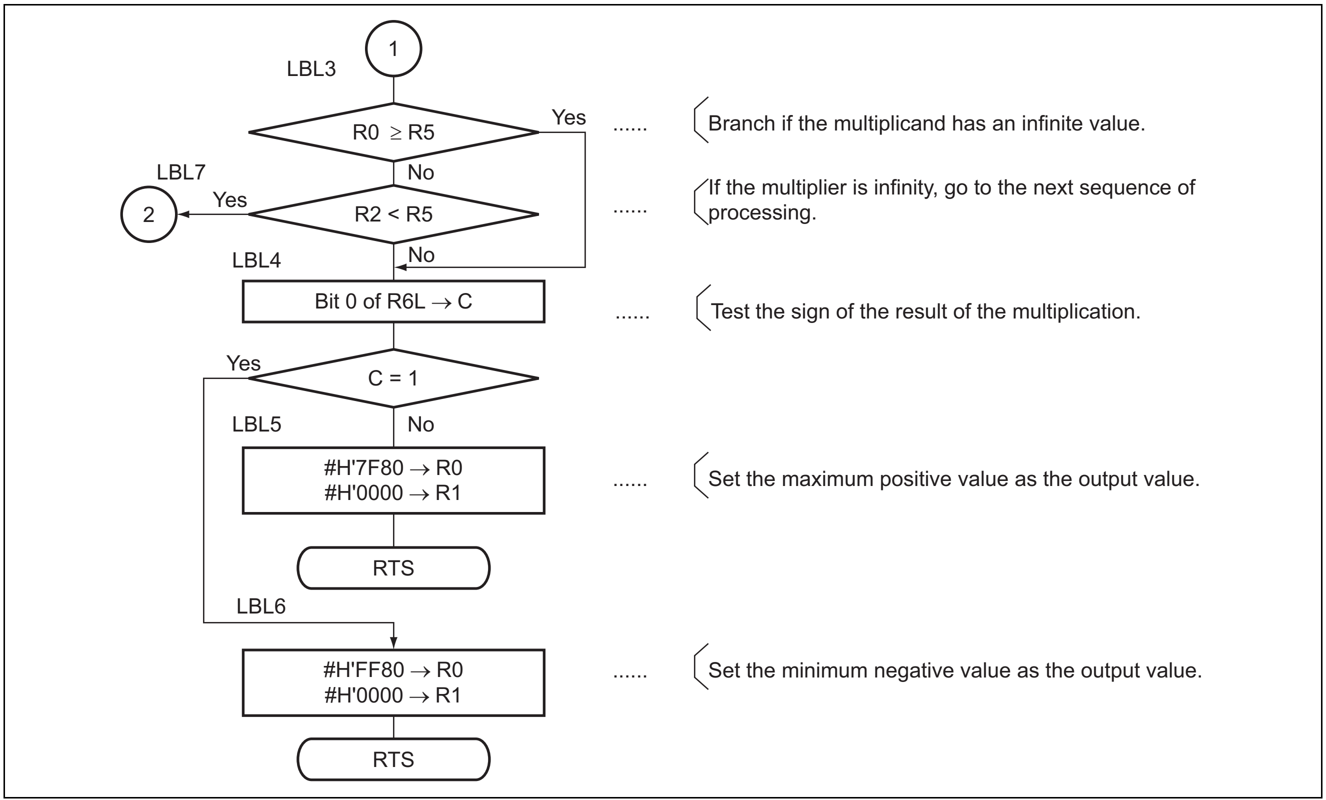
6.5 Principles of Operation

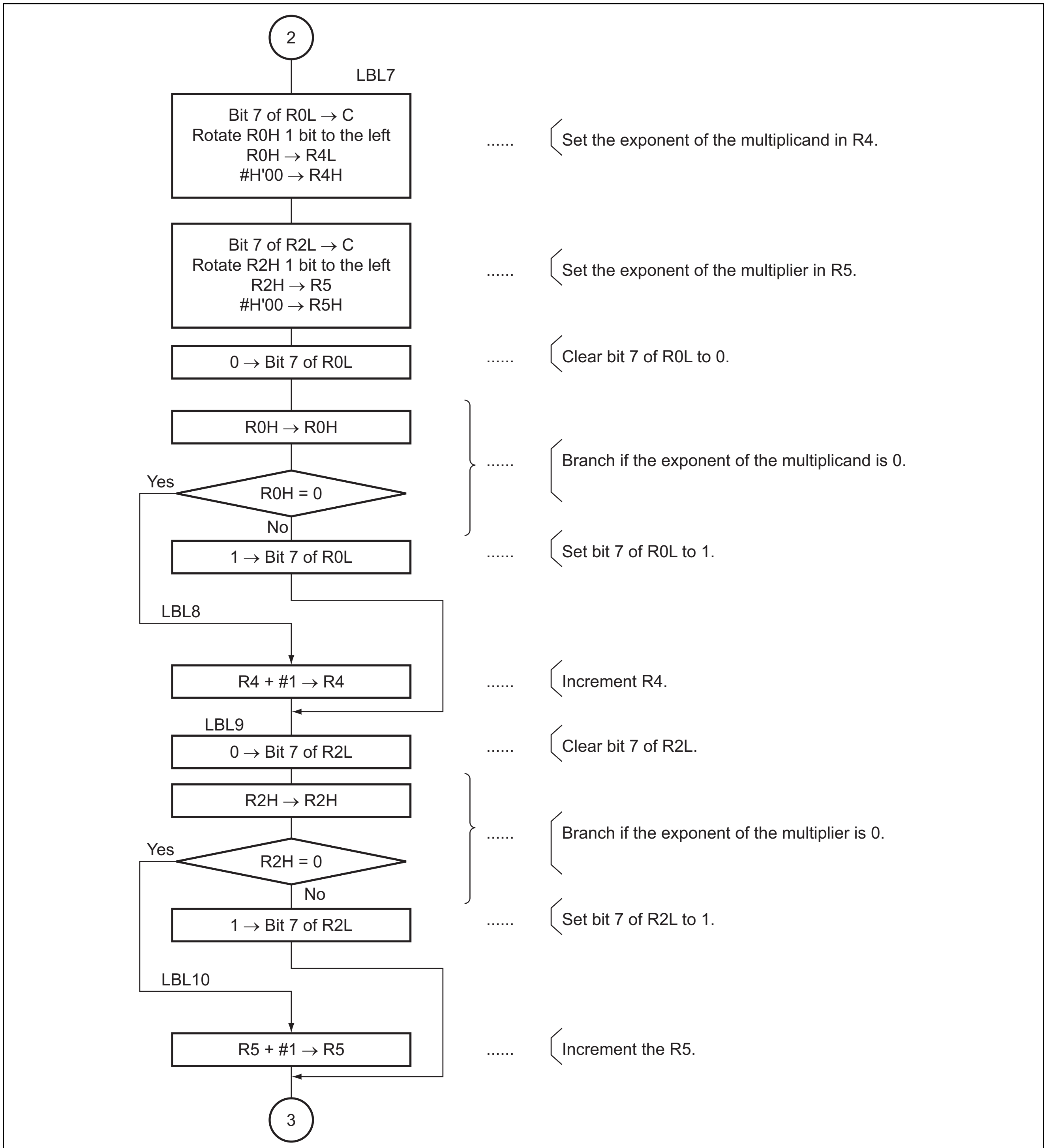
Multiplication of the single-precision floating-point numbers is according to the following sequence.

