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

H8/300L Series

Square Root of a 32-Bit Binary Number (SQRT)

Introduction

1. The software SQRT finds the square root of a 32-bit binary number and outputs the result as a 16-bit binary number.
2. All arguments used with the software SQRT are unsigned integers.
3. All data is manipulated in general-purpose registers.

Target Device

H8/300L Series

Contents

1. Arguments

Description		Memory area	Data length (bytes)
Input	32-bit binary number	R4, R5	4
Output	Square root	R3	2

2. Changes to Internal Registers and Flags

R0	R1	R2	R3	R4	R5	R6	R7
×	×	×	↕	×	×	×	•

I	U	H	U	N	Z	V	C
•	•	×	•	×	×	×	×

• : No change
× : Undefined
↕ : Result

3. Specifications

Program memory (bytes)
94
Data memory (bytes)
0
Stack (bytes)
0
Clock cycle count
1340
Reentrant
Possible
Relocation
Possible
Interrupt
Possible

4. Notes

The clock cycle count (1340) in the specifications is for the example shown in figure 5.1.

5. Description

5.1 Details of functions

1. The following arguments are used with the software SQRT:
   R4: Sets, as an input argument, the upper word of a 32-bit binary number whose square root is to be found.
   R5: Sets, as an input argument, the lower word of the 32-bit binary number whose square root is to be found.
   R3: The square root of the 32-bit binary number is placed here as an output argument.
2. The following figure illustrates the execution of the software SQRT. When the input arguments are set as shown in (1), the square root is placed in R3 as shown in (2).

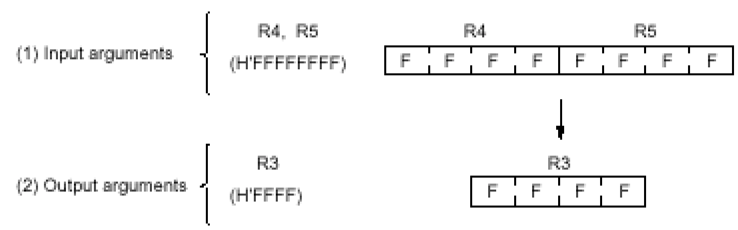


Figure 5.1 Example of Software SQRT Execution

5.2 Notes on usage

1. When the upper bits are not used (see figure 5.2), set them to 0; otherwise, the correct result might not be obtained because undefined data in the higher-order bits is included in computation of the square root.

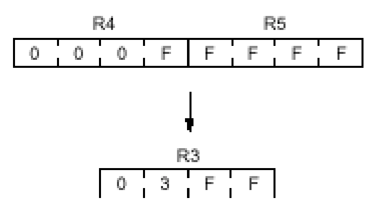


Figure 5.2 Examples of Operation with Upper Bits Unused

2. The fractional part of the result is discarded.

5.3 Data memory

The software SQRT uses no data memory.

5.4 Example of use

Set a 32-bit binary number and call the software SQRT as a subroutine.

```
WORK1      . RES. W    2              ......  Reserves a data memory area in which the
                                              user program places a 32-bit binary
                                              number whose square root is to be found.

WORK2      . RES. W    2              ......  Reserves a data memory area in which the
                                              user program places the square root (16-
                                              bit binary) of the 32-bit binary number.
             :
           MOV. W    @WORK1, R4       ......  Places in the input argument the 32-bit
           MOV. W    @WORK1+2, R5             binary number set by the user program.

           JSR       @SQRT            ......  Calls the software SQRT as a subroutine.

           MOV. W  R3,   @WORK2       ......  Places the 16-bit binary square root (set in
                                              the output argument) in the data memory
                                              area of the user program.
```

