To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

SuperH RISC Engine

SH-DSP Software

Application Note

ADE-502-069
Rev. 1.0
9/21/1999
Hitachi, Ltd.

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi’s or any third party’s patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party’s rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi’s sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi’s sales office for any questions regarding this document or Hitachi semiconductor products.

Preface

The SH-DSP is a CPU core belonging to the SuperH RISC engine family. It is a 32-bit RISC microcontroller based on the SH-2 CPU, optimized for signal processing performance, and incorporating a DSP unit.

These application notes contain example code that makes use of the special features of the SH-DSP as well as explanations of how to utilize the hardware. It is hoped that these application notes will be of use to programmers designing applications that make use of the DSP functions.

Note that though the operation of the example code contained in these application notes has been verified, it is still necessary to confirm its operation when in an actual implementation.

For more information on the hardware, please refer to the hardware manual for the appropriate product.

Please feel free to contact Hitachi for detailed information on development systems.

SH-DSP Code Samples

These application notes contain example code written to illustrate the special features of the SH-DSP.

Figure 1 shows the format used for listings of source code in the application notes. The main program code is transferred to XRAM and the program is executed in XRAM. This format is compatible with the SH7612. When using other SH-DSP models, the following modifications and cautions apply:

- XRAM starting address setting .... (1)
- Vector and stack pointer (YRAM ending address + 1 byte) settings .... (2)
- Usage of commands with other SH-DSP models .... (3)
- Since space for the data used by the main program is reserved in XRAM or YRAM, changes to XRAM or YRAM address settings to match microcontroller used .... (4)

```
;*****************************************************************************
;*                          Symbol definition
;*****************************************************************************
;       [      XRAM address (SH7612)     ]
XRAM_TOP        .EQU   H'1000E000 -------------------------------------- (2)
;*****************************************************************************
;*                      Program transfer routine
;*****************************************************************************
                .SECTION VECT,CODE,LOCATE=H'0
;
                .DATA.L     _PRES        ;_PRES     -------------------- (1)
                .DATA.L     H'10020000   ;SP
                .SECTION ROM,CODE,LOCATE=H'1000

_PRES:  MOV.L          #XRAM_TOP,R1
        MOV.L          #MAIN,R10
        MOV.L          #MAIN_E,R11
PRG_MOVE:
        MOV.W          @R10+,R0
        MOV.W          R0,@R1
        ADD            #2,R1
        CMP/GE         R11,R10
        BF             PRG_MOVE
        MOV.L          #XRAM_TOP,R0
        JMP            @R0                  ;Branch to program starting address
        NOP                                 ;at transfer destination

                       Main program ---------------------------------- (3)

                           Data -------------------------------------- (4)
        .END
```

Figure 1 Source Code Format

Contents

RENESAS

Section 1 Example of Calling Functions (DSP Library) from C Source Code

1.1 C Source Code Employing Functions (DSP Library)

The example code below, “dsplbr.c,” illustrates calling the “Mean” function in the DSP library (shdsplib.lib) from C source code.

```
/*
    <<SH-DSP Application Notes>>
        -- DSP library usage example --
        "dsplbr.c"
*/
#include "ensigdsp.h"                           /* Mean value definition */ -------- (1)
#define N 6                                     /* Input data number */

    short dat[6]={45,61,516,3000,-974,10214}  /* Input data */

    #pragma section X                           /* XRAM address */ ------------------ (2)
        static short    datx[N];
    #pragma section Y                           /* YRAM address */ ------------------ (3)
        static short    daty[N];
    #pragma section ANS                         /* Address for storing mean value */
        static short    answer;
    #pragma section


main()
{
    short      i,output[1];                     /* output for storing variable i
                                                   and Mean function calculation
                                                   result */
    int        src_x;                           /* Argument specifying storage area
                                                   for input data */
    for(i=0;i<N;i++)
        {
        datx[i] = dat[i];                       /* Copy input data to XRAM */
        daty[i] = dat[i];                       /* Copy input data to YRAM */
        }

    /*  select XRAM     */
   *1
    src_x = 1;                                  /* Use XRAM area for Mean ---------- (4)
                                                   function calculation */
        Mean(output,datx,N,src_x);              /* Pass Mean function arguments and
                                                   calculate mean value */
        answer = output[0];                     /* Store Mean function calculation
                                                   result at answer address * /

    while(1);                                   /* Processing complete */

}
```

*1 Refer to 1.3 Function Execution Process for details.

(1) The format of the functions in the library shdsplib.lib are defined in the header file ensigndsp.h.

(2) To ensure efficient X bus data transfer with the DSP unit, it is necessary to place datX[N] in XRAM. Section X needs to be set when linking to addresses in XRAM. (See 1.2 Linking Assignments.)

(3) To ensure efficient Y bus data transfer with the DSP unit, it is necessary to place datY[N] in YRAM. Section Y needs to be set when linking to addresses in XRAM. (See 1.2 Linking Assignments.)

(4) If srx_x = 1, an area in XRAM is used for Mean function calculations. If srx_x = 0, an area in YRAM is used.

1.2 Linking Assignments

When using the DSP library the utmost care must be taken to ensure that the section setting is correct. The example code dsplbr.c shown in section 1.1 has two sections, X and Y. If XRAM and YRAM address are not set for these sections, the functions' internal calculations cannot be performed correctly. These addresses are assigned in the subcommand file.

1.2.1 "prglnk1.sub" Subcommand File for Linking

```
INPUT      vect,dsplbr
START      BX(1000ff00),BANS(1000fff0),BY(1001e000) ------------------ (1)
LIBRARY    shdsplib.lib ---------------------------------------------- (2)
PRINT      dsplbr.map
OUTPUT     dsplbr.abs
FORM       A
DEBUG
EXIT
```

(1) BX(1000ff00) assigns #pragma section X (section X) of dsplbr.c to address H'1000FF00.
BY(1001e000) assigns #pragma section Y (section Y) of dsplbr.c to address H'1001E000.

(2) This specifies shdsplib.lib, which includes the Mean function, as the library to be edited.

1.2.2 "ini.bat" Batch File for Creating Absolute Files

```
asmsh vect.src -cpu=shdsp -debug -lis
shc dsplbr.c -cpu=sh2 -lis -debug -include=ensigdsp.h
lnk -subcommand=prglnk1.sub
```

1.2.3 "vect.src" Vector Table for "dsplbr.c" Program, which Uses DSP Library

```
;********************************************************
;*      <<SH-DSP Application Notes>>
;*             -- DSP library usage example --
;*
;*                     "vect.src"
;*******************************************************

             .import           _main

             .section  vect,data,locate=h'0
             .data.l           _main
             .data.l           h'10020000
             .end
```

1.3 Function Execution Process

Excerpts from the example code dsplbr.c shown in section 1.1, and the assembler code resulting from the functions used, as shown below.

```
      :
src_x = 1;

Mean(output,datx,N,src_x;)

answer = output[0]
      :
```

Assembler code resulting from function

```
Address        Label     Assembler
1001e2fc      _Mean     CMP/PZ    R7
1001e2fe                BF        @1001E322:8
1001e300                MOV       #H'01,R1
1001e302                CMP/GT    R1,R7
                    :
1001e486                NEG       R2,R2
1001e488                MOV.W     R2,@R4
1001e48a                RTS
```

In table 1.1, the input data is arranged starting at address H'1000FF00. It is assumed that the data in RAM has been cleared to 0. The data remains the same after the function is executed.

Table 1.1 Memory Map

XRAM Memory

H'1000FF00	002D	003D	0204	0BB8
H'1000FF08	FC32	27E6	0000	0000

Table 1.2 Function Execution Process

Excerpt from dsplbr.c Code	Register Contents
Mean(output,datx,N,src_x);	Before execution: R4=H'1001FFFC, R5=H'1000FF00, R6=6, R7=1 After execution: R4=H'1001FFFC, R5=H'1000FF0C, R6=6, R7=H'10000

The function arguments are assigned the declaration sequence R4 to R7, so output=H'1001FFFC, datx=H'1000FF00, N=6, src_x=1 is passed to the function. The calculation result is held in @R4.

⇩

Table 1.3 C Source Code Execution Process (Process Inside Memory Map)

Excerpt from dsplbr.c Code	YRAM Memory
answer = output[0];	Before execution: H'1001FF00 0000 0000 0000 0000 After execution: H'1001FF00 0860 0000 0000 0000

The C source code then stores the function calculation result from @R4 in answer (H'1001FF0).

Table 1.4 Mean Function Calculation Result

Input Value (decimal)	Input Value (hexadecimal)	Logical Value (decimal)	Logical Value (hexadecimal)	Output Value (hexadecimal)
45	H'2D	2143.666667	H'860 (2144 calculated as a decimal value)	H'860
61	H'3D			
516	H'204			
3000	H'BB8			
–974	H'FC32			
10214	H'27E6			

Section 2 X/Y Bus Data Access

2.1 X Memory Read

Overview

The data from the XRAM_ADD address (H'1000FF00) and XRAM_ADD+2 address (H'1000FF02) is transferred, respectively, to registers X0 and X1.

Description

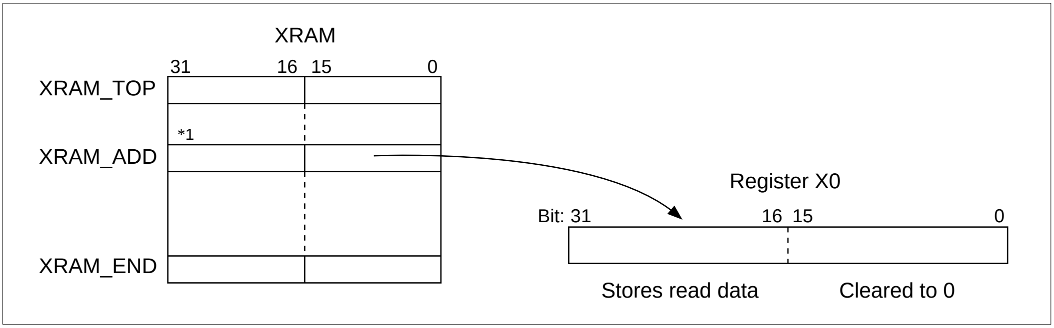
Table 2.1 shows the types of X memory read instructions and the registers that can be used as operands. Data can be read from X memory using the commands listed in table 2.1.

When reading data from X memory the transfer data length is 16 bits, so the data is stored as the upper word of register X0 or X1. When this happens, the lower word of register X0 or X1 is cleared to 0. Processes (1) and (2) in the flowchart are illustrated below.

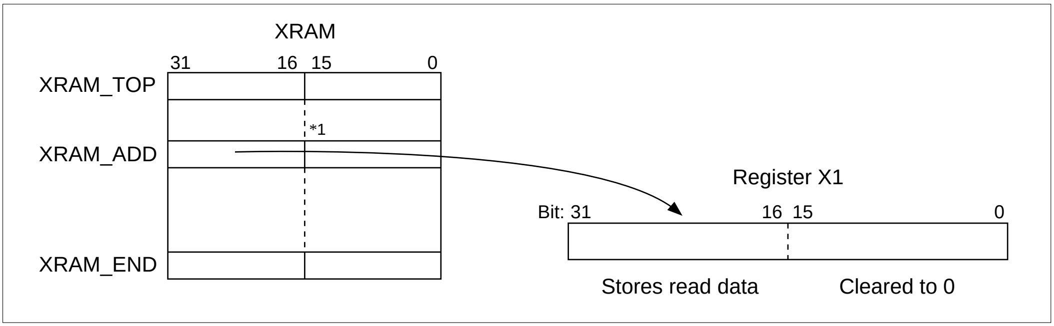
Table 2.1 X Memory Read Instruction Types

X Memory Read Instruction	Source Register (Ax)	Destination Register (Dx)	Index Register (Ix)
MOVX.W @Ax,Dx	R4, R5	X0, X1	R8
MOVX.W @Ax+,Dx			
MOVX.W @Ax+Ix,Dx			

Process (1)

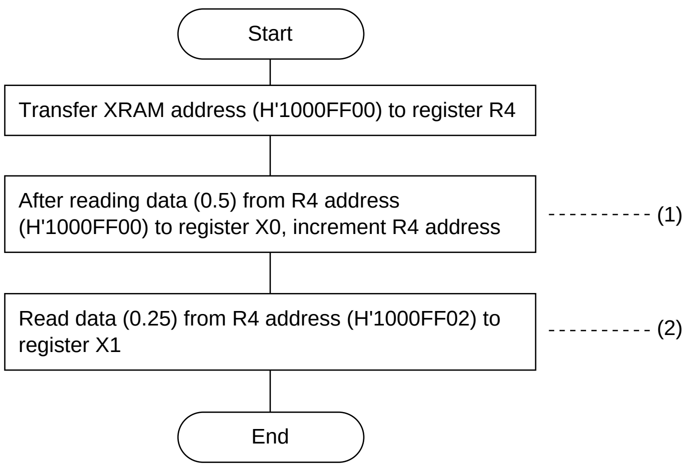


Process (2)



*1 : Ignored

Flowchart



Main Program

```
;**********************************************************************
;*                  X memory read
;**********************************************************************
MAIN:    MOV.L      #XRAM_ADD,R4     ;XRAM_ADD address -> register R4
         MOVX.W     @R4+,X0          ;(H'1000FF00) -> X0
         MOVX.W     @R4,X1           ;(H'1000FF02) -> X1
EXIT:    BRA        EXIT
         NOP
MAIN_E:  NOP
```

Data

```
;***************************************************************
;*            Data
;***************************************************************
              .SECTION XRAM,DATA,LOCATE=H'1000FF00
XRAM_ADD:     .XDATA.W      0.5,0.25
```

2.2 X Memory Write

Overview

The data from the XRAM_ADD1 address (H'1000FF00) and XRAM_ADD1+2 address (H'1000FF02) is transferred the XRAM_ADD2 address and XRAM_ADD2+2 address.

Description

Table 2.2 shows the types of X memory write instructions and the registers that can be used as operands. Data can be written to X memory using the commands listed in table 2.2.

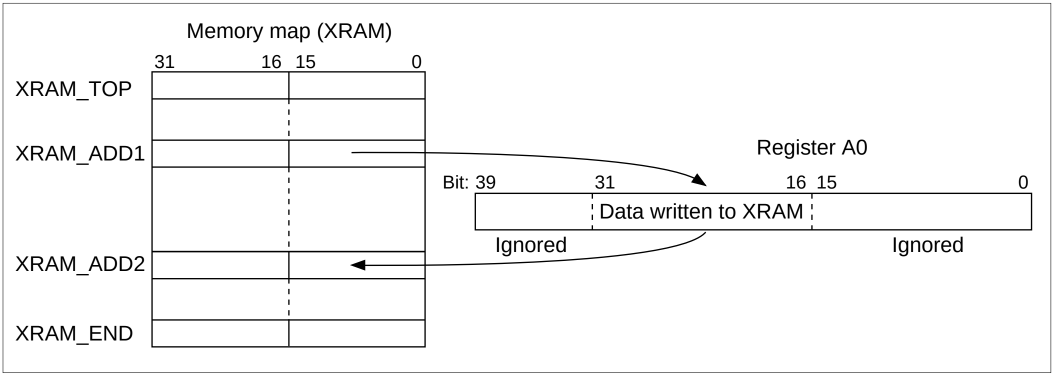
When writing data to X memory the transfer data length is 16 bits, so the upper word data from register A0 or A1, as specified by the instruction, is stored in X memory. When this happens, the guard bit and lower word of register A0 or A1 is ignored. The X memory write instructions can use only registers A0 and A1 as source registers (see Table 2.2 X Memory Write Instruction Types), so when transferring data to register A0 or A1, single data transfers with register A0 or A1 as the destination operand are used. Processes (1) and (2) in the flowchart are illustrated below.