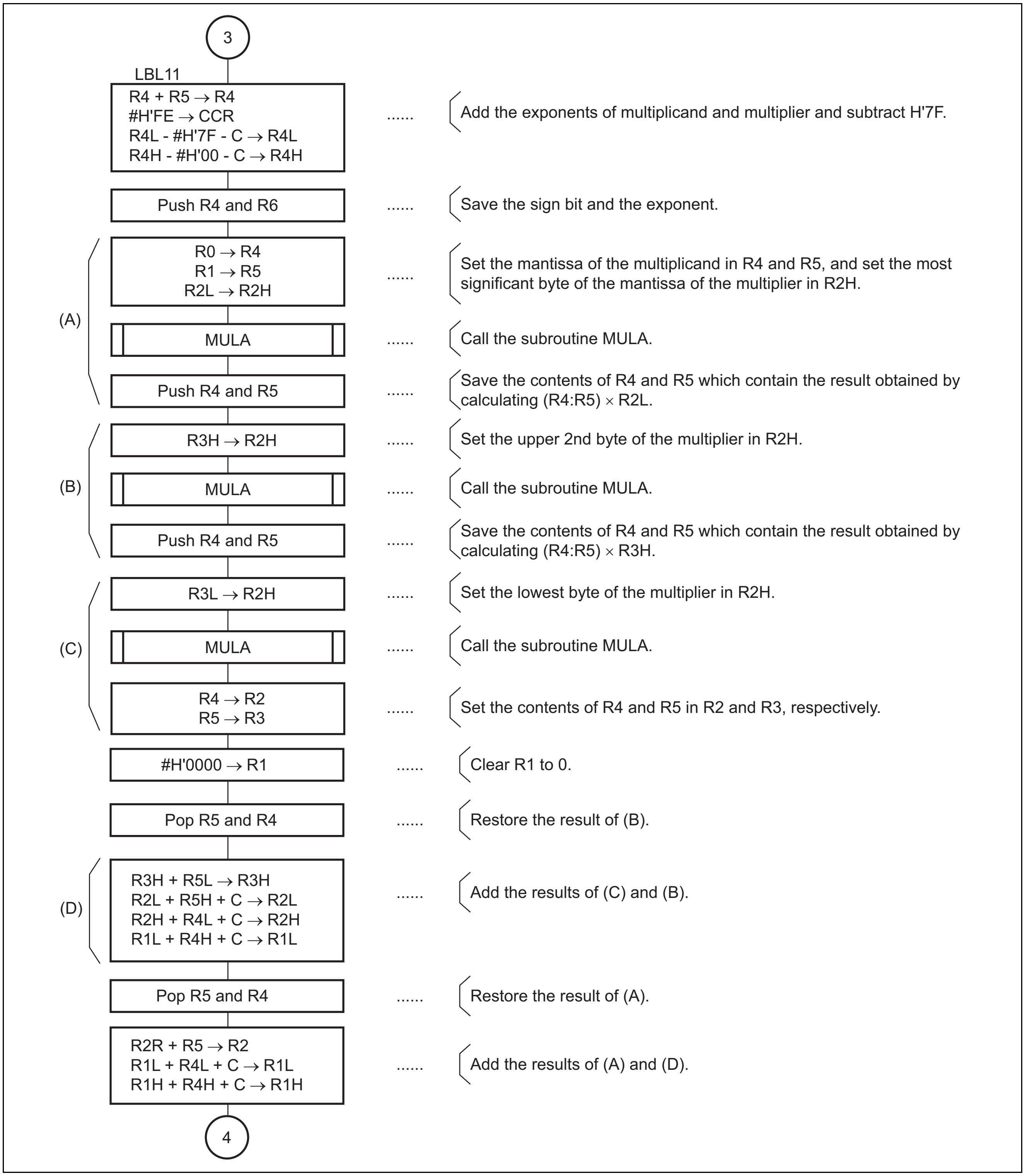
1. The multiplicand and multiplier are checked for zero values.
   If one or both holds a zero, H'00000000 is output.
2. The multiplicand and multiplier are checked for infinite (+∞ or –∞) values.
   If one or both of them is infinite (+∞ or –∞), the result is as given in table 6.1.
3. The exponents of the multiplicand and multiplier are matched.
   Let R1 be the multiplicand (sign bit = S1, exponent = α1, mantissa = β1) and R2 the multiplier (sign bit = S2, exponent = α2, mantissa = β2); R1 and R2 are then expressed as follows.
   $R1 = (-1)^{S1} \times 2^{\alpha1-127} \times \beta1$
   $R2 = (-1)^{S2} \times 2^{\alpha2-127} \times \beta2$
   Multiplication is as follows.
   $R1 \times R2 = (-1)^{S1+S2} \times 2^{\alpha1+\alpha2-127-127} \times \beta1 \times \beta2$
   In the floating-point data format, H'7F (D'127) is added to the actual exponent, so the equation will be as follows.
   $R1 \times R2 = (-1)^{S1+S2} \times 2^{\alpha1+\alpha2-127} \times \beta1 \times \beta2$
   Multiplication according to this equation is carried out in the following sequence.
   1) The exponents are added to each other.
      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 α1 and α2, and H'7F (D'127) is added to the exponent of the result. The result may thus be expressed as follows.
      $(\alpha1 - H'7F) + (\alpha2 - H'7F) + H'7F = \alpha1 + \alpha2 - H'7F$
      One is added to the exponent of a number in denormalized format before the calculation.
   2) The mantissas are multiplied by each other.
      The implicit MSB is included in the multiplication.
      For a number in the denormalized format, the implicit MSB of the mantissa is taken to be zero.
   3) The result of multiplication is corrected to produce a number in the floating-point data format.