5.5 Operation

1. Figure 5.3 shows the method of finding the square root H'05 (binary) of H'22 (a 16-bit binary)

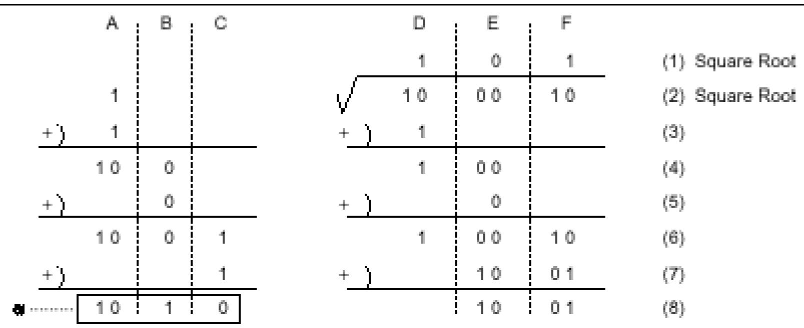
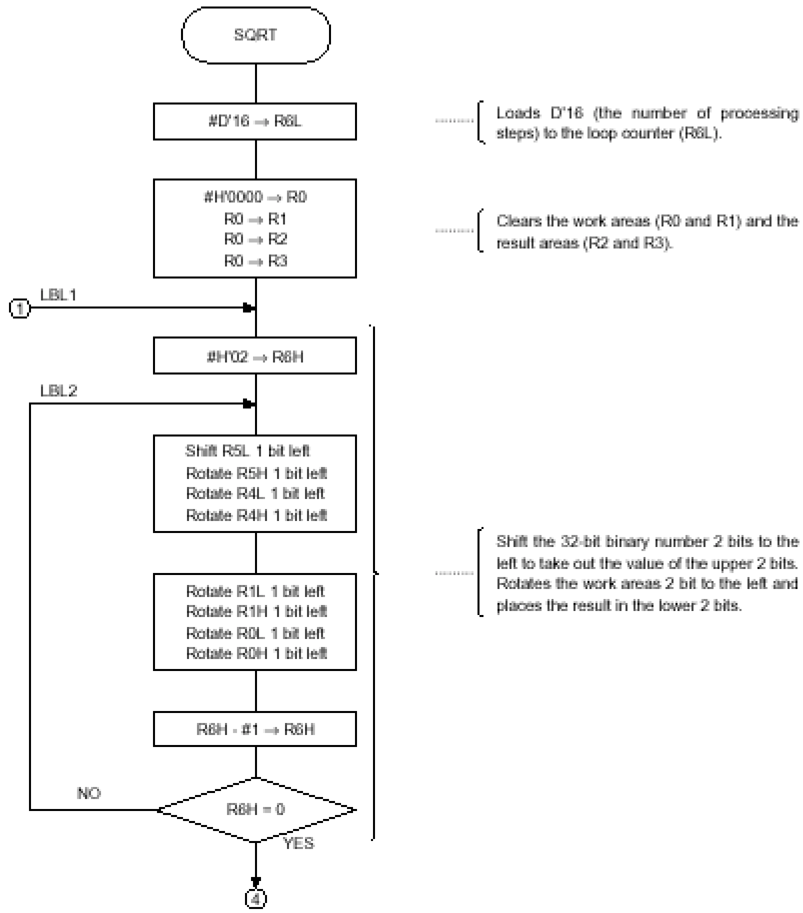
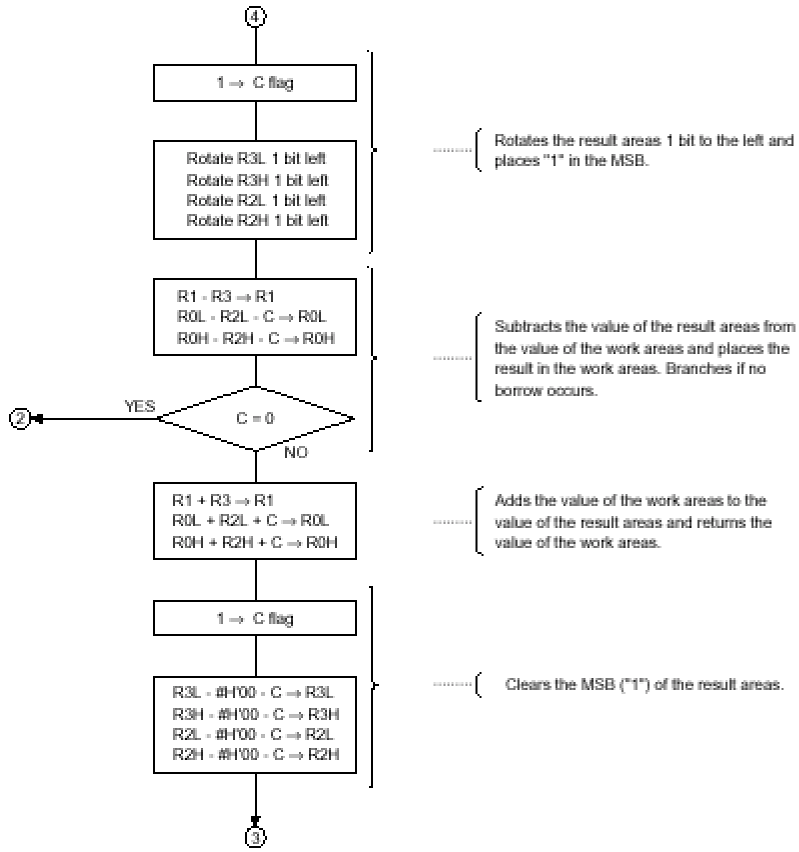