Table 2.2 X Memory Write Instruction Types

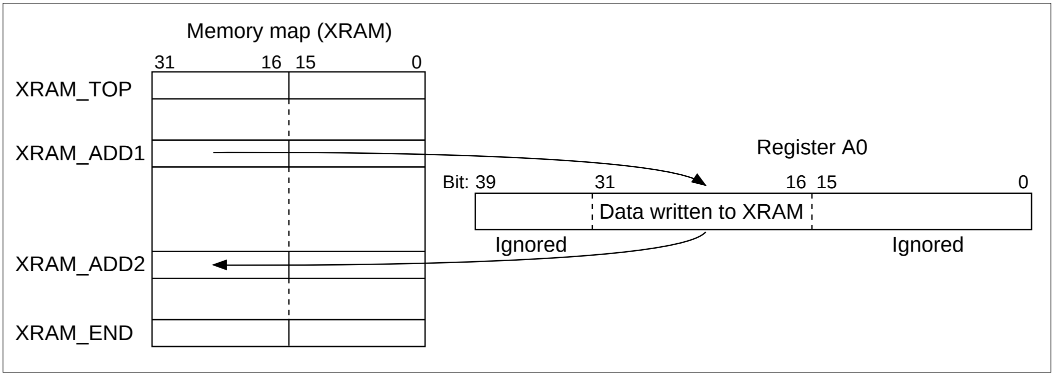
X Memory Write Instruction	Source Register (Da)	Destination Register (Ax)	Index Register (Ix)
MOVX.W Da,@Ax	A0, A1	R4, R5	R8
MOVX.W Da,@Ax+			
MOVX.W Da,@Ax+Ix			

RENESAS

Process (1)

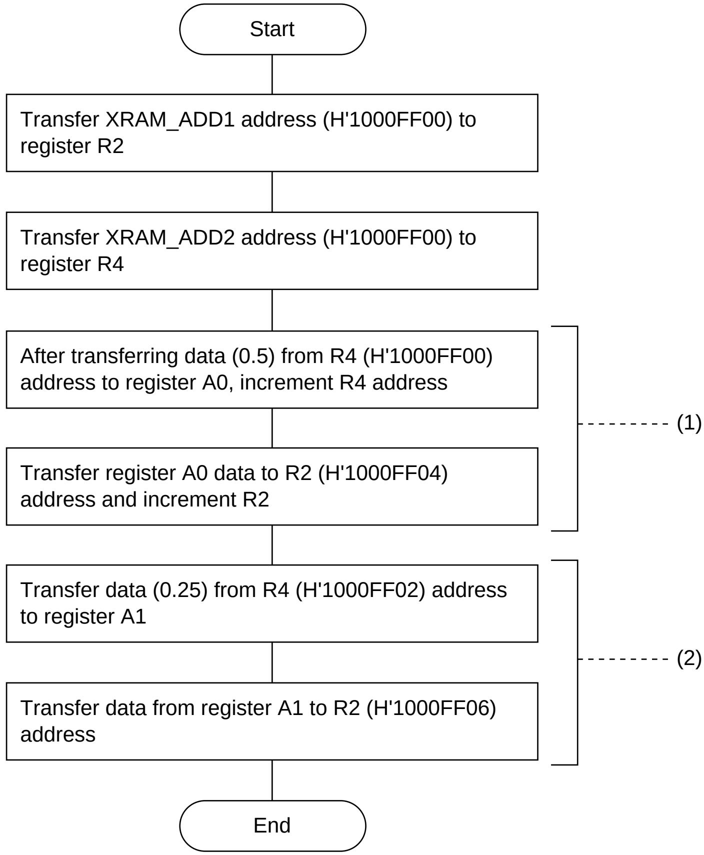


Process (2)



RENESAS

Flowchart



RENESAS

Main Program

```
***********************************************************************
;*                 X memory write
;**********************************************************************
MAIN:    MOV.L      #XRAM_ADD1,R2    ;XRAM_ADD1 -> R2 register
         MOV.L      #XRAM_ADD2,R4    ;XRAM_ADD2 -> R4 register
         MOVS.W     @R2+,A0          ;(H'1000FF00) -> A0 register
         MOVX.W     A0,@R4+          ;A0 register data -> XRAM_ADD2
         MOVS.W     @R2,A1           ;(H'1000FF00) -> A1 register
         MOVX.W     A1,@R4           ;A1 register data -> XRAM_ADD2+2
EXIT:    BRA        EXIT
         NOP
MAIN_E:  NOP
```

Data

```
;***************************************************************
;*            Data
;***************************************************************
              .SECTION XRAM,DATA,LOCATE=H'1000FF00
XRAM_ADD1:    .XDATA.W       0.5,0.25
XRAM_ADD2:    .RES.W         2
```

2.3 Y Memory Read

Overview

The data from the TRAM_ADD address (H'1001FF00) and YRAM_ADD+2 address (H'1001FF02) is transferred, respectively, to registers Y0 and Y1.

Description

Table 2.3 shows the types of Y memory read instructions and the registers that can be used as operands. Data can be read from Y memory using the commands listed in table 2.3.

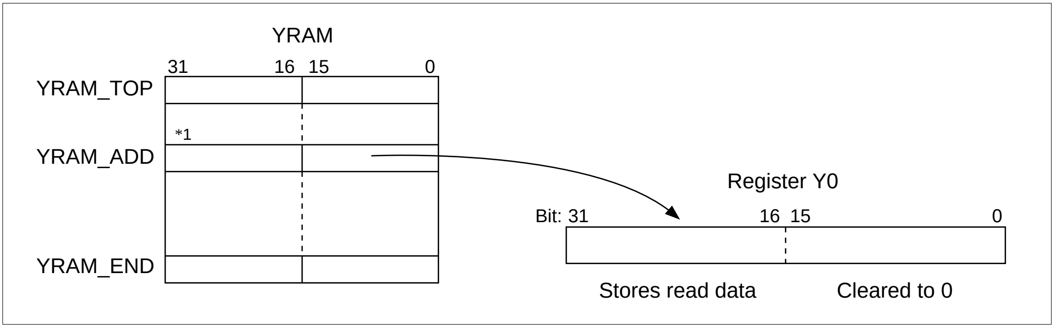
When reading data from Y memory the transfer data length is 16 bits, so the data is stored as the upper word of register Y0 or Y1. When this happens, the lower word of register Y0 or Y1 is cleared to 0. Processes (1) and (2) in the flowchart are illustrated below.

Table 2.3 Y Memory Read Instruction Types

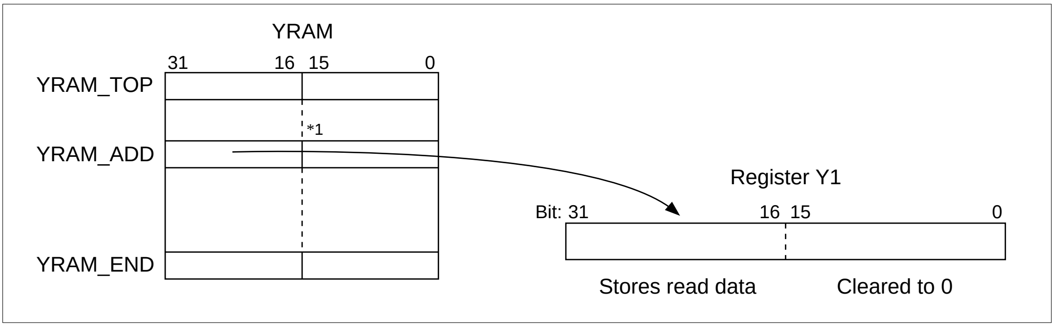
Y Memory Read Instruction	Source Register (Ay)	Destination Register (Dy)	Index Register (Iy)
MOVY.W @Ay,Dy	R6, R7	Y0, Y1	R9
MOVY.W @Ay+,Dy			
MOVY.W @Ay+Iy,Dy			

RENESAS

Process (1)

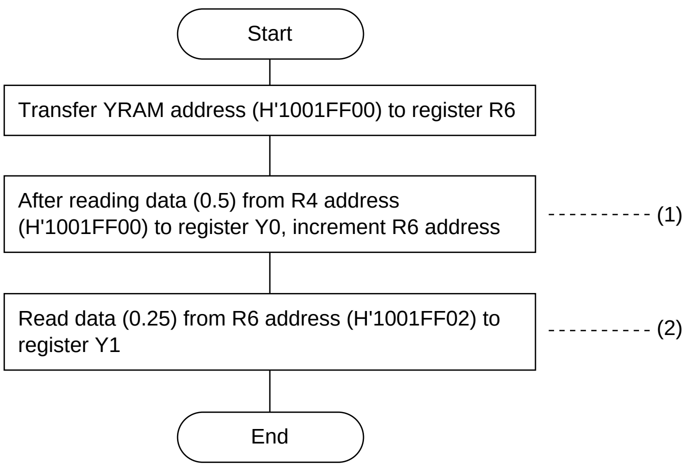


Process (2)



*1 : Ignored

Flowchart



RENESAS

Main Program

```
;*********************************************************************
;*                 Y memory read
;*********************************************************************
MAIN:    MOV.L      #YRAM_ADD,R6     ;YRAM_ADD address -> R6 register
         MOVX.W     @R6+,Y0          ;(H'1001FF00) -> Y0
         MOVX.W     @R6,Y1           ;(H'1001FF02) -> Y1
EXIT:    BRA        EXIT
         NOP
MAIN_E:  NOP
```

Data

```
;**************************************************************
;*            Data
;**************************************************************
              .SECTION YRAM,DATA,LOCATE=H'1001FF00
YRAM_ADD:     .XDATA.W      0.5,0.25
```

2.4 Y Memory Write

Overview

The data from the YRAM_ADD1 address (H'1001FF00) and YRAM_ADD1+2 address (H'1001FF02) is transferred the YRAM_ADD2 address and YRAM_ADD2+2 address.

Description

Table 2.4 shows the types of Y memory write instructions and the registers that can be used as operands. Data can be written to Y memory using the commands listed in table 2.4.

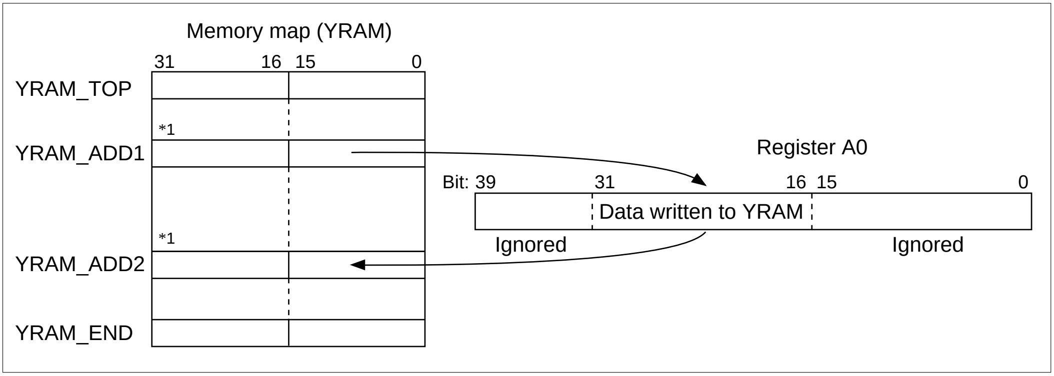
When writing data to Y memory the transfer data length is 16 bits, so the upper word data from register A0 or A1, as specified by the instruction, is stored in Y memory. When this happens, the guard bit and lower word of register A0 or A1 is ignored. The Y memory write instructions can use only registers A0 and A1 as source registers (see Table 2.4 Y Memory Write Instruction Types), so when transferring data to register A0 or A1, single data transfers with register A0 or A1 as the destination operand are used. Processes (1) and (2) in the flowchart are illustrated below.

Table 2.4 Y Memory Write Instruction Types

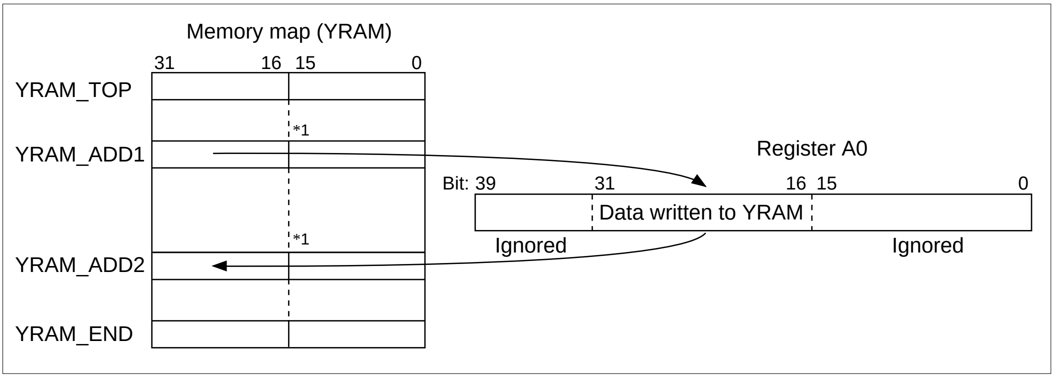
Y Memory Write Instruction	Source Register (Da)	Destination Register (Ax)	Index Register (Ix)
MOVY.W Da,@Ax	A0, A1	R6, R7	R9
MOVY.W Da,@Ax+			
MOVY.W Da,@Ax+Ix			

RENESAS

Process (1)

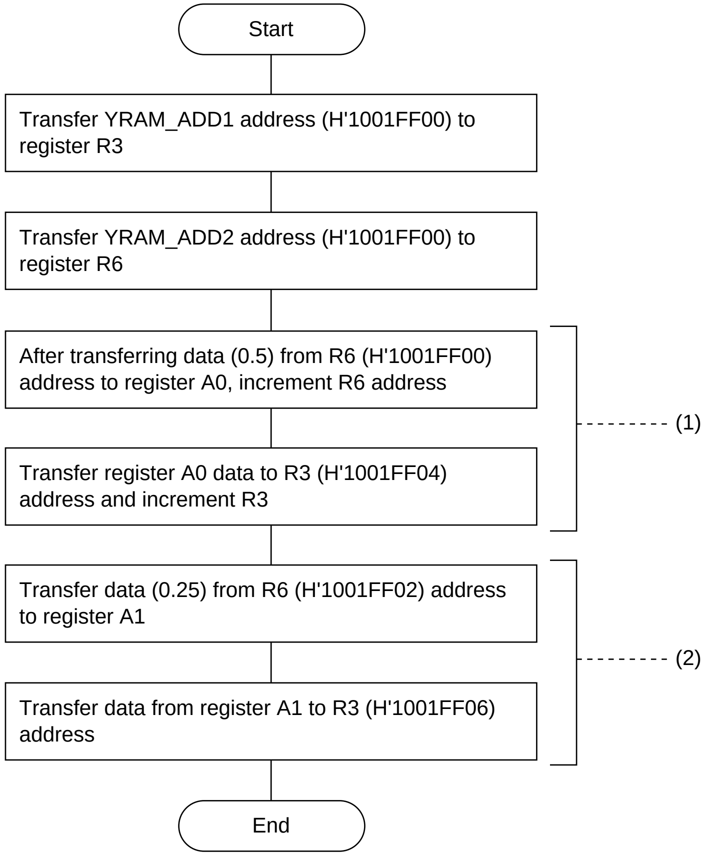


Process (2)



*1 : Ignored

Flowchart



RENESAS

Main Program

```
***********************************************************************
;*                   Y Memory Write
;**********************************************************************
MAIN:    MOV.L      #YRAM_ADD1,R3    ;YRAM_ADD1 -> R3 register
         MOV.L      #YRAM_ADD2,R6    ;YRAM_ADD2 -> R6 register
         MOVS.W     @R3+,A0          ;(H'1001FF00) -> A0 register
         MOVX.W     A0,@R6+          ;A0 register data -> YRAM_ADD2
         MOVS.W     @R3,A1           ;(H'1001FF00) -> A1 register
         MOVX.W     A1,@R6           ;A1 register data -> YRAM_ADD2+2
EXIT:    BRA        EXIT
         NOP
MAIN_E:  NOP
```

Data

```
;****************************************************************
;*             Data
;****************************************************************
              .SECTION YRAM,DATA,LOCATE=H'1001FF00
YRAM_ADD1:    .XDATA.W      0.5,0.25
YRAM_ADD2:    .RES.W        2
```

Section 3 16-bit Fixed-point Multiplication

Overview

Multiplies the 16-bit data at the XRAM-ADD address (H'1000FF000) and the 16-bit data at the YRAM-ADD address (H'1001FF002). The result is stored at the ANS address (H'1001FF002).