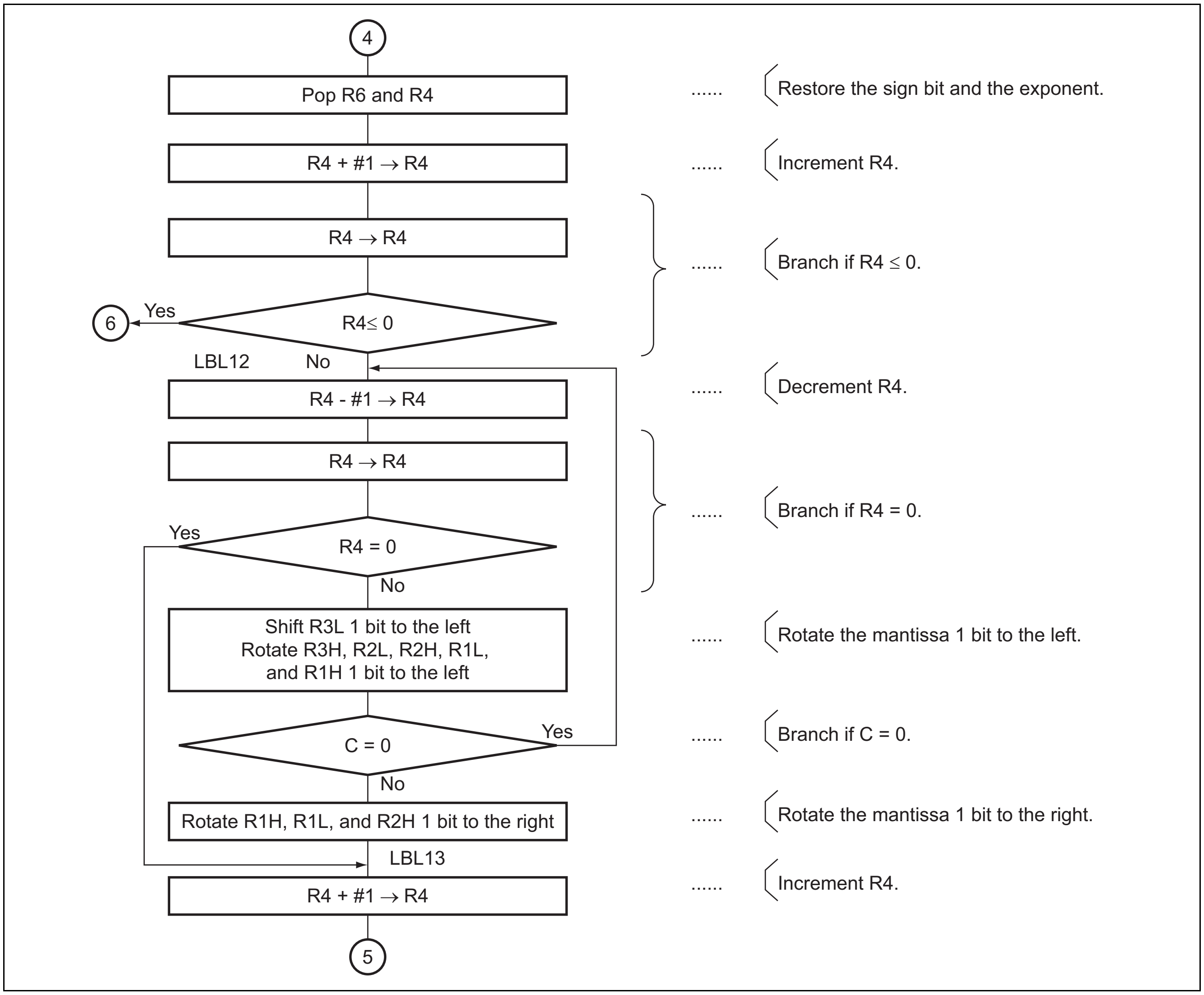
7. Flowchart

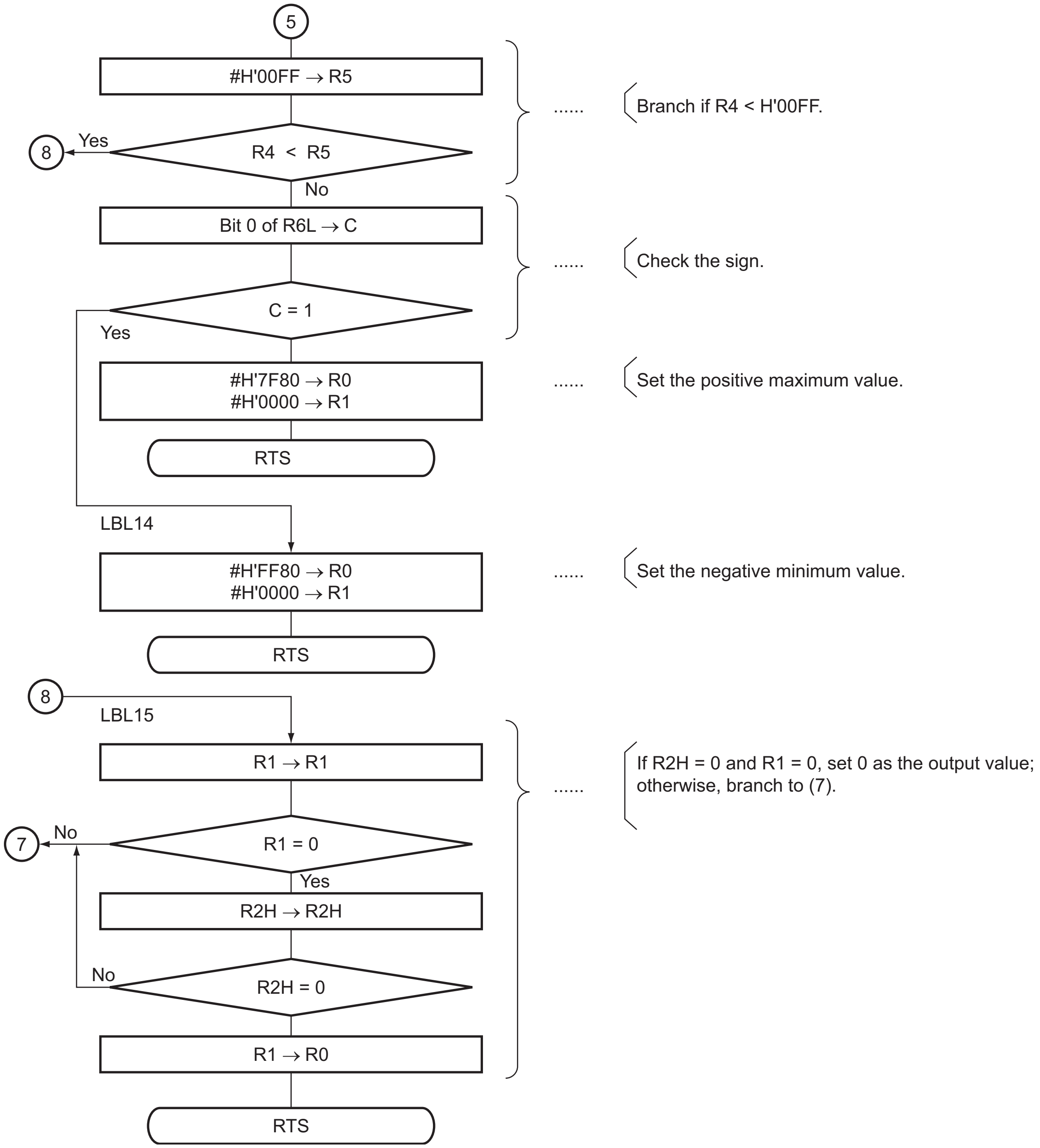

FMUL
#H'00 → R6L
#H'7F80 → R5
Bit 7 of R0H → C
C → Bit 0 of R6L
Set the sign bit of the multiplicand in bit 0 of R6L.
0 → Bit 7 of R0H
Clear bit 7 of R0H.
XOR bit 7 of R2H and
bit 0 of R6L → C
C → Bit 0 of R6L
Perform the sign processing of the multiplier and
set the result in bit 0 of R6L.
0 → Bit 7 of R2H
Clear bit 7 of R2H.
R1 → R1
Yes
R1 ≠ 0
No
R0 → R0
Yes
R0 ≠ 0
No
LBL1
Check if the multiplicand is 0.
R3 → R3
1
Yes
R3 ≠ 0
No
R2 → R2
Yes
R2 ≠ 0
No