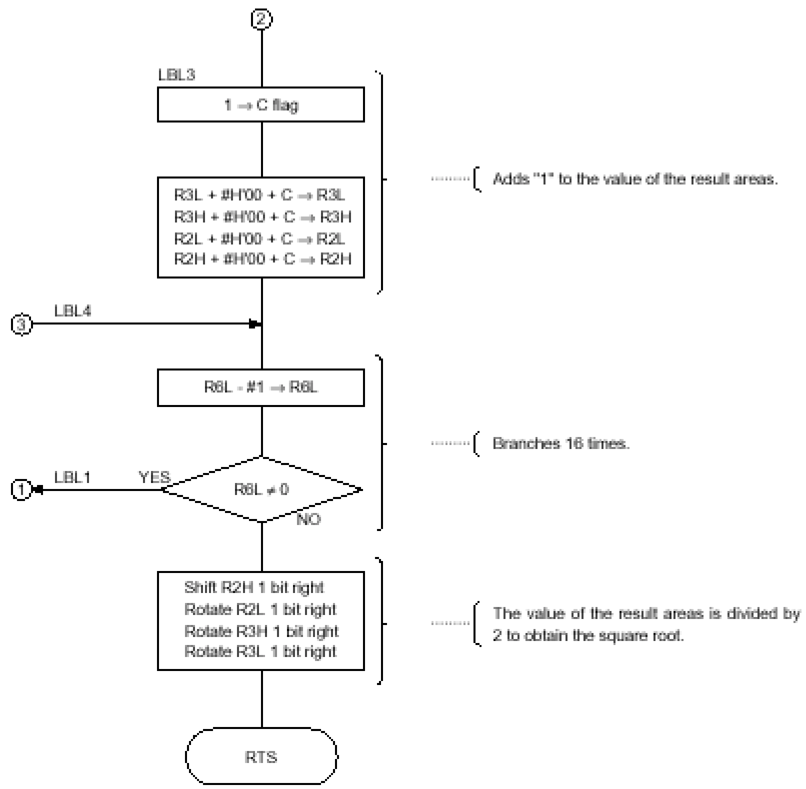
Figure 5.3 Computation to Find Square Root

   A. As shown in figure 5.3, the square root can be found by processing every two-bit unit, from highest to lowest order, of the original number.
   B. The square root (1) is equal to α (found through the operations A, B and C in the figure) divided by 2. The software SQRT computes this α to find the square root.
2. Details on the program are given below:
   A. D'16 is set in R6L, which is the number of two-bit units in a 32-bit binary number.
   B. The area for storage of the square root (R2, R3) and the working area (R0, R1) are cleared.
   C. The two highest-order bits of the input binary number is extracted to R0 and R1 by rotating R4, R5, R0, and R1 two bits to the left.
   D. "1" is placed in R2 and R3. (2)
   E. R2 and R3 are subtracted from R0 and R1 to find the difference (D, (2), (3), and (4)). The difference is placed in R0 and R1.
   F. If the result is positive, R2 and R3 are incremented. (A to (4))
      If the result is negative, R2 and R3 are decremented, and R2 and R3 are added to R0 and R1. (D, E, (6))
3. In the software SQRT, R6 is decremented each time the steps C through F is performed. This processing is repeated until R6 reaches "0".