Description

1. Data Transfer

   Transfer of the data from the XRAM-ADD address (H'1000FF000) and the YRAM-ADD address (H'1001FF002) is performed using X bus data transfer and Y bus data transfer, as described in 2. X/Y Bus Data Access. In process (1) in the flowchart the XRAM and YRAM data is read simultaneously, but no contention occurs because the X bus and Y bus are independent of each other. The format is shown below.

   The sequence is [X bus data transfer] then [Y bus data transfer]. If these are described in a single step, the instructions may be combined as either [X memory read] [Y memory write] or [X memory write] [Y memory read].

   Format: MOVX.W @R5,X1 MOVY.W @R7,Y1

2. Fixed-point Multiplication

The PMULS instruction is used to perform fixed-point multiplication in process (2) in the flowchart. The format is shown below. The fixed-point multiplication process is shown in figure 3.1. Only the upper word data from source 1 and source 2 is valid. For example, if the longword H'12345678 was read from the source, the portion that would actually be multiplied would be H'1234.

Format: PMULS Se,Sf,Dg

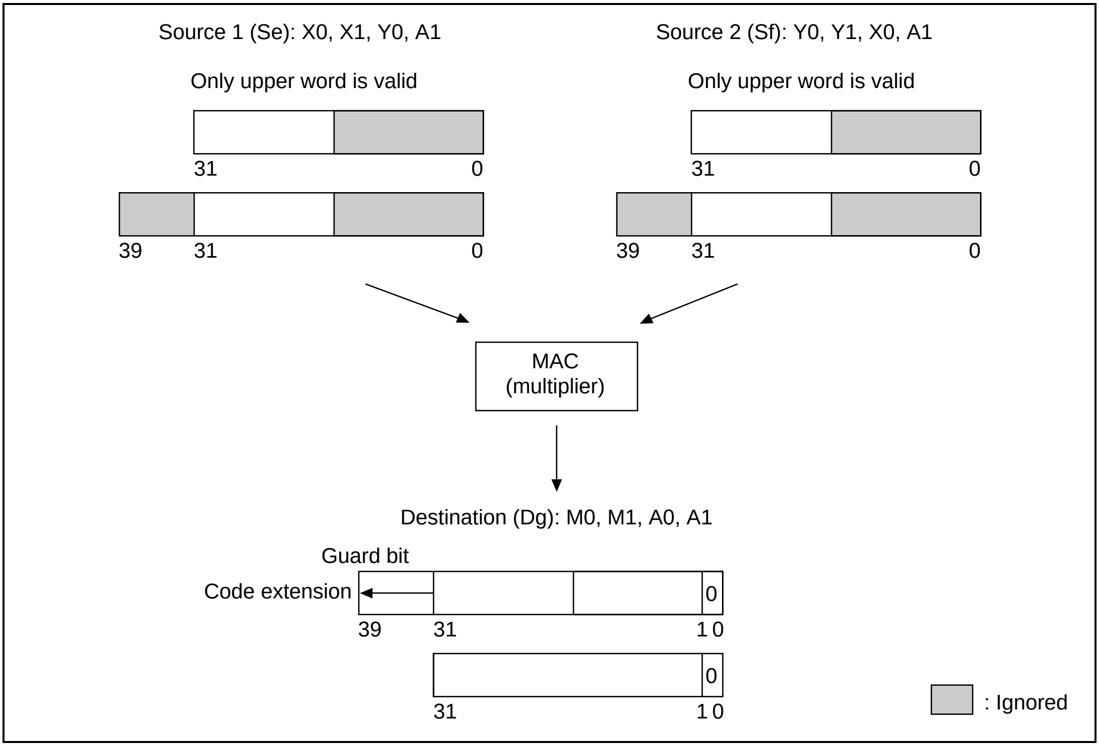


Figure 3.1 Fixed-point Multiplication Process

RENESAS

3. Overflow

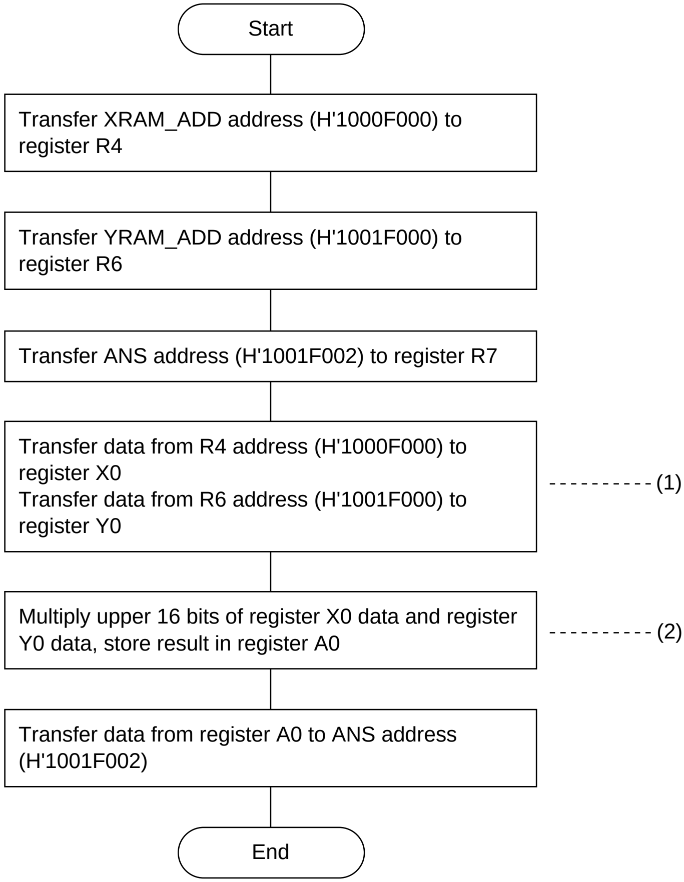
An overflow can occur during fixed-point multiplication only if the operation is H'8000(–1.0) × H'8000(–1.0), in which case the calculation result is H'8000(–1.0). This can happen only when the destination register is a register other than A0 or A1, both of which have guard bits. If the destination register is A0 or A1, the result of the above calculation is the correct value of H'008000000(1.0). Refer to table 3.1 for additional fixed-point multiplication execution examples.

Since the destination register used in the example main program is A0, no overflow problem occurs.

Table 3.1 Fixed-point Multiplication Execution Examples

Operation Example	State of Operation Result	Destination Register	Operation Result
H'4000 (0.5) × H'2000 (0.25)	Positive	M0, M1	H'1000 0000 (0.125)
		A0, A1	H'00 1000 0000 (0.125)
H'0800 (0.0625) × H'FC00 (–0.03125)	Negative	M0, M1	H'FFC00 0000 (–1.95×10^{-3})
		A0, A1	H'FF FFC00 0000 (–1.95×10^{-3})
H'8000 (–1.0) × H'8000 (–1.0)	Overflow	M0, M1	H'8000 0000 (–0.1)
		A0, A1	H'00 8000 0000 (1.0)

Flowchart



Start
Transfer XRAM_ADD address (H'1000F000) to register R4
Transfer YRAM_ADD address (H'1001F000) to register R6
Transfer ANS address (H'1001F002) to register R7
Transfer data from R4 address (H'1000F000) to register X0
Transfer data from R6 address (H'1001F000) to register Y0
(1)
Multiply upper 16 bits of register X0 data and register Y0 data, store result in register A0
(2)
Transfer data from register A0 to ANS address (H'1001F002)
End

Main Program

```
;*****************************************************************************************
;*              16-bit fixed-point multiplication routine
;*****************************************************************************************
MAIN:   MOV.L   #0,R4                                               ;Clear register R4
        MOV.L   #0,R6                                               ;Clear register R6
        MOV.L   #XRAM_ADD,R4                                        ;XRAM address -> register R4
        MOV.L   #YRAM_ADD,R6                                        ;YRAM address -> register R6
        MOV.L   #ANS,R7                                             ;ANS address -> register R7
                              MOVX.W @R4,X0  MOVY.W @R6,Y0          ;XRAM and YRAM address data ->
                                                                    registers X0 and Y0
                        PMULS  X0,Y0,A0                             ;16-bit fixed-point
                                                                    multiplication
                                             MOVY.W A0,@R7          ;Store multiplication result
EXIT:   BRA     EXIT
        NOP
MAIN_E: NOP
```

Data

```
;***************************************************************
;*             Data
;***************************************************************
               .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:      .XDATA.W        0.0625

               .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:      .XDATA.W        0.03125
ANS:           .RES.W          1
```


RENESAS

Section 4 Parallel Execution Instruction

Overview

Four data values obtained sequentially from the XRAM-ADD address (H'1000FF000) and the YRAM-ADD address (H'1001FF000) are added and multiplied. The addition result is stored at the ANS1 address (H'1000FF004) and the multiplication result at the ANS2 address (H'1001FF004).

Description

1. Structure of Parallel Execution Instruction

   The parallel execution instruction is used to transfer data between a DSP register and X memory or Y memory at the same time a DSP operation is being executed. Table 4.1 shows the data transfer and DSP operation structure. The parallel execution instruction comprises a DSP operation portion and a data transfer portion. Table 4.2 lists format examples for the parallel execution instruction. The DSP operation portion is a single instruction like the regular PAND, PINC, and PSHA instructions. However, as shown in table 4.2, its has two-instruction structure the case of the PADD and PMULS instructions, or the PSUB and PMULS instructions. The data transfer portion consists of two instructions, one the data transfer instruction for X memory and the other the data transfer instruction for Y memory. Either one of these data transfer instructions may be used.

Table 4.1 Data Transfer and DSP Operation Structure

	Type	Bus Used	Data Transfer Length	Parallel Processing with DSP Operation	Parallel Processing of Data Transfers	Instruction Length
(1)	Double data transfer	X bus Y bus	16 bits	No	No: One or the other data transfer	16 bits
					Yes: Data transfer with X memory and Y memory at same time	
(2)				Yes	No: One or the other data transfer	32 bits
					Yes: Data transfer with X memory and Y memory at same time	
	Single data transfer	C bus*1	16 bits 32 bits	No		16 bits

*1: Note that the name differs depending on the product.

RENESAS

Table 4.2 Parallel Execution Instruction Format Examples

DSP Operation Portion				Data Transfer Portion			
PADD	X0,Y0,A0	PMULS	X0,Y0,A1	MOVX.W	A0,@R4	MOVY.W	A1,@R6
PSUB	X1,Y1,A1	PMULS	X0,Y1,A0	MOVX.W	@R5,X1	MOVY.W	@R7,Y1
PADD	X0,Y0,A0	PMULS	X0,Y0,A1	MOVX.W	A0,@R4		
PINC	X0,Y0,A0			MOVY.W	@R6,Y1		
PAND	X0,Y0,A0			MOVX.W	A0,@R5		
PSHA	X0,Y0,A0			MOVX.W	@R4,X1	MOVY.W	A1,@R7

2. Parallel Processing of Double Data Transfer and DSP Operation

   Process (1) in the flowchart on the following page is double data transfer with no DSP operation instruction parallel processing, which is indicated as **(1)** in table 4.1, and processes (2) and (3) are double data transfer with parallel processing of DSP operation instructions, which is indicated as **(2)** in table 4.1. Processes (2) and (3) consist of four instructions, which is the maximum number that can be declared in a single step. In this case, one execution state is used.

3. Effect of DSP Operation Portion Result on Data Transfer Portion

   Table 4.3 shows the effect of the DSP operation portion result on the data transfer portion. Instruction 2 (process (3)) uses A0 and A1 as the destination register for the DSP operation portion and also as the source register for the data transfer portion. However, the result of the DSP operation portion is not the data stored in the data transfer portion. In this case the underlined registers are affected, so the calculation result from instruction 1 (process (2)) operation portion is stored in the instruction 2 (process (3)) data transfer portion.

   Figure 4.1 shows the instruction 2 pipeline flow. When instructions are executed in parallel, each of the instructions is processed independently, as shown in figure 4.1. The reason the DSP operation portion result does not become the data stored in the data transfer portion in this case is that the WB/DSP stage, in which DSP operations are performed using PADD and PMULS, is later than the MA stage, in which memory access is performed using MOVX.W and MOVY.W.

   Note that after the execution of instruction 2 (process (3)), the X1 and Y1 addition and multiplication results are stored in registers A0 and A1.

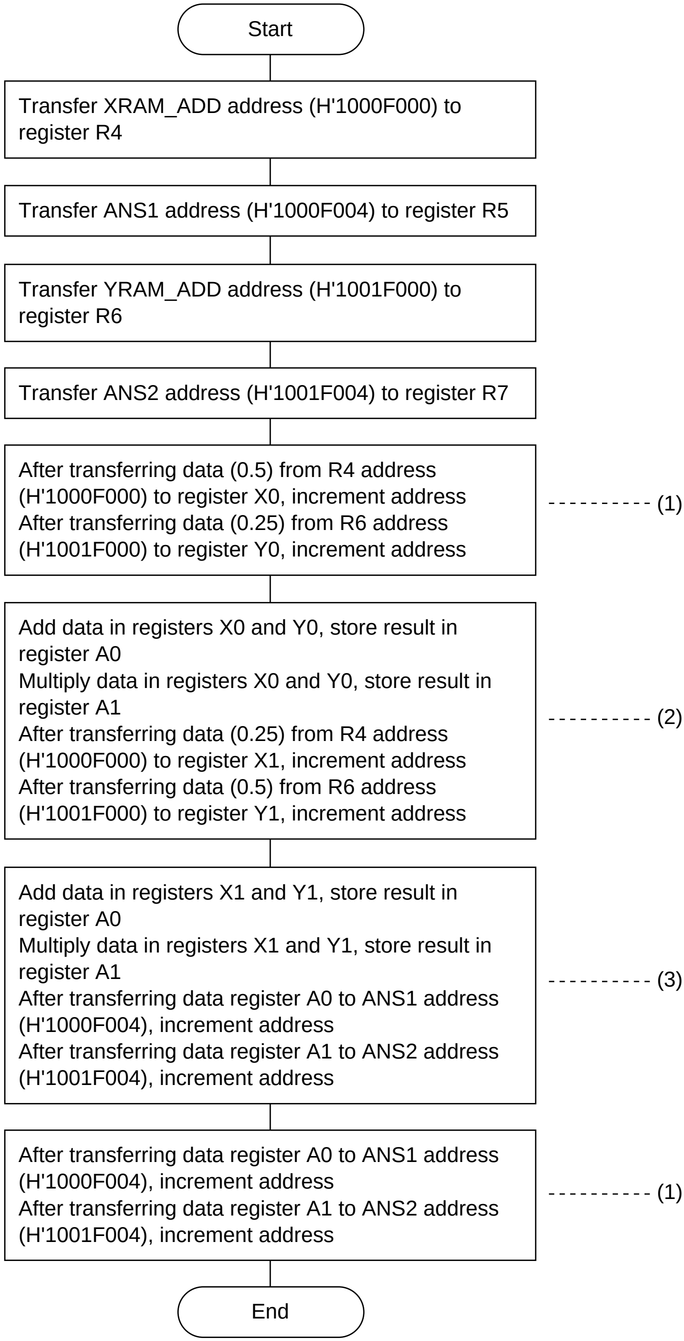
Table 4.3 Effect of DSP Operation Portion Result on Data Transfer Portion

Excerpts from Main Program							
;Instruction 1							
PADD	X0,Y0,A0	PMULS	X0,Y0,A1	MOVX.W	@R4,X1	MOVY.W	@R6,Y1
;Instruction 2							
PADD	X1,Y1,A0	PMULS	X1,Y1,A1	MOVX.W	A0,@R5+	MOVY.W	A1,@R7+
Content of Registers							
Before execution of instruction 2: X1=H'1000 0000, Y1=H'0800 0000, A0=H'6000 0000, A1=H'1000 0000							
After execution of instruction 2: X1=H'1000 0000, Y1=H'0800 0000, A0=H'1800 0000, A1=H'0100 0000							

Slot		↔	↔	↔	↔	↔
PADD	X1,Y1,A0	IF	ID	EX	MA	WB/DSP
PMULS	X1,Y1,A1	IF	ID	EX	MA	WB/DSP
MOVX.W	A0,@R5+	IF	ID	EX	MA	WB/DSP
MOVY.W	A1,@R7+	IF	ID	EX	MA	WB/DSP