Check if the multiplier is 0.
LBL2
#H'0000 → R0
R0 → R1
Set 0 as the output value.
RTS

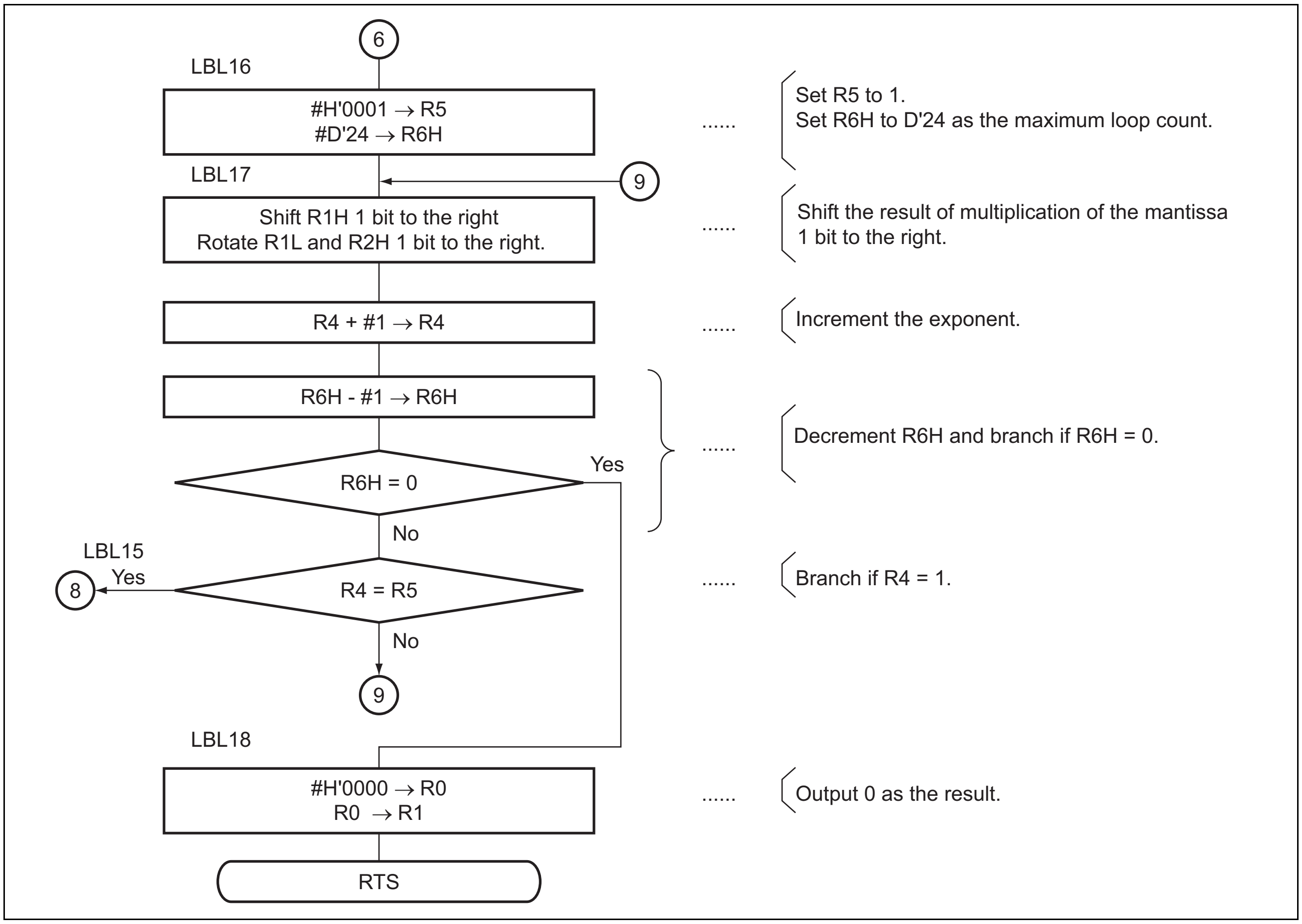

1
LBL3
R0 ≥ R5
Yes
No
LBL7
2
Yes
R2 < R5
No
LBL4
Bit 0 of R6L → C
Yes
C = 1
LBL5
No
#H'7F80 → R0
#H'0000 → R1
RTS
LBL6
#H'FF80 → R0
#H'0000 → R1
RTS
Branch if the multiplicand has an infinite value.
If the multiplier is infinity, go to the next sequence of processing.
Test the sign of the result of the multiplication.
Set the maximum positive value as the output value.
Set the minimum negative value as the output value.