6. Flowchart


SQRT
#D'16 → R6L
Loads D'16 (the number of processing steps) to the loop counter (R6L).
#H'0000 → R0
R0 → R1
R0 → R2
R0 → R3
Clears the work areas (R0 and R1) and the result areas (R2 and R3).
1
LBL1
#H'02 → R6H
LBL2
Shift R5L 1 bit left
Rotate R5H 1 bit left
Rotate R4L 1 bit left
Rotate R4H 1 bit left
Rotate R1L 1 bit left
Rotate R1H 1 bit left
Rotate R0L 1 bit left
Rotate R0H 1 bit left
Shift the 32-bit binary number 2 bits to the left to take out the value of the upper 2 bits. Rotates the work areas 2 bit to the left and places the result in the lower 2 bits.
R6H - #1 → R6H
NO
R6H = 0
YES
4


4
1 → C flag
Rotate R3L 1 bit left
Rotate R3H 1 bit left
Rotate R2L 1 bit left
Rotate R2H 1 bit left
Rotates the result areas 1 bit to the left and places "1" in the MSB.
R1 - R3 → R1
R0L - R2L - C → R0L
R0H - R2H - C → R0H
C = 0
YES
NO
2
Subtracts the value of the result areas from the value of the work areas and places the result in the work areas. Branches if no borrow occurs.
R1 + R3 → R1
R0L + R2L + C → R0L
R0H + R2H + C → R0H
Adds the value of the work areas to the value of the result areas and returns the value of the work areas.
1 → C flag
R3L - #H'00 - C → R3L
R3H - #H'00 - C → R3H
R2L - #H'00 - C → R2L
R2H - #H'00 - C → R2H
Clears the MSB ("1") of the result areas.
3


2
LBL3
1 → C flag
R3L + #H'00 + C → R3L
R3H + #H'00 + C → R3H
R2L + #H'00 + C → R2L
R2H + #H'00 + C → R2H
Adds "1" to the value of the result areas.
3
LBL4
R6L - #1 → R6L
Branches 16 times.
1
LBL1
YES
R6L ≠ 0
NO
Shift R2H 1 bit right
Rotate R2L 1 bit right
Rotate R3H 1 bit right
Rotate R3L 1 bit right
The value of the result areas is divided by 2 to obtain the square root.
RTS