Figure 4.1 Instruction 2 Pipeline Flow

Flowchart


Start
Transfer XRAM_ADD address (H'1000F000) to register R4
Transfer ANS1 address (H'1000F004) to register R5
Transfer YRAM_ADD address (H'1001F000) to register R6
Transfer ANS2 address (H'1001F004) to register R7
After transferring data (0.5) from R4 address (H'1000F000) to register X0, increment address
After transferring data (0.25) from R6 address (H'1001F000) to register Y0, increment address
(1)
Add data in registers X0 and Y0, store result in register A0
Multiply data in registers X0 and Y0, store result in register A1
After transferring data (0.25) from R4 address (H'1000F000) to register X1, increment address
After transferring data (0.5) from R6 address (H'1001F000) to register Y1, increment address
(2)
Add data in registers X1 and Y1, store result in register A0
Multiply data in registers X1 and Y1, store result in register A1
After transferring data register A0 to ANS1 address (H'1000F004), increment address
After transferring data register A1 to ANS2 address (H'1001F004), increment address
(3)
After transferring data register A0 to ANS1 address (H'1000F004), increment address
After transferring data register A1 to ANS2 address (H'1001F004), increment address
(1)
End

Main Program

```
;*******************************************************************************************
;*                      Parallel data transfer routine
;*******************************************************************************************
MAIN:   MOV.L           #XRAM_ADD,R4
        MOV.L           #ANS1,R5
        MOV.L           #YRAM_ADD,R6
        MOV.L           #ANS2,R7
                                            MOVX.W @R4+,X0  MOVY.W @R6+,Y0
                                                            ;No parallel processing
        PADD  X0,Y0,A0  PMULS X0,Y0,A1     MOVX.W @R4,X1   MOVY.W @R6,Y1
                                                            ;Parallel processing
        PADD  X1,Y1,A0  PMULS X1,Y1,A1     MOVX.W A0,@R5+  MOVY.W A1,@R7+
                                                            ;Parallel processing
                                            MOVX.W A0,@R5   MOVY.W A1,@R7
                                                            ;No parallel processing
EXIT:   BRA             EXIT
        NOP
MAIN_E: NOP
```

Data

```
;***********************************************************************
;*            Data(X/YRAM)
;***********************************************************************
             .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:    .XDATA.W    0.5,0.125                 ;DSP operation data
ANS1:        .RES.W      2                         ;DSP operation result storage area

             .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:    .XDATA.W    0.25,0.0625               ;DSP operation data
ANS2:        .RES.W      2                         ;DSP operation result storage area
```


RENESAS

Section 5 Repeat Instruction

Overview

The average of ten data values stored in XRAM and YRAM is obtained. To accomplish this, the repeat function is used for transferring data from XRAM and YRAM to the DSP unit, and for adding the ten data values.

Description

1. DSP Repeat Control

   Three settings are required in order to perform repeat control: I the start address setting for the program to be repeated, II the end address setting for the program to be repeated, III and the setting for the number of repetitions to be performed. After settings I through III have been completed, Process IV is to start the program to be repeated. Note that a minimum of one instruction is required between the processing of III and IV.

   The sequence of processes I through IV is shown below.

   I LDRS instruction is used to set the repeat start address in the RS register.
   II LDRE instruction is used to set the repeat end address in the RE register.
   III SETRC instruction is used to set the number of repetitions in the RC register.
   : (Minimum of one instruction inserted.)
   IV Program to be repeated is started.

   Process (1) in the flowchart on the next page corresponds to I through III above. After the program to be repeated is started (IV), it is repeated within the scope of process (2). Two main programs are shown in the example, but their function is the same. In (1) repeat control instructions (LDRS, LDRE, and SETRC) are used, and in (2) the extended instruction REPEAT is used. REPEAT automatically generates the CPU instructions (LDRS, LDRE, and SETRC) used to repeat the instructions between the start and end addresses. In the format shown below if the number of repetitions is omitted, the SETRC instruction is not generated.

   Format: REPEAT [start address], [end address], [number of repetitions]

In program (1) the repeat start and end addresses are different from the actual addresses, and this is because the address setting change depending on the number of instructions in the program to be repeated. Table 5.1 shows how the RS and RE settings change depending on the number of instructions within the range to be repeated. These are the addresses actually repeated by the program when the repeat start and end addresses are set in RS and RE. Therefore, it is necessary to label the repeat start and end addresses while keeping the offsets listed in Table 5.1 in mind. The setting method for RS and RE in program (1) is described on the next page.

RPT_S0+N: Address N bytes from the instruction preceding the instruction at the start address of the program to be repeated

RPT_S: Start address of the program to be repeated

RPT_E: End address of the program to be repeated

RPT_E3+4: Address 4 bytes from the instruction three instructions before the instruction at the end address of the program to be repeated

Table 5.1 RS and RE Setting Values Based on Number of Instructions Within Repeat

	Number of Instructions in Program to be Repeated			
	1	**2**	**3**	**4**
RS	RPT_S0 + 8	RPT_S0 + 6	RPT_S0 + 4	RPT_S
RE	RPT_S0 + 4	RPT_S0 + 4	RPT_S0 + 4	RPT_E3 + 4

2. Repeat Control Using CPU Instructions

Example (a) shows the method for setting addresses in RS and RE. If there are three instructions in the portion to be repeated, RS and RE must be set to the RPT_S0+4 address, as indicated in Table 5.1. The double data transfer instructions in lines **(1)** and **(2)** of this program have a 16-bit instruction length, so the RPT_S0+4 address corresponds to the RPT_E0 address. If RS and RE are set to the address RPT_E0, the result is program (b).

```
        LDRS            RPT_S0+4 address           ;Repeat start address
        LDRE            RPT_S0+4 address           ;Repeat end address
        SETRC           #5                         ;Repeat counter setting/5 repetitions
RPT_S0:                         (1)  MOVX.W @R5,X1  MOVY.W @R7,Y1    ;Clear X1, Y1 = 1/10
RPT_S:                          (2)  MOVX.W @R4+,X0 MOVY.W @R6+,Y0
RPT_E0: PADD   X0,Y0,M0
RPT_E:  PADD   X1,M0,X1                                              ;X1/data total
                       PMULS  X1,Y1,A1                               ;A1/average value
```

(a) RS and RE Address Setting Method

```
        LDRS            RPT_E0                     ;Repeat start address
        LDRE            RPT_E0                     ;Repeat end address
        SETRC           #5                         ;Repeat counter setting/5 repetitions
RPT_S0:                            MOVX.W @R5,X1  MOVY.W @R7,Y1     ;Clear X1, Y1 = 1/10
RPT_S:                             MOVX.W @R4+,X0 MOVY.W @R6+,Y0
RPT_E0: PADD   X0,Y0,M0
RPT_E:  PADD   X1,M0,X1                                              ;X1/data total
                       PMULS  X1,Y1,A1                               ;A1/average value
```

(b) RS and RE Address Setting Method

3. Repeat Control Using Extended Instructions

When the extended instruction REPEAT is used there is no need to perform complicated labeling, as is the case when using CPU instructions for repeat control. The following explanation is based on the expanded image of a portion of a repeat program shown as (a) below. With REPEAT one only needs to declare the labels for the start (RPT_S) and end (RPT_E) addresses of the program to be repeated, and then the assembler automatically calculates the address values to be used for the RS and RE settings (RPT_E0 if the code to be repeated contains three instructions), and generates the LDRS, LDRE, and SETRC instructions. When the extended instruction REPEAT is actually used, the result is the repeat program shown in example (b) below.

```
        REPEAT  RPT_S,RPT_E,#5

        LDRS            RPT_E0   ;RPT_S0+4
        LDRE            RPT-E0   ;RPT_S0+4
        SETRC           #5
RPT_S0:                                             MOVX.W @R5,X1   MOVY.W @R7,Y1
RPT_S:                                              MOVX.W @R4+,X0  MOVY.W @R6+,Y0
RPT_E0: PADD   X0,Y0,M0
RPT_E:  PADD   X1,M0,X1
                        PMULS  X1,Y1,A1
```

Expands to CPU instructions for repeat control.

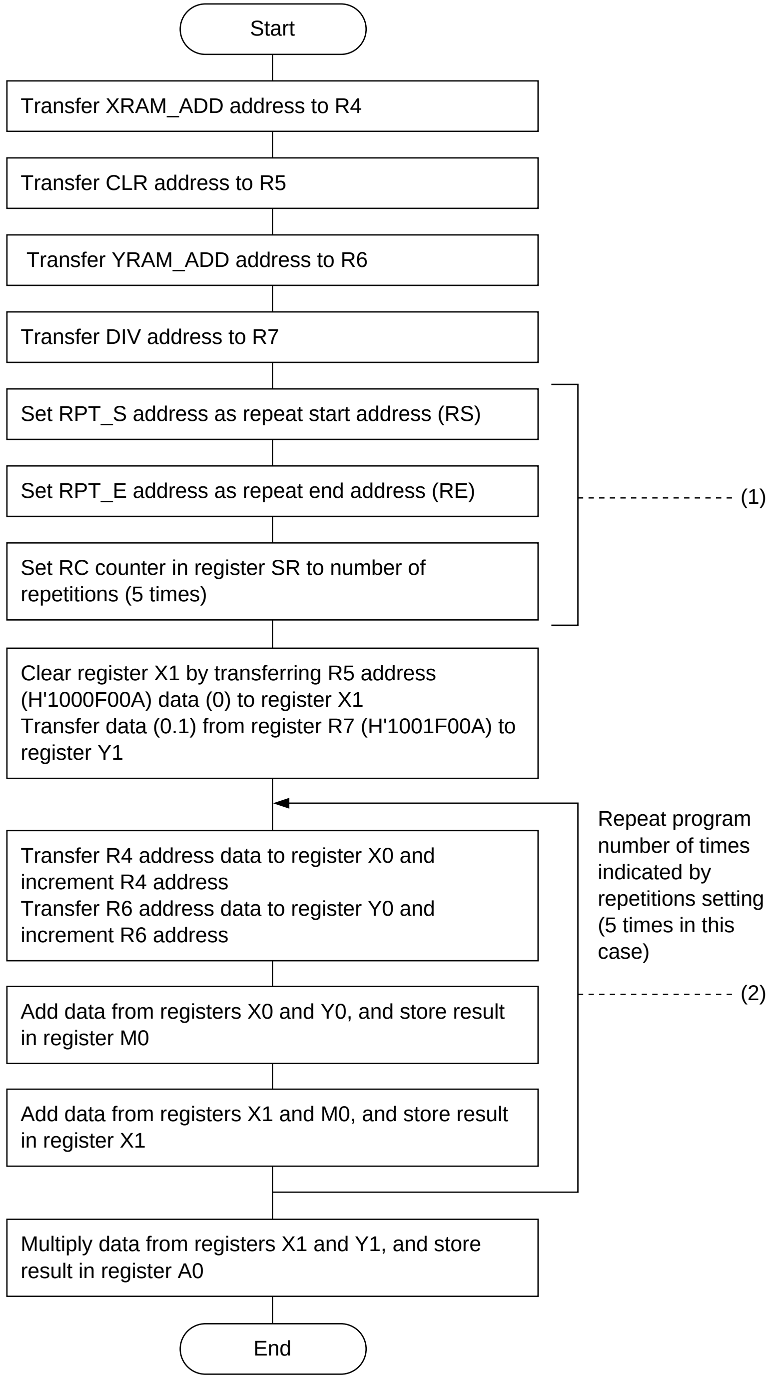
(a) Expanded Image of Repeat Program

```
        REPEAT  RPT_S,RPT_E,#5
RPT_S0:                                             MOVX.W @R5,X1   MOVY.W @R7,Y1
RPT_S:                                              MOVX.W @R4+,X0  MOVY.W @R6+,Y0
RPT_E0: PADD   X0,Y0,M0
RPT_E:  PADD   X1,M0,X1
                        PMULS  X1,Y1,A1
```

(b) Repeat Program Using Extended Instruction REPEAT

RENESAS

Flowchart



Start
Transfer XRAM_ADD address to R4
Transfer CLR address to R5
Transfer YRAM_ADD address to R6
Transfer DIV address to R7
Set RPT_S address as repeat start address (RS)
Set RPT_E address as repeat end address (RE)
Set RC counter in register SR to number of repetitions (5 times)
(1)
Clear register X1 by transferring R5 address (H'1000F00A) data (0) to register X1
Transfer data (0.1) from register R7 (H'1001F00A) to register Y1
Transfer R4 address data to register X0 and increment R4 address
Transfer R6 address data to register Y0 and increment R6 address
Repeat program number of times indicated by repetitions setting (5 times in this case)
(2)
Add data from registers X0 and Y0, and store result in register M0
Add data from registers X1 and M0, and store result in register X1
Multiply data from registers X1 and Y1, and store result in register A0
End

Main Program

(1) Repeat Control Using CPU Instructions

```
;*********************************************************************************************
;*                    Repeat routine
;*********************************************************************************************
MAIN:   MOV.L        #XRAM_ADD,R4
        MOV.L        #CLR,R5
        MOV.L        #YRAM_ADD,R6
        MOV.L        #DIV,R7
        LDRS         RPT_E0                                              ;Repeat start address
        LDRE         RPT_E0                                              ;Repeat end address
        SETRC        #5                                                  ;Repeat counter setting/5
                                                                         repetitions
                               MOVX.W @R5,X1   MOVY.W @R7,Y1   ;Clear X1, Y1 = 1/10
RPT_S:                         MOVX.W @R4+,X0  MOVY.W @R6+,Y0
RPT_E0: PADD  X0,Y0,M0
RPT_E:  PADD  X1,M0,X1                                                   ;X1/data total
                     PMULS  X1,Y1,A1                                     ;A1/average value
EXIT:   BRA   EXIT
        NOP
MAIN_E: NOP
```

(2) Repeat Control Using Extended Instruction REPEAT

```
;*********************************************************************************************
;*                    Repeat routine
;*********************************************************************************************
MAIN:   MOV.L        #XRAM_ADD,R4
        MOV.L        #CLR,R5
        MOV.L        #YRAM_ADD,R6
        MOV.L        #DIV,R7
        MOV.L        #5,R0
        REPEAT RPT_S,RPT_E,R0                                             ;CPU instructions for
                                                                          repeat control generated
                                                                          automatically
                               MOVX.W @R5,X1   MOVY.W @R7,Y1    ;Clear X1, Y1 = 1/10
RPT_S:                         MOVX.W @R4+,X0  MOVY.W @R6+,Y0
        PADD  X0,Y0,M0
RPT_E:  PADD  X1,M0,X1                                                    ;X1/data total
        PMULS X1,Y1,A1                                                    ;A1/average value
EXIT:   BRA   EXIT
        NOP
MAIN_E: NO
```


RENESAS

Data

* Same data used by main programs (1) and (2)

```
;******************************************************************************************
;*         Data (X/YRAM)
;******************************************************************************************

           .SECTION XRAM,CODE,LOCATE=H'1000F000
XRAM_ADD:  .XDATA.W    0.0625,0.125,0.0625,0.0625,0.03125  ;DSP operation data
CLR;       .DATA.W     0                                   ;DSP operation result storage area

           .SECTION YRAM,CODE,LOCATE=H'1001F000
YRAM_ADD:  .XDATA.W    0.0625,0.125,0.03125,0.125,0.0625   ;DSP operation data
DIV:       .XDATA.W    0.1                                 ;DSP operation result storage area
```

Section 6 Examples of Arguments Passed Between CPU Instructions and DSP Instructions

Overview

The two 16-bit fixed-point data values stored at the XRAM_ADD address (H'1000F000) and YRAM_ADD address (H'1001F000) are multiplied using DSP instructions and CPU instructions.

Description

When data is passed between CPU instructions and DSP instructions, R4, R5, R6, and R7 are used as pointers and the data is passed via XRAM and YRAM. The procedure when the result of a calculation performed by the DSP is used by the CPU is described below.

As can be seen in (2-1), (3-1), and (3-2), both the (2) DSP multiplication routine and (3) CPU multiplication routine of the example main program read data stored in XRAM and YRAM.