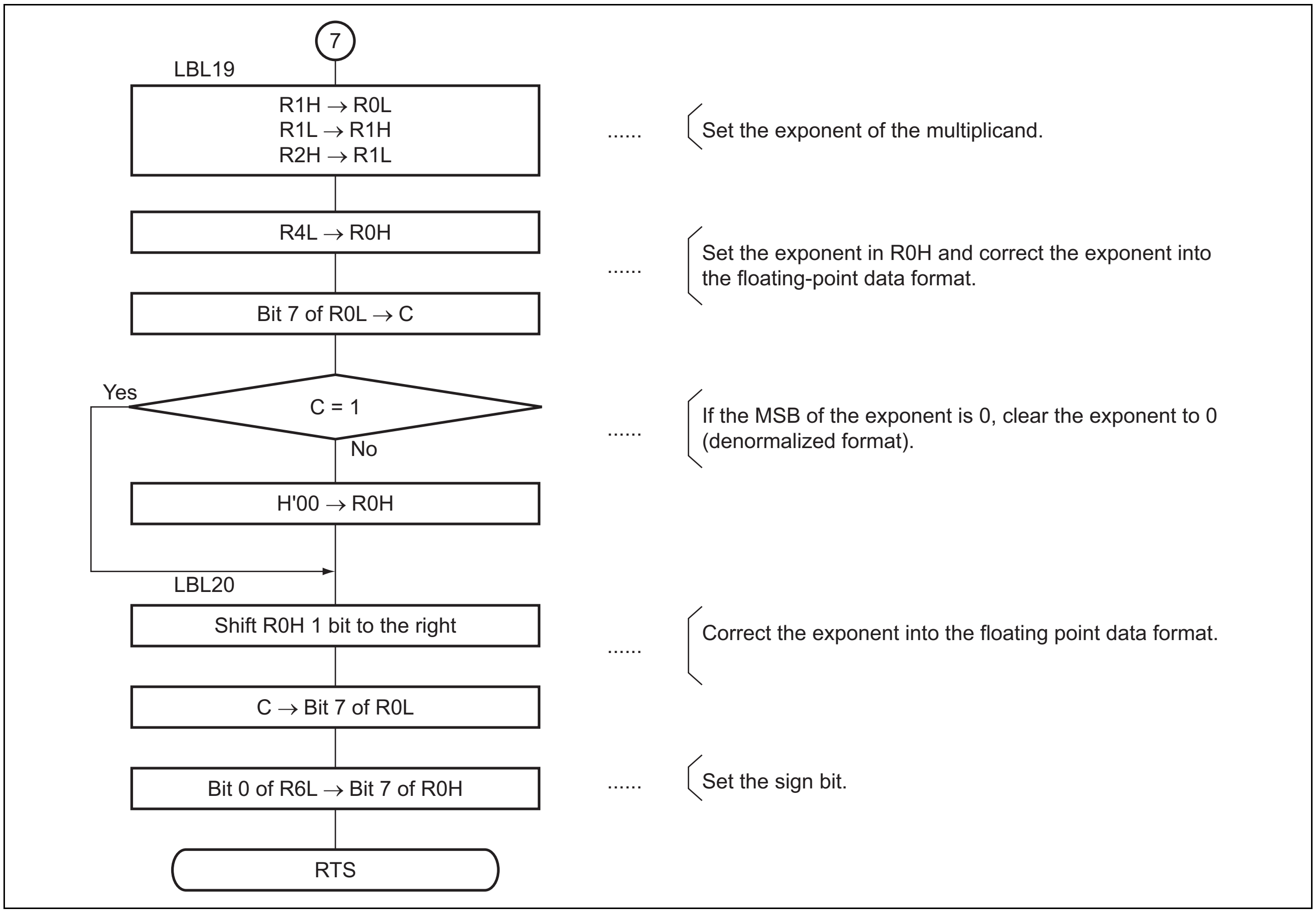

2
LBL7
Bit 7 of R0L → C
Rotate R0H 1 bit to the left
R0H → R4L
#H'00 → R4H
...... Set the exponent of the multiplicand in R4.
Bit 7 of R2L → C
Rotate R2H 1 bit to the left
R2H → R5
#H'00 → R5H
...... Set the exponent of the multiplier in R5.
0 → Bit 7 of R0L
...... Clear bit 7 of R0L to 0.
R0H → R0H
R0H = 0
Yes
No
...... Branch if the exponent of the multiplicand is 0.
1 → Bit 7 of R0L
...... Set bit 7 of R0L to 1.
LBL8
R4 + #1 → R4
...... Increment R4.
LBL9
0 → Bit 7 of R2L
...... Clear bit 7 of R2L.
R2H → R2H
R2H = 0
Yes
No
...... Branch if the exponent of the multiplier is 0.
1 → Bit 7 of R2L
...... Set bit 7 of R2L to 1.
LBL10
R5 + #1 → R5
...... Increment the R5.
3