7. Program List

```
*** H8/300 ASSEMBLER VER 1.0B ** 08/18/92 10:23:40
PROGRAM NAME =
                                        ;****************************************************************************
                                        ;*
                                        ;*      00 - NAME                   :32 BIT SQUARE ROOT (SQRT)
                                        ;*
                                        ;****************************************************************************
                                        ;*
                                        ;*      ENTRY                       :R4,R5 (32 BIT BINARY)
                                        ;*
                                        ;*      RETURN                      :R3 (SQUARE ROOT)
                                        ;*
                                        ;****************************************************************************
                                        ;
SQRT_cod C      0000                            .SECTION  SQRT_code,CODE,ALIGN=2
                                                .EXPORT   SQRT
                                        ;
SQRT_cod C              00000000        SQRT    .EQU $                      ;Entry point
SQRT_cod C      0000    FE10                    MOV.B     #D'16,R6L         ;Set shift counter
SQRT_cod C      0002    79000000                MOV.W     #H'0000,R0        ;Clear R0
SQRT_cod C      0006    0D01                    MOV.W     R0,R1             ;Clear R1
SQRT_cod C      0008    0D02                    MOV.W     R0,R2             ;Clear R2
SQRT_cod C      000A    0D03                    MOV.W     R0,R3             ;Clear R3
SQRT_cod C      000C                    LBL1
SQRT_cod C      000C    F602                    MOV.B     #H'02,R6H
SQRT_cod C      000E                    LBL2
SQRT_cod C      000E    100D                    SHLL.B    R5L               ;Shift 32 bit binary 1 bit left
SQRT_cod C      0010    1205                    ROTXL.B   R5H
SQRT_cod C      0012    120C                    ROTXL.B   R4L
SQRT_cod C      0014    1204                    ROTXL.B   R4H
SQRT_cod C      0016    1209                    ROTXL.B   R1L
SQRT_cod C      0018    1201                    ROTXL.B   R1H
SQRT_cod C      001A    1208                    ROTXL.B   R0L
SQRT_cod C      001C    1200                    ROTXL.B   R0H
SQRT_cod C      001E    1A06                    DEC.B     R6H               ;Decrement R6H
SQRT_cod C      0020    46EC                    BNE       LBL2              ;Branch if Z=0
SQRT_cod C      0022    0401                    ORC.B     #H'01,CCR         ;Set C flag of CCR
SQRT_cod C      0024    120B                    ROTXL.B   R3L               ;Rotate square root
SQRT_cod C      0026    1203                    ROTXL.B R3H
SQRT_cod C      0028    120A                    ROTXL.B R2L
SQRT_cod C      002A    1202                    ROTXL.B R2H
SQRT_cod C      002C    1931                    SUB.W     R3,R1             ;R1 - R3 -> R1
SQRT_cod C      002E    1EA8                    SUBX.B    R2L,R0L           ;R0L - R2L - C -> R0L
SQRT_cod C      0030    1E20                    SUBX.B    R2H,R0H           ;R0H - R2H - C -> R0H
SQRT_cod C      0032    4412                    BCC       LBL3              ;Branch if C = 0
SQRT_cod C      0034    0931                    ADD.W     R3,R1             ;R1 + R3 -> R1
SQRT_cod C      0036    0EA8                    ADDX.B    R2L,R0L           ;R0L + R2L + C -> R0L
SQRT_cod C      0038    0E20                    ADDX.B    R2H,R0H           ;R0H + R2H + C -> R0H
SQRT_cod C      003A    0401                    ORC.B     #H'01,CCR         ;Bit set C flag of CCR
SQRT_cod C      003C    BB00                    SUBX.B    #H'00,R3L         ;R3L - #H'00 - C -> R3L
```

```
SQRT_cod C   003E   B300              SUBX.B    #H'00,R3H     ;R3H - #H'00 - C -> R3H
SQRT_cod C   0040   BA00              SUBX.B    #H'00,R2L     ;R2L - #H'00 - C -> R2L
SQRT_cod C   0042   B200              SUBX.B    #H'00,R2H     ;R2H - #H'00 - C -> R2H
SQRT_cod C   0044   400A              BRA       LBL4          ;Branch always
SQRT_cod C   0046              LBL3
SQRT_cod C   0046   0401              ORC.B     #H'01,CCR     ;Bit set C flag of CCR
SQRT_cod C   0048   9B00              ADDX.B    #H'00,R3L     ;R3L + #H'00 + C -> R3L
SQRT_cod C   004A   9300              ADDX.B    #H'00,R3H     ;R3H + #H'00 + C -> R3H
SQRT_cod C   004C   9A00              ADDX.B    #H'00,R2L     ;R2L + #H'00 + C -> R2L
SQRT_cod C   004E   9200              ADDX.B    #H'00,R2H     ;R2H + #H'00 + C -> R2H
SQRT_cod C   0050              LBL4
SQRT_cod C   0050   1A0E              DEC.B     R6L           ;Decrement shift counter
SQRT_cod C   0052   46B8              BNE       LBL1          ;Branch if Z=0
SQRT_cod C   0054   1102              SHLR.B    R2H
SQRT_cod C   0056   130A              ROTXR.B   R2L
SQRT_cod C   0058   1303              ROTXR.B   R3H           ;Rotate square root
SQRT_cod C   005A   130B              ROTXR.B   R3L
SQRT_cod C   005C   5470              RTS
                               ;
                                      .END
*****TOTAL ERRORS 0
*****TOTAL WARNINGS 0
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.18.03	—	First edition issued