Example arguments:

```
PADD      X0,Y0,A0    ; Stores result of adding X0 and Y0 in A0
MOVX.W    A0,@R4      ; Transfers A0 data to R4 address
MOV.W     @R4,R0      ; Transfers R4 address data to R0
```

Some points need to be kept in mind when transferring data. Some of the DSP instructions are for handling fixed-point data, and when fixed-point multiplication is performed the result is matched to the MSB. However, when multiplication is performed using CPU instructions, integer multiplication is performed and the is matched to the LSB. This means that the calculation result will differ from that obtained using DSP instructions.

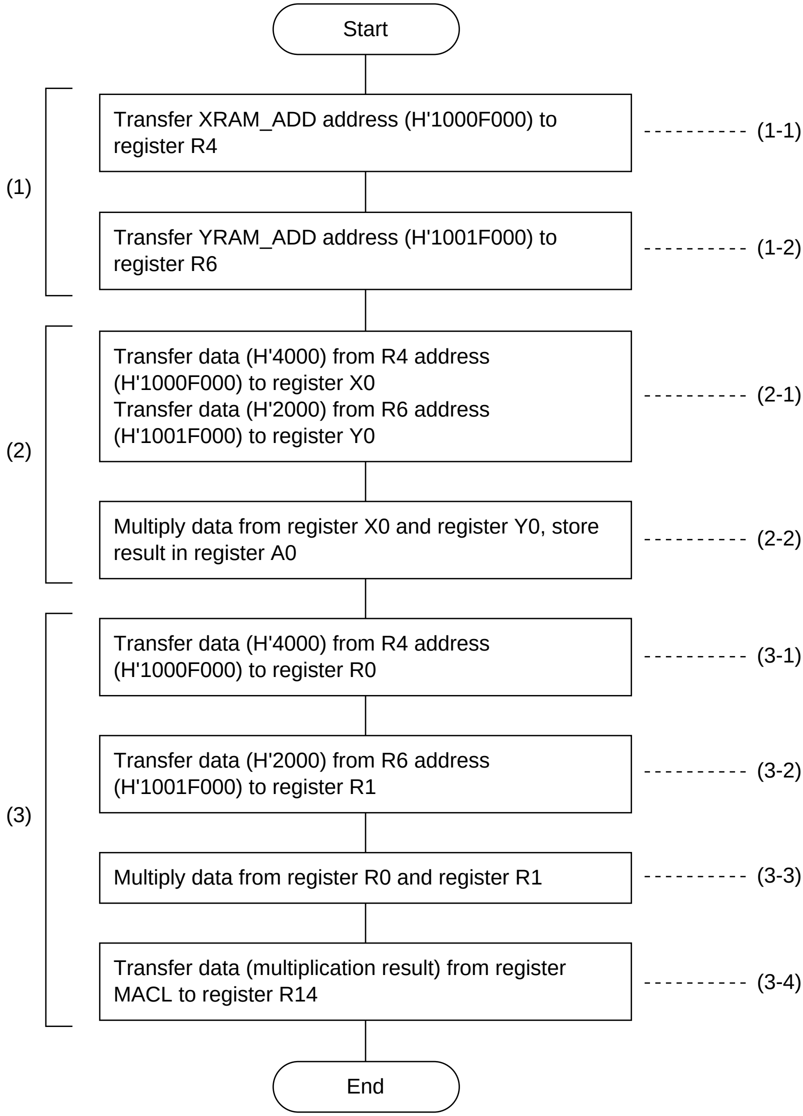
The multiplication process used in (2-1), (3-1), and (3-2) in the (2) DSP multiplication routine and (3) CPU multiplication routine in the flowchart on the following page is shown in table 6.1. This shows that the calculation results after execution differ even if the source operand data is identical. When a DSP instruction (PMULS) is used to multiply integer data, it is necessary to convert the calculation result from fixed-bit data into integer format by performing a bit shift.

Table 6.1 DSP and CPU Multiplication Process

	Excerpt from Main Program		Register Contents
(2) DSP multiplication routine	PMULS	X0,Y0,A0	Before execution: X0=H'4000, Y0=2000 After execution: A0=H'1000 0000
(3) CPU multiplication routine	MULS.W STS	R0,R1 MACL,R14	Before execution: R0=H'4000, R1=H'2000 After execution: R14=H'0800 0000

RENESAS

Flowchart


Start
Transfer XRAM_ADD address (H'1000F000) to register R4
(1-1)
(1)
Transfer YRAM_ADD address (H'1001F000) to register R6
(1-2)
Transfer data (H'4000) from R4 address (H'1000F000) to register X0
Transfer data (H'2000) from R6 address (H'1001F000) to register Y0
(2-1)
(2)
Multiply data from register X0 and register Y0, store result in register A0
(2-2)
Transfer data (H'4000) from R4 address (H'1000F000) to register R0
(3-1)
Transfer data (H'2000) from R6 address (H'1001F000) to register R1
(3-2)
(3)
Multiply data from register R0 and register R1
(3-3)
Transfer data (multiplication result) from register MACL to register R14
(3-4)
End

Main Program

```
;*******************************************************************************************
;*                    Initial setting routine
;*******************************************************************************************
MAIN:   MOV.L          #XRAM_ADD,R4
        MOV.L          #YRAM_ADD,R6

;*******************************************************************************************
;*                    DSP multiplication routine
;*******************************************************************************************
                                  MOVX.W @R4,X0  MOVY.W @R6,Y0 ;Load 0.5,0.25
        PMULS  X0,Y0,A0                                        ;A0 = multiplication result

;*******************************************************************************************
;*                    CPU multiplication routine
;*******************************************************************************************
        MOV.L          @R4,R0                                  ;H'4000 load
        MOV.L          @R6,R1                                  ;H'2000 load
        MULS.W         R0,R1
        STS            MACL,R14                                ;R14 = multiplication result

EXIT:   BRA            EXIT
        NOP
MAIN_E: NOP
```

Data

```
;***********************************************************************
;*          Data
;***********************************************************************
           .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:  .XDATA.W  0.5                            ;DSP operation data

           .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD   .XDATA.W  0.25                           ;DSP operation data
           .END
```


RENESAS

Section 7   32-bit Multiplication

Overview

The 32-bit data value stored at the XRAM_ADD address (H'1000F000) and the 32-bit data value stored at the YRAM_ADD address (H'1001F000) are multiplied, and the result (64-bit) is transferred from the ANS address (H'1001F100) to the ANS+7 address (H'1001F107), where it is stored.

Description

1. Overview of Calculation Method

   The addresses where the multiplier and multiplicand of a 32-bit multiplication operation are stored, and the address where the result is stored, are shown in figure 7.1. Figure 7.2 shows an overview of the calculation method for 32-bit multiplication. The 32-bit data values (the multiplier and multiplicand) are separated into their upper and lower 16-bit segments (here provisionally called A, B, C, and D), which are then multiplied to produce the 64-bit operation result. The top bit (MSB) of the 16-bit data input to the multiplier is interpreted as the sign bit, and it has a weight of $-2^0 = -1$. Therefore, in the example program the first top bit (MSB) is replaced with 0, the product of the various segments is calculated, and a correction items are added using the top bit in order to obtain the 32-bit multiplication result.

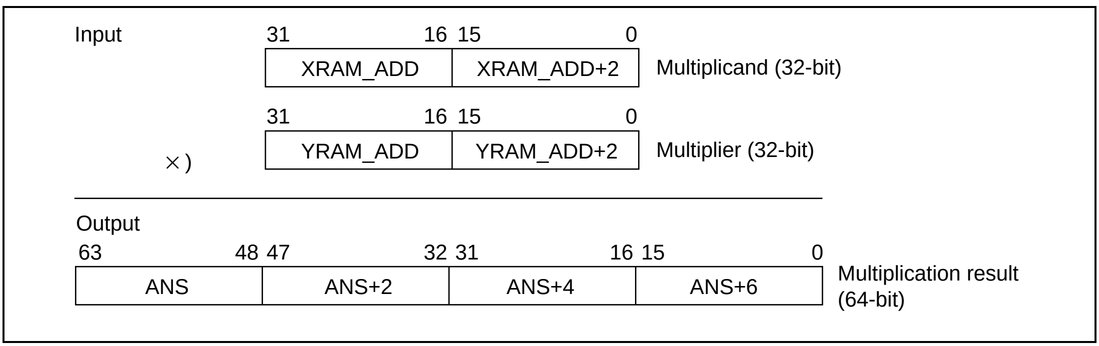


Figure 7.1   32-bit Multiplication

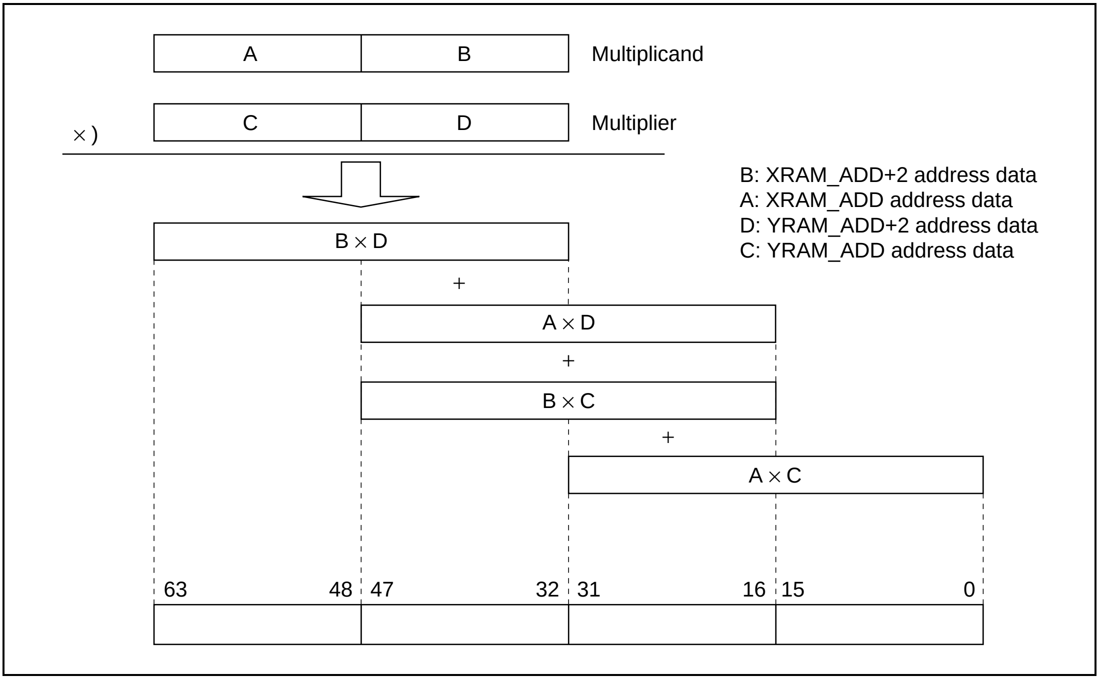


Figure 7.2 Overview of Calculation Method for 32-bit Multiplication

2. Double-length Calculation Algorithm

If the single-precision number of bits is n, “double-length” refers to 2n bits. Therefore, 2n bit numbers can be expressed as shown in figure 7.3.

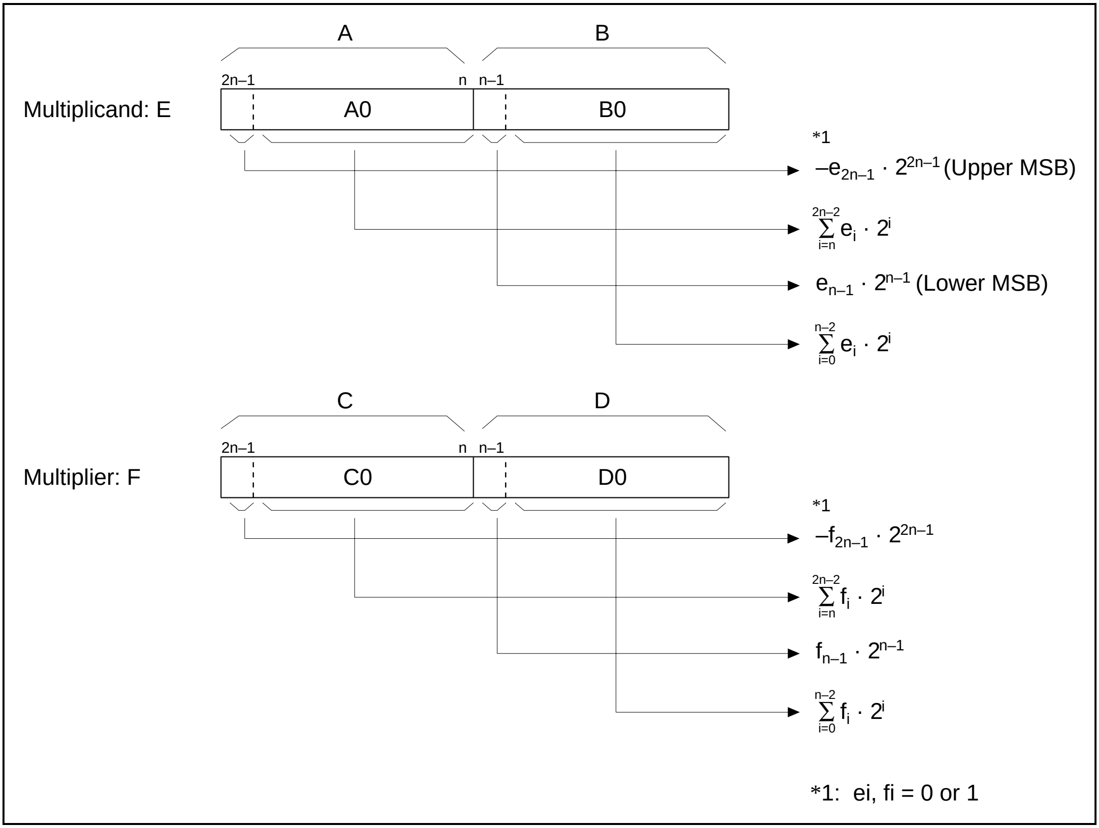


Figure 7.3 Structure of 2n-bit Numbers

Here, if $\Sigma e_i \cdot 2^i = A0$, $\Sigma e_i \cdot 2^i = B0$, $\Sigma e_i \cdot 2^i = C0$, $\Sigma e_i \cdot 2^i = D0$, performing the double-length multiplication $E \times F$ is can be expressed as:

$$
\begin{aligned}
E \times F = \; & (-e_{2n-1} \cdot 2^{2n-1} + A0 + e_{2n-1} \cdot 2^{n-1+} + B0) \times (-f_{2n-1} \cdot 2^{2n-1} + C0 + f_{2n-1} \cdot 2^{n-1+} + D0) \\
= \; & e_{2n-1} \cdot f_{2n-1} \cdot 2^{4n-2} \; \textbf{(1)} \\
& -e_{2n-1} \cdot 2^{2n-1} (C0 + f_{n-1} \cdot 2^{n-1+} + D0) \; \textbf{(2)} \\
& -f_{2n-1} \cdot 2^{2n-1} (A0 + e_{n-1} \cdot 2^{n-1+} + B0) \; \textbf{(3)} \\
& +e_{n-1} \cdot 2^{n-1} (C0 + f_{n-1} \cdot 2^{n-1+} + D0) \; \textbf{(4)} \\
& +f_{n-1} \cdot 2^{n-1} (A0 + B0) \; \textbf{(5)} \\
& +A0 \cdot C0 + A0 \cdot D0 + B0 \cdot C0 + B0 \cdot D0 \textbf{(6)}
\end{aligned}
$$

In the above equation, **(6)** is the product of the segments and **(1)** through **(5)** are correction items.

The correction items involve determining whether the sign bit is “0” or “1” and, if it is “1”, adding it to or deleting it from the product of the segments.

Figure 7.4 shows a 32-bit double-length multiplication algorithm that uses the above equation. The whole can be subdivided into the following six parts:

In part (1), in order to clear the sign bits of A, B, C, and D to 0, the logical product with H'7FFF is obtained, resulting in A0, B0, C0, and D0. In part (2), the product is calculated for the following four segments: $A0 \cdot C0$, $A0 \cdot D0$, $B0 \cdot C0$, and $D0 \cdot C0$. In parts (3) through (6), the sum is obtained for each digit, and the results are stored at the ANS, ANS+2, ANS+4, and ANS+6 addresses.