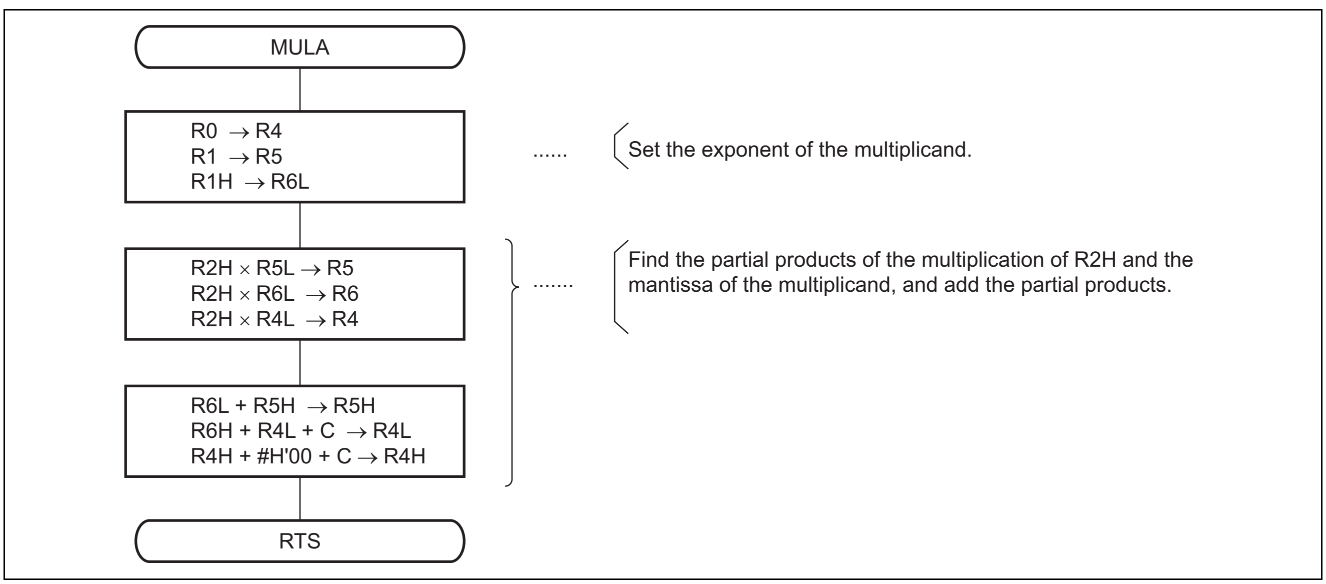

3
LBL11
R4 + R5 → R4
#H'FE → CCR
R4L - #H'7F - C → R4L
R4H - #H'00 - C → R4H
Add the exponents of multiplicand and multiplier and subtract H'7F.
Push R4 and R6
Save the sign bit and the exponent.
(A)
R0 → R4
R1 → R5
R2L → R2H
Set the mantissa of the multiplicand in R4 and R5, and set the most significant byte of the mantissa of the multiplier in R2H.
MULA
Call the subroutine MULA.
Push R4 and R5
Save the contents of R4 and R5 which contain the result obtained by calculating (R4:R5) × R2L.
(B)
R3H → R2H
Set the upper 2nd byte of the multiplier in R2H.
MULA
Call the subroutine MULA.
Push R4 and R5
Save the contents of R4 and R5 which contain the result obtained by calculating (R4:R5) × R3H.
(C)
R3L → R2H
Set the lowest byte of the multiplier in R2H.
MULA
Call the subroutine MULA.
R4 → R2
R5 → R3
Set the contents of R4 and R5 in R2 and R3, respectively.
#H'0000 → R1
Clear R1 to 0.
Pop R5 and R4
Restore the result of (B).
(D)
R3H + R5L → R3H
R2L + R5H + C → R2L
R2H + R4L + C → R2H
R1L + R4H + C → R1L
Add the results of (C) and (B).
Pop R5 and R4
Restore the result of (A).
R2R + R5 → R2
R1L + R4L + C → R1L
R1H + R4H + C → R1H
Add the results of (A) and (D).
4


4
Pop R6 and R4
R4 + #1 → R4
R4 → R4
R4≤ 0
Yes
6
LBL12
No
R4 - #1 → R4
R4 → R4
R4 = 0
Yes
No
Shift R3L 1 bit to the left
Rotate R3H, R2L, R2H, R1L, and R1H 1 bit to the left
C = 0
Yes
No
Rotate R1H, R1L, and R2H 1 bit to the right
LBL13
R4 + #1 → R4
5
...... Restore the sign bit and the exponent.
...... Increment R4.
...... Branch if R4 ≤ 0.
...... Decrement R4.
...... Branch if R4 = 0.
...... Rotate the mantissa 1 bit to the left.
...... Branch if C = 0.
...... Rotate the mantissa 1 bit to the right.
...... Increment R4.


5
#H'00FF → R5
8
Yes
R4 < R5
No
Branch if R4 < H'00FF.
Bit 0 of R6L → C
C = 1
Yes
Check the sign.
#H'7F80 → R0
#H'0000 → R1
Set the positive maximum value.
RTS
LBL14
#H'FF80 → R0
#H'0000 → R1
Set the negative minimum value.
RTS
8
LBL15
R1 → R1
7
No
R1 = 0
Yes
R2H → R2H
No
R2H = 0
R1 → R0
If R2H = 0 and R1 = 0, set 0 as the output value; otherwise, branch to (7).
RTS


6
LBL16
#H'0001 → R5
#D'24 → R6H
...... Set R5 to 1.
Set R6H to D'24 as the maximum loop count.
LBL17
9
Shift R1H 1 bit to the right
Rotate R1L and R2H 1 bit to the right.
...... Shift the result of multiplication of the mantissa 1 bit to the right.
R4 + #1 → R4
...... Increment the exponent.
R6H - #1 → R6H
R6H = 0
Yes
No
...... Decrement R6H and branch if R6H = 0.
LBL15
Yes
8
R4 = R5
No
9
...... Branch if R4 = 1.
LBL18
#H'0000 → R0
R0 → R1
...... Output 0 as the result.
RTS


7
LBL19
R1H → R0L
R1L → R1H
R2H → R1L
Set the exponent of the multiplicand.
R4L → R0H
Bit 7 of R0L → C
Set the exponent in R0H and correct the exponent into the floating-point data format.
Yes
C = 1
No
H'00 → R0H
If the MSB of the exponent is 0, clear the exponent to 0 (denormalized format).
LBL20
Shift R0H 1 bit to the right
C → Bit 7 of R0L
Correct the exponent into the floating point data format.
Bit 0 of R6L → Bit 7 of R0H
Set the sign bit.
RTS


MULA
R0 → R4
R1 → R5
R1H → R6L
Set the exponent 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 the partial products of the multiplication of R2H and the mantissa of the multiplicand, and add the partial products.
RTS