RENESAS

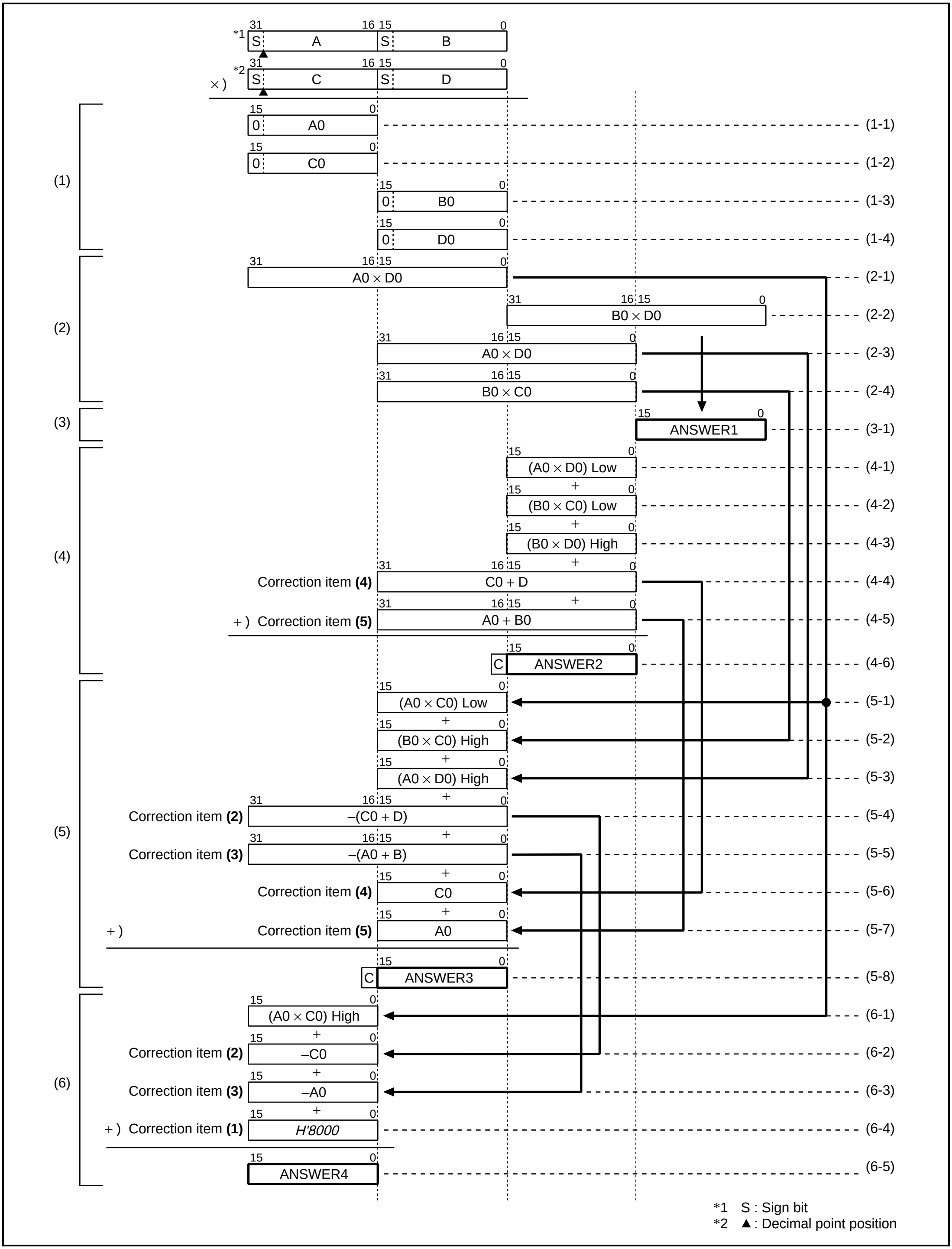
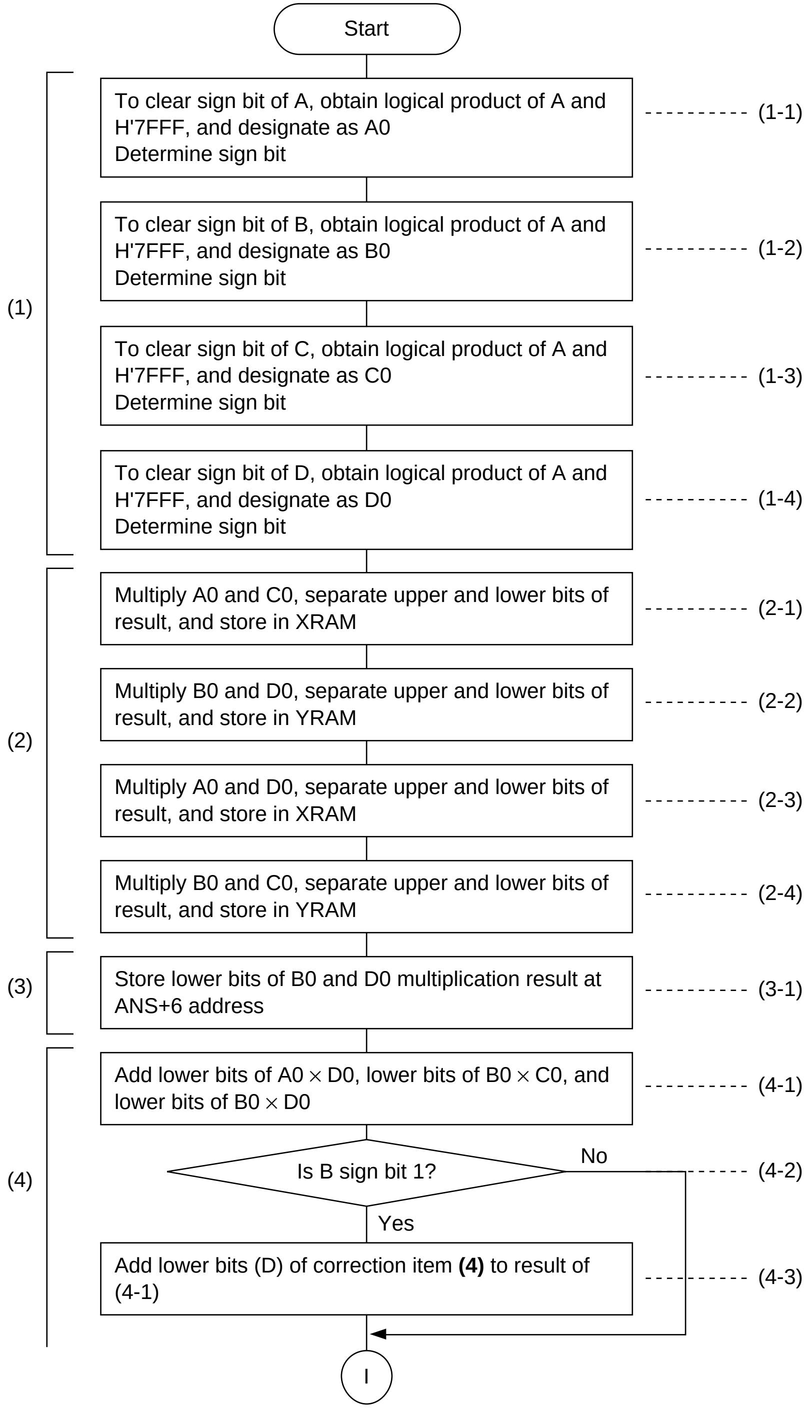
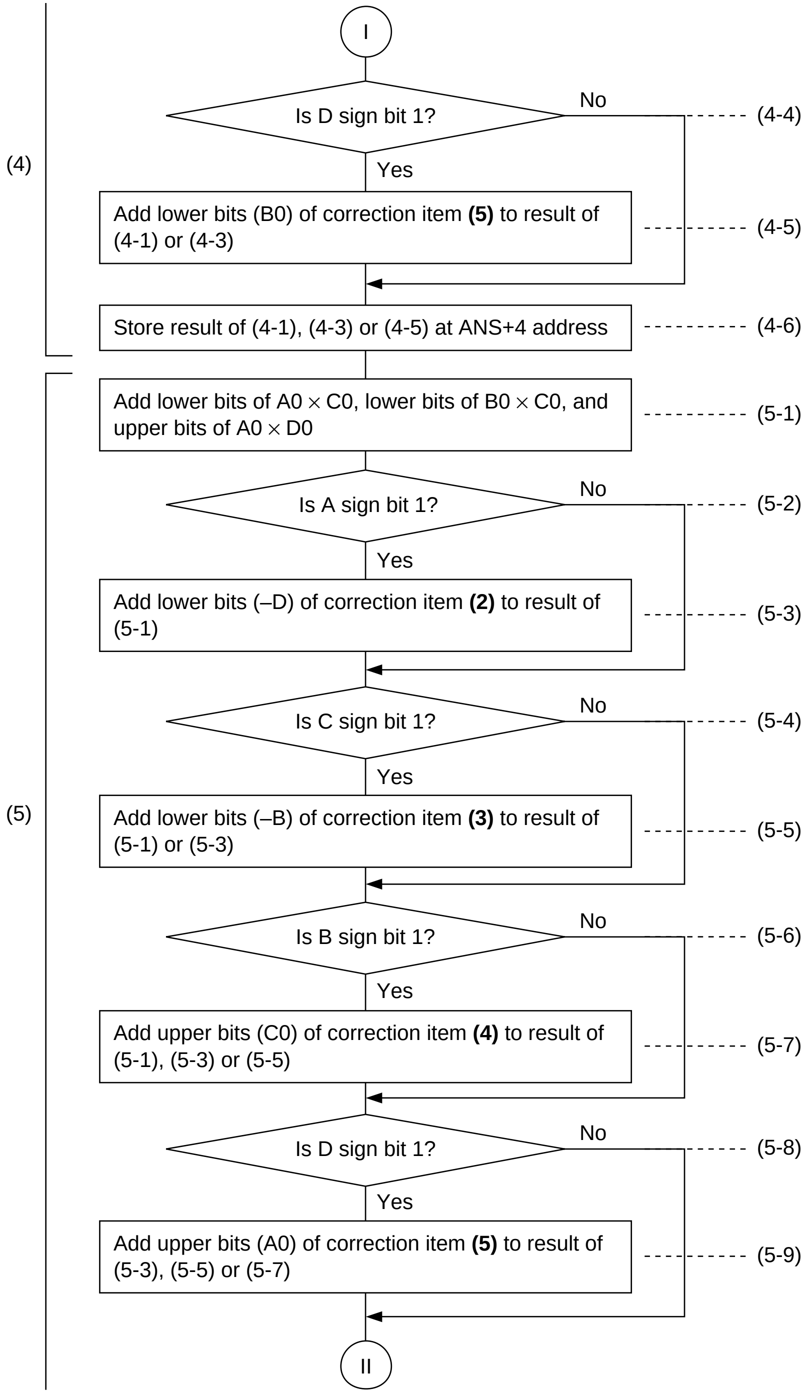


Figure 7.4 32-bit Double-length Multiplication Algorithm

Flowchart

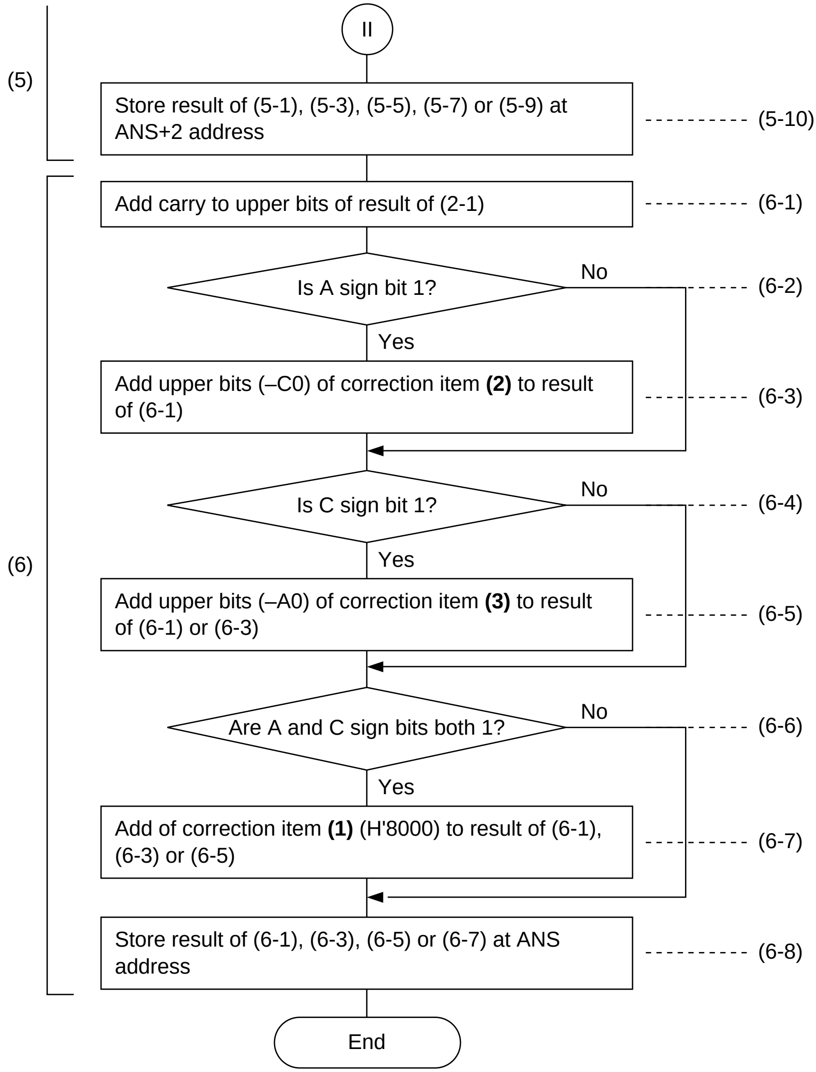


RENESAS



I
Is D sign bit 1?
No
(4-4)
Yes
Add lower bits (B0) of correction item (5) to result of (4-1) or (4-3)
(4-5)
Store result of (4-1), (4-3) or (4-5) at ANS+4 address
(4-6)
(4)
Add lower bits of A0 × C0, lower bits of B0 × C0, and upper bits of A0 × D0
(5-1)
Is A sign bit 1?
No
(5-2)
Yes
Add lower bits (–D) of correction item (2) to result of (5-1)
(5-3)
Is C sign bit 1?
No
(5-4)
Yes
Add lower bits (–B) of correction item (3) to result of (5-1) or (5-3)
(5-5)
Is B sign bit 1?
No
(5-6)
Yes
Add upper bits (C0) of correction item (4) to result of (5-1), (5-3) or (5-5)
(5-7)
Is D sign bit 1?
No
(5-8)
Yes
Add upper bits (A0) of correction item (5) to result of (5-3), (5-5) or (5-7)
(5-9)
(5)
II

RENESAS


II
(5)
Store result of (5-1), (5-3), (5-5), (5-7) or (5-9) at ANS+2 address
(5-10)
Add carry to upper bits of result of (2-1)
(6-1)
Is A sign bit 1?
No
(6-2)
Yes
Add upper bits (–C0) of correction item **(2)** to result of (6-1)
(6-3)
Is C sign bit 1?
No
(6-4)
(6)
Yes
Add upper bits (–A0) of correction item **(3)** to result of (6-1) or (6-3)
(6-5)
Are A and C sign bits both 1?
No
(6-6)
Yes
Add of correction item **(1)** (H'8000) to result of (6-1), (6-3) or (6-5)
(6-7)
Store result of (6-1), (6-3), (6-5) or (6-7) at ANS address
(6-8)
End

Main Program

```
;*******************************************************************************************
;*                    32-bit fixed-point multiplication routine
;*
;*                    [A][B] × [C][D]
;*
;*******************************************************************************************
MAIN: MOV.L   #XRAM_ADD,R4
      MOV.L   #WORKX,R5                                         ;XRAM for work
      MOV.L   #YRAM_ADD,R6
      MOV.L   #WORKY,R7                                         ;YRAM for work
;Clear sign
      MOV.W           #H'7FFF,R0
      MOV.W           R0,@R7
      PCLR    A1                MOVX.W @R4+,X0  MOVY.W @R7,Y0  ;A,H'7FFF load
      PAND    X0,Y0,A0                          MOVY.W @R6+,Y1 ;A0,C load
      MOV.W           R0,@R5                                   ;H'7FFF -> #WORKX
      PSHA    #1,X0             MOVX.W @R5,X1                  ;A sign chech,H'7FFF load
  DCT  PINC   A1,A1             MOVX.W A0,@R5+                 ;A0 store
      PAND    X1,Y1,A0          MOVX.W @R4,X0                  ;C0,B load
      MOV.L           R4,@-R15
      MOV.L           #SIGNA,R4
      PCLR    A1                MOVX.W A1,@R4+
      PSHA    #1,Y1                             MOVY.W A0,@R7+ ;C sign check,C0 store
  DCT PINC    A1,A1                             MOVY.W @R6,Y1  ;B sign check,D load
      PAND    X0,Y0,A0          MOVX.W A1,@R4+                 ;B0
      PCLR    A1
      PSHA    #1,X0             MOVX.W A0,@R5
  DCT PINC    A1,A1
      PAND    X1,Y1,A0          MOVX.W A1,@R4+                 ;D0,B0 store
      PCLR    A1
      PSHA    #1,Y1
  DCT PINC    A1,A1                             MOVY.W A0,@R7  ;D0 store
                                MOVX.W A1,@R4
      MOV.L           @R15+,R4

;****************************************************************
;*Segment product calculation routine/   B0×D0,A0×C0,B0×C0,A0×D0
;****************************************************************
      MOV.L           #WORKX,R5
      MOV.L           #WORKY,R7
                                MOVX.W @R5+,X0  MOVY.W @R7+,Y0 ;A0,C0
      PMULS   X0,Y0,A1          MOVX.W @R5+,X1  MOVY.W @R7+,Y1 ;A0×C0,B0,D0
      PMULS   X1,Y1,A0          MOVX.W A1,@R5+                 ;B0×D0, (A0×C0)H store
      PSHA    #16,A1                            MOVY.W A0,@R7+ ;(A0×C0)L, (B0×D0)H store
```

```
        PSHA   #16,A0          MOVX.W A1,@R5+                   ;(B0×D0)L, (A0×C0)L store
        PMULS  X0,Y1,A1                       MOVY.W A0,@R7+;A0×D0, (B0×D0)L store
        PSHA   #16,A1          MOVX.W A1,@R5+                   ;(A0×D0)L, (A0×D0)H store
        PMULS  X1,Y0,A1        MOVX.W A1,@R5                    ;B0×C0, (A0×D0)L store
        PSHA   #16,A1                         MOVY.W A1,@R7+;(B0×C0)L, (B0×C0)H store
                                              MOVY.W A1,@R7 ;(B0×C0)L store

;******************
;*ANSWER1 STORE
;******************
        MOV.L          R7,@-R15                              ;push R7
        MOV.L          #ANS,R7
        ADD            #6,R7
                                              MOVY.W A0,@R7+;Store in ANS1
        ADD            #-2,R7
        MOV.L          R7,R14                                ;R14=#ANS+2
        MOV.L          @R15+,R7                              ;pop R7

*******************************************************************************************
;*2-word calculation routine/   R4=#XRAM_ADD+2,R5=#WORKX+10,R6=#YRAM_ADD+2,R7=#WORKY+10
;******************************************************************************************
        PCOPY  X1,M1
        MOV.L  #-6,R9
        PCLR   A1              MOVX.W @R5,X1  MOVY.W @R7+R9,Y1 ;(A0×D0)L lode,
                                                                (B0×C0)L load
        PADD   X1,Y1,A0                       MOVY.W @R7+,Y1   ;(A0×D0)L+(B0×C0)L,
                                                                (B0×D0)H load
    DCT PINC   A1,A1                                           ;carry check
        PADD   A0,Y1,A0                                        ;(A0×D0)L+(B0×C0)
                                                                L+(B0×D0)H
    DCT PINC   A1,A1                                           ;carry check
        MOV.W          #H'0,R10
        MOV.L          #SIGND,R0
        MOV.W          @R0+,R1
        CMP/EQ         R10,R1                                  ;Is B negative?
        BT             HOSEI4_L
                                              MOVY.W @R6,Y1    ;Load D
        PADD   A0,Y1,A0                                        ;Add D
    DCT PINC   A1,A1
HOSEI4_L:
        MOV.W          @R0,R1
        CMP/EQ         R10,R1                                  ;Is D negative?
        BT             HOSEI5_L
        PADD   A0,M1,A0                                        ;Add B0
    DCT PINC   A1,A1
HOSEI5_L:
        MOV.L          R4,@-R15                                ;push R4
```

```
      MOV.L          #CARRY,R4
                               MOVX.W A1,@R4                  ;carry store
      MOV.L          @R15+,R4                                 ;pop R4
;******************
;*ANSWER2 STORE
;******************
      MOV.L          R7,@-R15                                 ;push R7
      MOV.L          R14,R7
                                              MOVY.W A0,@R7+   ;ANS2 store
      ADD            #-2,R7
      MOV.L          R7,R14                                   ;R14=#ANS+4
      MOV.L          @R15+,R7                                 ;pop R7
;*****************************************************************************************
;*3-word calculation routine/   R4=#XRAM_ADD+2,R5=#WORKX+10,R6=#YRAM_ADD+2,R7=#WORKY+6
;*****************************************************************************************
      MOV.L  #-4,R8
      PCOPY  X0,A1             MOVX.W @R5+R8,X0 MOVY.W @R7+,Y1 ;dummy load
                               MOVX.W @R5+,X0   MOVY.W @R7+,Y1 ;(A0×C0)L lode,
                                                                (B0×C0)H load
      PADD   X0,Y1,M1          MOVX.W @R5,X1                  ;(A0×C0)L+(B0×C0)H,
                                                                (A0×D0)H load
  DCT PINC   M0,M0                                            ;carry check
      PADD   X1,M1,A0                                         ;(A0×C0)L+(B0×C0)
                                                               H+(A0×D0)H
  DCT PINC   M0,M0                                            ;carry check
;Correction
      MOV.W          #H'0,R10
      MOV.L          #SIGNA,R0
      MOV.W          @R0+,R1
      CMP/EQ         R10,R1                                   ;Is A negative?
      BT             HOSEI2_L
      PSUB   A0,Y1,A0                                         ;Subtract D (correction 2)
  DCT PDEC   M0,M0
HOSEI2_L:
      MOV.W          @R0+,R1
      CMP/EQ         R10,R1                                   ;Is C negative?
      BT             HOSEI3_L
                               MOVX.W @R4,X1
      PCOPY  X1,M1
      PSUB   A0,M1,A0                                         ;Subtract B (correction 3)
  DCT PDEC   M0,M0

HOSEI3_L:
      MOV.W          @R0+,R1
      CMP/EQ         R10,R1                                   ;Is B negative?
      BT             HOSEI4_H
      PADD   A0,Y0,A0                                         ;Subtract C0 (correction 4)
```


RENESAS

```
    DCT PINC   M0,M0
  HOSEI4_H:
        MOV.W          @R0+,R1
        CMP/EQ         R10,R1                                ;Is D negative?
        BT             HOSEI5_H
        PCOPY  A1,M1
        PADD   A0,M1,A0                                      ;Add A0 (correction 5)
    DCT PINC   M0,M0
  HOSEI5_H:
        PCOPY  A0,M1
        MOV.L          #CARRY,R4
                                             MOVX.W @R4,X1   ;Load carry
        PADD   X1,M1,A0                                      ;Add carry
    DCT PINC   M0,M0                                         ;Check carry

  ;**************
  ;*ANSWER3 STORE
  ;**************
        MOV.L          R14,R7
                                             MOVY.W A0,@R7+;ANS3 store
        ADD            #-2,R7

  ;******************************************************************************************
  ;*4-word calculation routine/   R4=#XRAM_ADD+2,R5=#WORKX+8,R6=#YRAM_ADD+2,R7=#WORKY+10
  ;******************************************************************************************
        PCLR   Y1              MOVX.W @R5+R8,X1             ;dummy load
        PCLR   M1              MOVX.W @R5,X1                ;(A0×C0)H load
        PADD   X1,M0,A0
    DCT PINC   M1,M1
  ;Correction
        MOV.L          #SIGNA,R0
        MOV.W          @R0+,R1
        CMP/EQ         R10,R1                               ;Is A negative?
        BT             HOSEI3_H
        PCOPY  A1,M0
        PSUB   A0,M0,A0                                     ;Subtract C0 (correction 2)
    DCT PDEC   M1,M1
        MOV.L          #H'0,R12
        ADD            #1,R12
  HOSEI2_H:
        MOV.W          @R0+,R1
        CMP/EQ         R10,R1                               ;Is C negative?
        BT             HOSEI4_H
        PSUB   A0,Y0,A0                                     ;Subtract A0 (correction 3)
    DCT PDEC   M1,M1
        ADD            #1,R12
  HOSEI3_H:
```

```
        MOV.L           #2,R1
        CMP/EQ          R1,R12                                  ;Are both A and C negative?
        BF              FIN
        MOV.W           #H'8000,R10
        MOV.W           R10,@R5
                                  MOVX.W @R5,X0
        PCOPY  X0,M1                                            ;Add H'8000 (correction 1)
        PADD   A0,M1,A0

;**************
;*ANSWER4 STORE
;**************
FIN:                                        MOVY.W A0,@R7  ;ANS4 store
EXIT:  BRA              EXIT
       NOP
MAIN_E:       NOP
```

Data

```
;*********************************************************************************************
;*            32-bit multiplication data (XRAM/YRAM)
;*********************************************************************************************
             .SECTION XRAM,DATA,LOCATE=H'1000F000
XRAM_ADD:    .XDATA.L        0.25002500  ;Multiplicand
WORKX:       .RES.W          6           ;Work area
CARRY:       .RES.W          1           ;Carry area
SIGNA:       .RES.W          1           ;For determining sign of multiplicand upper word A
SIGNC:       .RES.W          1           ;For determining sign of multiplier upper word C
SIGNB:       .RES.W          1           ;For determining sign of multiplicand lower word B
SIGND:       .RES.W          1           ;For determining sign of multiplier lower word D

             .SECTION YRAM,DATA,LOCATE=H'1001F000
YRAM_ADD:    .XDATA.L        0.50005000  ;Multiplier
WORKY:       .RES.W          6           ;Work area
ANS:         .RES.W          4           ;Multiplication result storage area
```

Section 8 Trigonometric Functions

Overview

Calculating the trigonometric functions SIN(X) and COS(X).

Description

1. Performing Trigonometric Functions

   Figure 8.1 shows curves for SIN(X) and COS(X). If the angle range is $-\pi \le X \le \pi$, the relationships expressed in equation (1) exists.

$$\left.\begin{aligned} SIN(-X) &= -SIN(X) \\ COS(-X) &= COS(X) \end{aligned}\right\} \text{-------------------------} \quad (1)$$

   Using the relationships expressed in equation (1), the SIN(X) and COS(X) of $-\pi \le X \le 0$ can be calculated by obtaining the SIN(X) and COS(X) of $0 \le X \le \pi$ and processing the sign.

   Next is figure 8.2 (a) and (b). The relationships of SIN(X) and COS(X), with $X = \pi/2$ at the center, are expressed in equation (2).

$$\left.\begin{aligned} SIN(X + \pi/2) &= -SIN(\pi/2 - X) \\ COS(X + \pi/2) &= COS(\pi/2 - X) \end{aligned}\right\} \text{-------------------------} \quad (2)$$

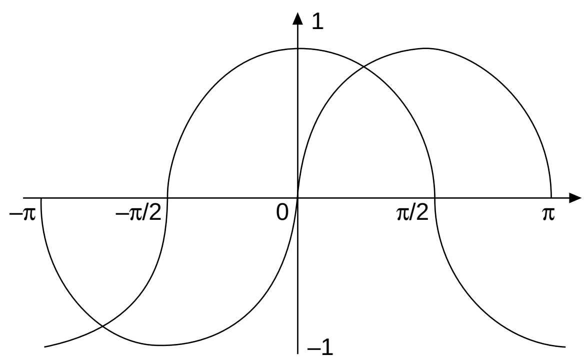


Figure 8.1 SIN(X) and COS(X) Curves

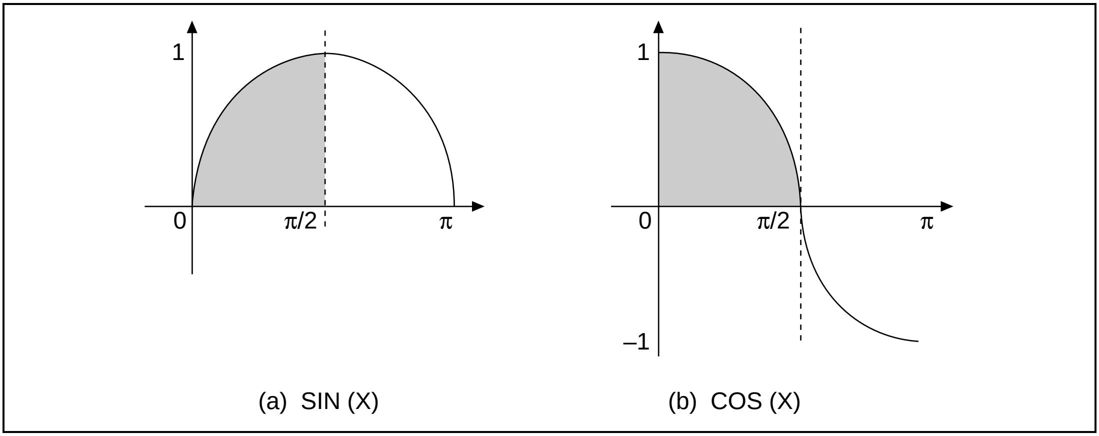


Figure 8.2 SIN(X) and COS(X) Curves with X = π/2 at Center

Based on the relationship between equations (1) and (2), the SIN(X) and COS(X) of $-\pi \le X \le \pi$ can be calculated by obtaining the SIN(X) and COS(X) of $0 \le X \le \pi$ and, finally, processing the sign. The example program divides $0 \le X \le \pi/2$ into 128 segments. If $X = n \cdot \pi/256 + \Delta X$ (n = 1, 2, ..., 128), the result is equation (3), based on the addition theorem of trigonometric functions.

$$\left.\begin{aligned} \mathrm{SIN}(X) &= \mathrm{SIN}(n \cdot \pi/256 + \Delta X) \\ &= \mathrm{SIN}(n \cdot \pi/256) \cdot \mathrm{COS}(\Delta X) - \mathrm{COS}(n \cdot \pi/256) \cdot \mathrm{SIN}(\Delta X) \\ \mathrm{COS}(X) &= \mathrm{COS}(n \cdot \pi/256 + \Delta X) \\ &= \mathrm{COS}(n \cdot \pi/256) \cdot \mathrm{COS}(\Delta X) - \mathrm{SIN}(n \cdot \pi/256) \cdot \mathrm{SIN}(\Delta X) \end{aligned}\right\} \text{------------} (3)$$

If we assume that in equation (3) ΔX is extremely small and approximate that $\mathrm{SIN}(\Delta X) = \Delta X$ and $\mathrm{COS}(\Delta X) = 1 - (\Delta X)^2/2$, the result is equation (4).

$$\left.\begin{aligned} \mathrm{SIN}(X) &= \mathrm{SIN}(n \cdot \pi/256) \cdot \{1 - (\Delta X)^2/2\} + \Delta X \cdot \mathrm{COS}(n \cdot \pi/256) \\ \mathrm{COS}(X) &= \mathrm{COS}(n \cdot \pi/256) \cdot \{1 - (\Delta X)^2/2\} - \Delta X \cdot \mathrm{SIN}(n \cdot \pi/256) \end{aligned}\right\} \text{---------------} (4)$$

In other words, by calculating equation (4) using ΔX and table data $(n \cdot \pi/256)$, we can obtain the SIN(X) and COS(X) of $0 \le X \le \pi/2$. The final result is then obtained by performing sign processing.