8. Program Listing

```
                    ;************************************************************************
                    ;*                                                                      *
                    ;*          NAME : FLOATING POINT MULTIPLICATION (FMUL)                 *
                    ;*                                                                      *
                    ;************************************************************************
                    ;*                                                                      *
                    ;*        ENTRY:   R0     (HIGHER WORD OF MULTIPLICAND)                 *
                    ;*                 R1     (LOWER WORD OF MULTIPLICAND)                  *
                    ;*                 R2     (HIGHER WORD OF MULTIPLIER)                   *
                    ;*                 R3     (LOWER WORD OF MULTIPLIER)                    *
                    ;*                                                                      *
                    ;*       RETURNS:  R0     (HIGHER WORD OF RESULT)                       *
                    ;*                 R1     (LOWER WORD OF RESULT)                        *
                    ;*                                                                      *
                    ;************************************************************************
                    ;
                            .CPU        300HN
0000                        .SECTION    FMUL_code,CODE,ALIGN=2
                            .EXPORT     FMUL
                    ;
     00000000       FMUL    .EQU        $           ;Entry point
0000 FE00                   MOV.B       #H'00,R6L   ;Clear R6L
0002 79057F80               MOV.W       #H'7F80,R5  ;Set H'7F80
                    ;
0006 7770                   BLD         #7,R0H      ;Set sign bit of multiplicand
0008 670E                   BST         #0,R6L      ;  to bit 0 of R6L
000A 7270                   BCLR        #7,R0H      ;Bit clear bit 7 of R0H
                    ;
000C 7772                   BLD         #7,R2H      ;
000E 750E                   BXOR        #0,R6L      ;Set sign bit of result
0010 670E                   BST         #0,R6L      ;  to bit 0 of R6L
0012 7272                   BCLR        #7,R2H      ;Bit clear bit 7 of R2H
                    ;
0014 0D11                   MOV.W       R1,R1       ;
0016 4604                   BNE         LBL1
0018 0D00                   MOV.W       R0,R0
001A 4708                   BEQ         LBL2        ;Branch if R1=R=0
001C                LBL1
001C 0D33                   MOV.W       R3,R3       ;
001E 460C                   BNE         LBL3        ;Branch if not R3 = 0
0020 0D22                   MOV.W       R2,R2
0022 4608                   BNE         LBL3        ;Branch if not R2 = 0
0024                LBL2
0024 79000000               MOV.W       #H'0000,R0  ;Set 0 as result
0028 0D01                   MOV.W       R0,R1
002A 5470                   RTS
002C                LBL3
002C 1D05                   CMP.W       R0,R5
002E 4304                   BLS         LBL4        ;Branch if R0>=R5
0030 1D25                   CMP.W       R2,R5
0032 4218                   BHI         LBL7        ;Branch if R2>=R5
                    ;
```

```
0034                LBL4
0034 770E                     BLD        #0,R6L       ;Load sign bit
0036 450A                     BCS        LBL6         ;Branch if C=1
0038                LBL5
0038 79007F80                 MOV.W      #H'7F80,R0   ;Set #H'7F800000 as result
003C 79010000                 MOV.W      #H'0000,R1
0040 5470                     RTS
                    ;
0042                LBL6
0042 7900FF80                 MOV.W      #H'FF80,R0   ;Set #H'FF800000 as result
0046 79010000                 MOV.W      #H'0000,R1
004A 5470                     RTS
                    ;
004C                LBL7
004C 7778                     BLD        #7,R0L
004E 1200                     ROTXL      R0H
0050 0C0C                     MOV.B      R0H,R4L      ;Set exponent of multiplicand in R4
0052 F400                     MOV.B      #H'00,R4H
                    ;
0054 777A                     BLD        #7,R2L
0056 1202                     ROTXL      R2H
0058 0C2D                     MOV.B      R2H,R5L      ;Set exponent of multiplier in R5
005A F500                     MOV.B      #H'00,R5H
                    ;
005C 7278                     BCLR       #7,R0L       ;Clear bit 7 of R0L
005E 0C00                     MOV.B      R0H,R0H
0060 4704                     BEQ        LBL8         ;Branch if multiplier is denormalized
0062 7078                     BSET       #7,R0L       ;Set implicit MSB
0064 4004                     BRA        LBL9         ;Branch always
0066                LBL8
0066 79140001                 ADD.W      #1,R4
006A                LBL9
006A 727A                     BCLR       #7,R2L       ;Clear bit 7 of R2L
006C 0C22                     MOV.B      R2H,R2H
006E 4704                     BEQ        LBL10        ;Branch if multiplier is denormalized
0070 707A                     BSET       #7,R2L       ;Set implicit MSB
0072 4004                     BRA        LBL11        ;Branch always
0074                LBL10
0074 79150001                 ADD.W      #1,R5
                    ;
0078                LBL11
0078 0954                     ADD.W      R5,R4        ;addition exponents
007A 06FE                     ANDC       #H'FE,CCR    ;Clear C flag of CCR
007C BC7F                     SUBX.B     #H'7F,R4L    ;R4L - #H'7F - C -> R4L
007E B400                     SUBX.B     #H'00,R4H
                    ;
0080 6DF4                     PUSH       R4           ;Push R4
0082 6DF6                     PUSH       R6           ;Push R6
                    ;
0084 0D04                     MOV.W      R0,R4
0086 0D15                     MOV.W      R1,R5
                    ;
0088 0CA2                     MOV.B      R2L,R2H
008A 5E000000                 JSR        @MULA        ;R2L * (R0L:R1) -> (R4:R5)
```

```
008E 6DF4                    PUSH      R4          ;Push R4
0090 6DF5                    PUSH      R5          ;Push R5
0092 0C32                    MOV.B     R3H,R2H     ;
0094 5E000000                JSR       @MULA       ;R3L * (R0L:R1) -> (R4:R5)
0098 6DF4                    PUSH      R4          ;Push R4
009A 6DF5                    PUSH      R5          ;Push R5
                     ;
009C 0CB2                    MOV.B     R3L,R2H     ;R3L * (R0L:R1) -> (R4:R5)
009E 5E000000                JSR       @MULA       ;Push R4
00A2 0D42                    MOV.W     R4,R2       ;Push R5
00A4 0D53                    MOV.W     R5,R3
                     ;
00A6 79010000                MOV.W     #H'0000,R1  ;Clear R1
00AA 6D75                    POP       R5          ;Pop   R5
00AC 6D74                    POP       R4          ;Pop   R4
                     ;
00AE 08D3                    ADD.B     R5L,R3H     ;R3H + R5L + C -> R3H
00B0 0E5A                    ADDX.B    R5H,R2L     ;R2L + R5H + C -> R2L
00B2 0EC2                    ADDX.B    R4L,R2H     ;R2H + R4L + C -> R2H
00B4 0E49                    ADDX.B    R4H,R1L     ;R1L + R4H + C -> R1L
                     ;
00B6 6D75                    POP       R5          ;Pop   R5
00B8 6D74                    POP       R4          ;Pop   R4
00BA 0952                    ADD.W     R5,R2       ;R2  + R5      -> R2L
00BC 0EC9                    ADDX.B    R4L,R1L     ;R1L + R4L + C -> R1L
00BE 0E41                    ADDX.B    R4H,R1H     ;R1H + R4H + C -> R1H
                     ;
00C0 6D76                    POP       R6          ;Pop   R6
00C2 6D74                    POP       R4          ;Pop   R4
00C4 79140001                ADD.W     #1,R4
00C8 0D44                    MOV.W     R4,R4
                     ;
00CA 474E                    BEQ       LBL16       ;Branch if R4=0
00CC 4B4C                    BMI       LBL16       ;Branch if R4<0
00CE                 LBL12
00CE 79340001                SUB.W     #1,R4
00D2 0D44                    MOV.W     R4,R4
00D4 4714                    BEQ       LBL13       ;Branch if R4=0
00D6 100B                    SHLL      R3L         ;Shift mantissa 1 bit left
00D8 1203                    ROTXL     R3H
00DA 120A                    ROTXL     R2L
00DC 1202                    ROTXL     R2H
00DE 1209                    ROTXL     R1L
00E0 1201                    ROTXL     R1H
00E2 44EA                    BCC       LBL12       ;Branch if C=0
00E4 1301                    ROTXR     R1H         ;Rotate mantissa 1 bit right
00E6 1309                    ROTXR     R1L
00E8 1302                    ROTXR     R2H
00EA                 LBL13
00EA 79140001                ADD.W     #1,R4
                     ;
00EE 790500FF                MOV.W     #H'00FF,R5
00F2 1D45                    CMP.W     R4,R5
```

```
00F4 4418                   BCC      LBL15       ;Branch if R5>R4
00F6 770E                   BLD      #0,R6L      ;Load sign bit
00F8 450A                   BCS      LBL14       ;Branch if C=1
00FA 79007F80               MOV.W    #H'7F80,R0  ;Set H'7F800000 to result
00FE 79010000               MOV.W    #H'0000,R1
0102 5470                   RTS
                    ;
0104                LBL14
0104 7900FF80               MOV.W    #H'FF80,R0  ;Set H'FF800000 to product
0108 79010000               MOV.W    #H'0000,R1
010C 5470                   RTS
010E                LBL15
010E 0D11                   MOV.W    R1,R1
0110 462A                   BNE      LBL19       ;Branch if not R1=0
0112 0C22                   MOV.B    R2H,R2H
0114 4626                   BNE      LBL19       ;Branch if not R2H=0
0116 0D10                   MOV.W    R1,R0
0118 5470                   RTS
                    ;
011A                LBL16
011A 79050001               MOV.W    #H'0001,R5  ;Set #H'0001 to R5
011E F618                   MOV.B    #D'24,R6H   ;Set bit counter
0120                LBL17
0120 1101                   SHLR     R1H         ;Shift mantissa 1 bit right
0122 1309                   ROTXR    R1L
0124 1302                   ROTXR    R2H
0126 79140001               ADD.W    #1,R4       ;Increment exponent
012A 1A06                   DEC.B    R6H         ;Decrement bit counter
012C 4706                   BEQ      LBL18       ;Branch if Z=1
012E 1D54                   CMP.W    R5,R4
0130 47DC                   BEQ      LBL15       ;Branch if R5=R4
0132 40EC                   BRA      LBL17       ;Branch always
0134                LBL18
0134 79000000               MOV.W    #H'0000,R0  ;Clear result
0138 0D01                   MOV.W    R0,R1
013A 5470                   RTS
                    ;
013C                LBL19
013C 0C18                   MOV.B    R1H,R0L
013E 0C91                   MOV.B    R1L,R1H
0140 0C29                   MOV.B    R2H,R1L
                    ;
0142 0CC0                   MOV.B    R4L,R0H
0144 7778                   BLD      #7,R0L
0146 4502                   BCS      LBL20       ;Branch if C=1
0148 F000                   MOV.B    #H'00,R0H
014A                LBL20                        ;Correct into floating-point format
014A 1100                   SHLR     R0H
014C 6778                   BST      #7,R0L
014E 770E                   BLD      #0,R6L
0150 6770                   BST      #7,R0H
0152 5470                   RTS
                    ;
                    ;-------------------------------------------------------------
```

```
215                             215     ;
216   0154                      216     MULA                            ;R2H * (R0L:R1) -> (R4:R5)
217   0154 0D04                 217             MOV.W       R0,R4       ;R0  -> R4
218   0156 0D15                 218             MOV.W       R1,R5       ;R1  -> R5
219   0158 0C1E                 219             MOV.B       R1H,R6L     ;R1H -> R6L
220                             220     ;
221   015A 5025                 221             MULXU       R2H,R5      ;R2H * R5L -> R5
222   015C 5026                 222             MULXU       R2H,R6      ;R2H * R6L -> R6
223   015E 5024                 223             MULXU       R2H,R4      ;R2H * R4L -> R4
224                             224     ;
225   0160 08E5                 225             ADD.B       R6L,R5H     ;R5H + R6L       -> R5H
226   0162 0E6C                 226             ADDX.B      R6H,R4L     ;R4L + R6H   + C -> R4L
227   0164 9400                 227             ADDX.B      #H'00,R4H   ;R4H + #H'00 + C -> R4H
228   0166 5470                 228             RTS
229                             229     ;
230                             230             .END
*****TOTAL ERRORS       0
*****TOTAL WARNINGS     0
```

<Reference> Description of Single-Precision Floating-Point Formats

Single-Precision Floating-Point Formats:

1. Internal Representation of Single-Precision Floating Point Numbers
   One of the following formats is used depending on the value of the single-precision floating-point data in this application note (a real number is indicated as R).
   1) Internal Representation When R=0

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

   All the 32 bits are 0.
   2) Normalized Format

31	30 … 23	22 … 0
S	α	β

   α is an index number with an 8-bit-long field. β is a mantissa with a 23-bit-long field. Here, the R value can be represented by the expression below (when $1 \le \alpha \le 254$).
   ↓Implicit MSB
   $R = 2^S \times 2^{\alpha-126} \times (1 + 2^{-1} \times \beta_{22} + 2^{-2} \times 21 + \ldots\ldots + 2^{-23} \times \beta_0)$
   where, βi is the value of the i-th bit of β ($0 \le i \le 22$), and S is the sign bit.
   3) Denormalized Format

31	30 … 23	22 … 0
S	0 0 0 0 0 0 0 0	β

   β is a mantissa with a 23-bit-long field. This format is used to represent a real number that is too small to be represented by the normalized format.
   Here, the R value can be represented by the expression below.
   $R = 2^S \times 2^{-126} \times (2^{-1} \times \beta_{22} + 2^{-2} \times 21 + \ldots\ldots + 2^{-23} \times \beta_0)$
   4) Infinity

31	30 … 23	22 … 0
S	1 1 1 1 1 1 1 1	β

   β is a mantissa with a 23-bit-long field. Note that if all the bits in the index part are 1, the R value is handled as follows, in this application note.
   When S = 0: Plus infinity
   $R = +\infty$
   When S = 1: Minus infinity
   $R = -\infty$

2. Internal Representation Examples
   S = B'0 (binary)
   α = B'10000011 (binary)
   β = B'1011100……0 (binary)
   Under the above conditions, the corresponding R value is represented as follows.
   $R = 2^0 \times 2^{131-126} \times (1 + 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5})$
   $= 16 + 8 + 2 + 1 + 0.5 = 27.5$
   1) Maximum and Minimum Values
   Here, the maximum and minimum values are absolute values. The maximum value is indicated as R_{MAX} and the minimum value is indicated as R_{MIN}. Up to the following values can be represented.
   $R_{MAX} = 2^{254-127} \times (1 + 2^{-1} + 2^{-2} + 2^{-3} + \ldots\ldots + 2^{-23})$
   $\approx 3.27 \times 10^{38}$
   $R_{MIN} = 2^{-126} \times 2^{-23} = 2^{-140} \approx 1.40 \times 10^{-45}$

Revision Record

Rev.	Date	Description	
		Page	Summary
2.00	Feb.28.06	—	Format has been changed from Hitachi version to Renesas version.
3.00	Jun.12.06	6	Error correction