2. Converting Input Values

Using conversion equation (5), the example program inputs to the DSP as angle parameters the input value X for the range $-\pi \le X \le \pi$ and a for the range $-1 \le X < 1$.

$$\left.\begin{array}{l} X = \pi \cdot a \\ a = X/\pi \end{array}\right\} \text{---} \quad (5)$$

X unit: rad
a unit: rad/π

Table 8.1 Relation Between Input Value a and Polarity

Input Value	Result SIN(X)	COS(X)	|a|
$-1 < \le a < -0.5$ ($-\pi \le X < -\pi/2$)	Negative	Negative	$\lvert a \rvert > 0.5$
$-0.5 \le a < 0$ ($-\pi/2 \le X < 0$)	Negative	Positive	$\lvert a \rvert \le 0.5$
$0 \le a \le 0.5$ ($0 \le X \le \pi/2$)	Positive	Positive	$\lvert a \rvert \le 0.5$
$0.5 < a < 1$ ($\pi/2 < X < \pi$)	Positive	Negative	$\lvert a \rvert > 0.5$

Here the range $0 \le X \le \pi/2$ corresponds to the range $0 \le X \le 0.5$. Also, the input value a is converted from the range $-1 < a \le 1$ to the range $0 \le a' \le 0.5$. Figure 8.3 shows the curves | SIN(X) | and | COS(X) |.

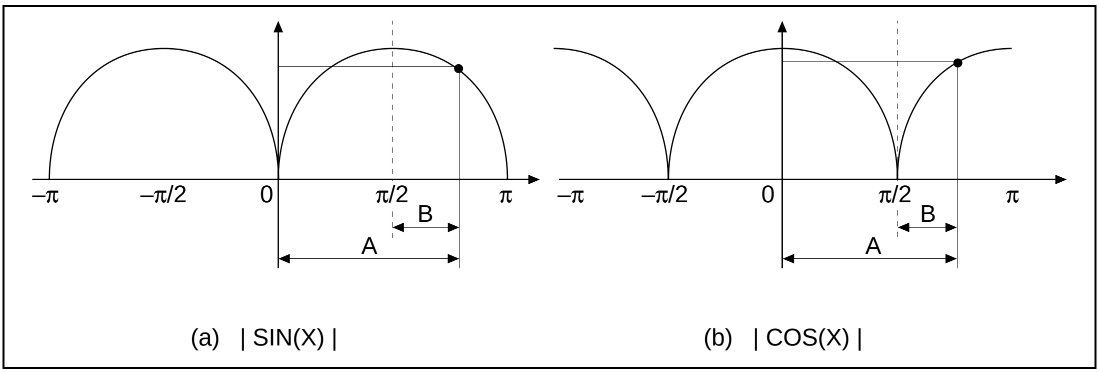


Figure 8.3 Curves | SIN(X) | and | COS(X) |

When obtaining the SIN(X) and COS(X) of point A in figure 8.3, if we assume that A = π/2 + B, then a = 0.5 + b. Therefore, it is possible to obtain the deviation | b | relative to X = π/2 using equation (6).

$$|b| = ||a| - 0.5| \quad \text{-----------------------} \quad (6)$$

Next, based on deviation | b |, equation (7) is used to calculate the conversion of input value a for the range –1 < a ≤ 1 to a' for the range 0 ≤ a' ≤ 0.5.

$$a' = |||a| - 0.5| - 0.5| \quad \text{-----------------------} \quad (7)$$

3. a' Table Data

The example program uses a table with 128 cells. In other words, the range 0 ≤ a' ≤ 0.5 is divided into 128 equal segments. The difference in a' due to the angle of each segment is expressed in equation (8).

$$0.5/128 = 0.00390625 \quad \text{-----------------------} \quad (8)$$

Table 8.2 shows the correspondence between table address n and a' in decimal notation and as 16-bit fixed-point expressions.

Table 8.2 Relationship Between Table Address n and a'

Table Address n	a'																
	n/256; Decimal Notation rad]/π	16-bit Fixed-point Expression															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0.00000000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0.00390625	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
2	0.00781250	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0.01171875	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0.01562500	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
⋮	⋮								⋮								
127	0.49609375	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
128	0.50000000	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▲ : Decimal point position

4. Method of Calculating ΔX

As shown in table 8.2, the upper nine bits of the a' data expressed in fixed-point format correspond to n, and the lower seven bits to the amount of shift from the table data Δa'. Figure 8.4 shows the bit structure of a'. By obtaining the value of a', it is possible to calculate the equation (2) table data address (the value of $n \cdot \pi/256$) as well as ΔX at the same time. Finally, table 8.1 is used for sign processing in order to obtain the SIN(X) and COS(X) of $-\pi \le X \le \pi$.

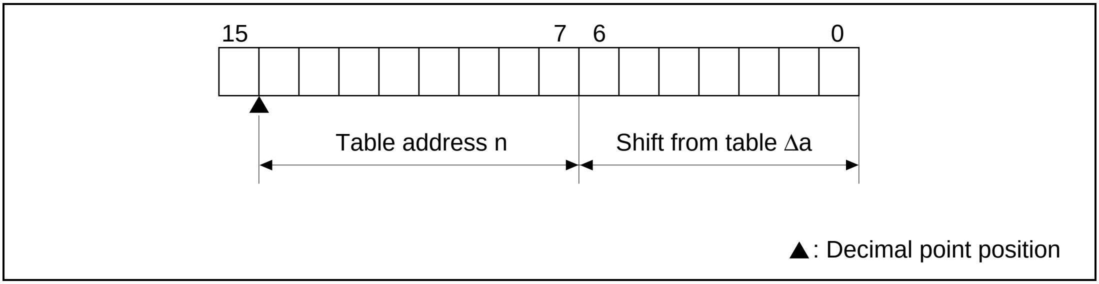


Figure 8.4 Bit Structure of a'

Figure 8.5 shows the relationship with the amount of shift between table values ΔX. Table shift ΔX can also be obtained by using the Δa of a' and equation (9).

$$\Delta X = \Delta a \cdot \pi \quad \text{------------------------------} \quad (9)$$

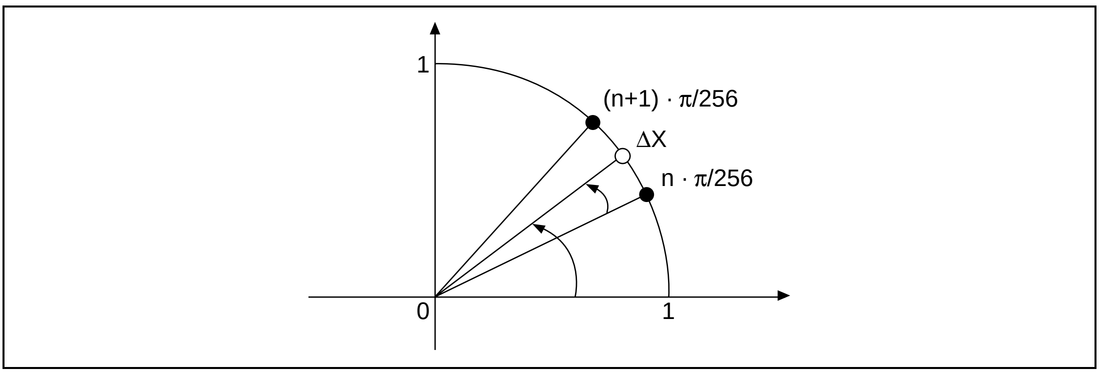


Figure 8.5 Relation With Amount of Shift Between Table Values

5. Overflow Processing

If the calculation result is as shown in equation (10), an overflow occurs.

$$\left.\begin{array}{l} |\,\mathrm{SIN}(X)\,| \;\geq\; 1 \\ |\,\mathrm{COS}(X)\,| \;<\; 0 \end{array}\right\} \text{------------------------------------} \quad (10)$$

In such cases the value is corrected using equation (11).

$$\left.\begin{array}{l} |\,\mathrm{SIN}(X)\,| \;=\; 1 - 2^{-15} \\ |\,\mathrm{COS}(X)\,| \;=\; 0 \end{array}\right\} \text{------------------------------------} \quad (11)$$

6. Algorithm for Calculating Trigonometric Functions

The algorithm for calculating trigonometric functions is as follows.

(1) Make initial settings.

(2) Load input value a, calculate | | | a | –0.5 | –0.5 | to obtain a'.

(3) Obtain logical product of above and #H'FF80 and calculate upper nine bits (n/256) of a'. Then calculate n and set value in Y bus index register (R9).

(4) Obtain logical product of above and #H'007F and calculate lower seven bits (Δa') of a'.

(5) Calculate $\pi\Delta$a'; calculate ΔX.

(6) Calculate $1 - (\Delta X)^2/2$. Load $\sin(n \times \pi/256)$ and $\cos(n \times \pi/256)$ from data table in YRAM.

(7) Calculate sin(X).

(8) Process sign of sin(X); store sin(X).

(9) Calculate cos(X).

(10) Process sign of cos(X); store cos(X).

RENESAS

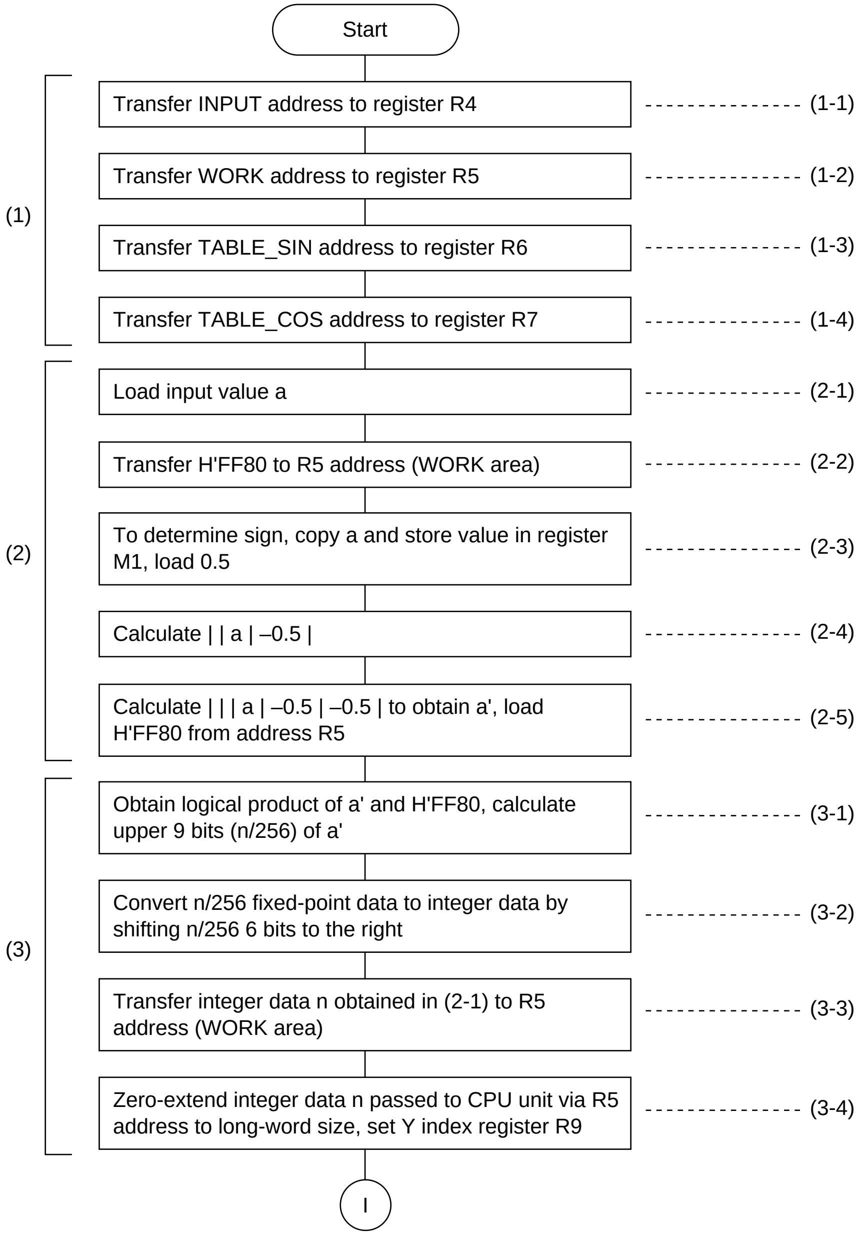
Execution Example

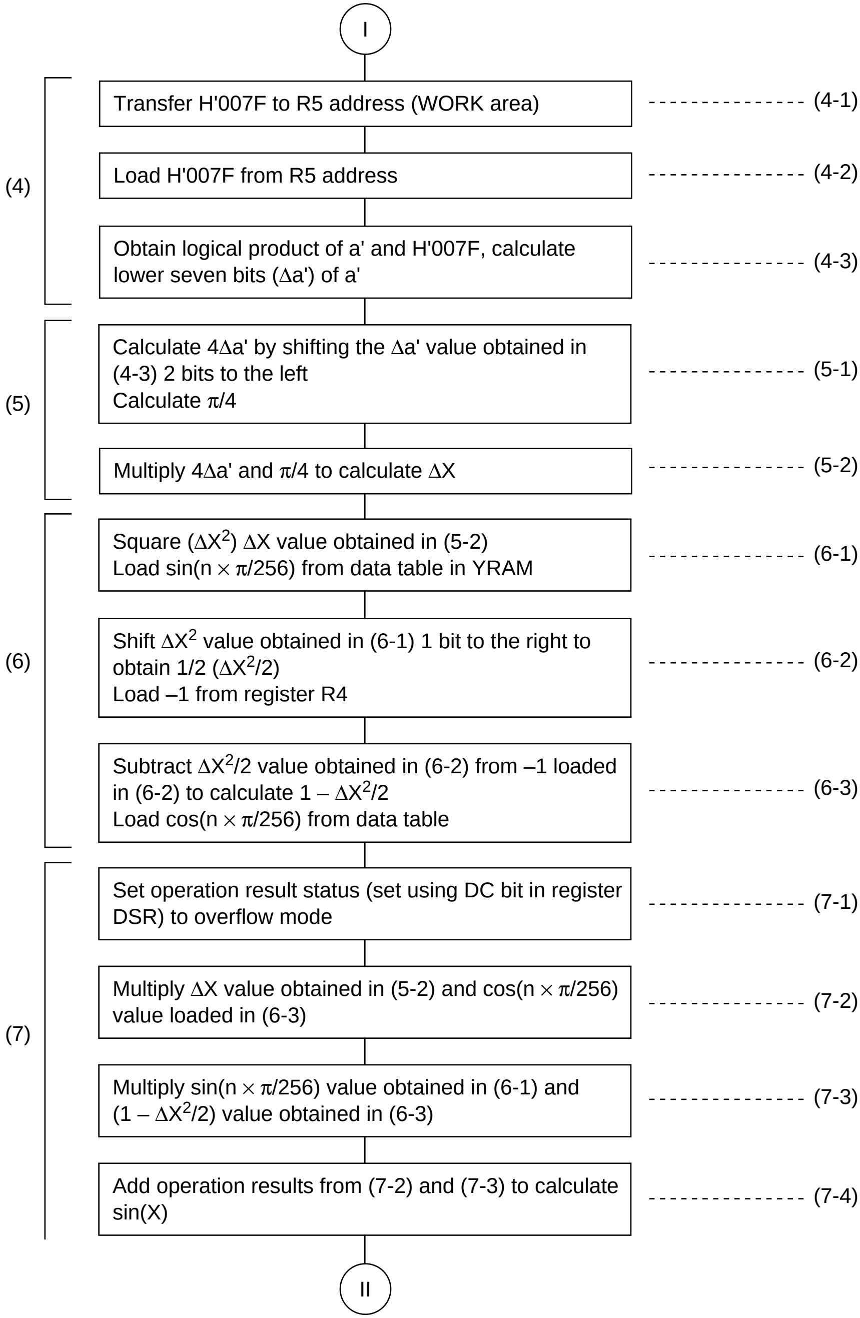
The sin(X) and cos(X) (OUTPUT) calculation results obtained based on the input value a (INPUT) are shown in table 8.3.

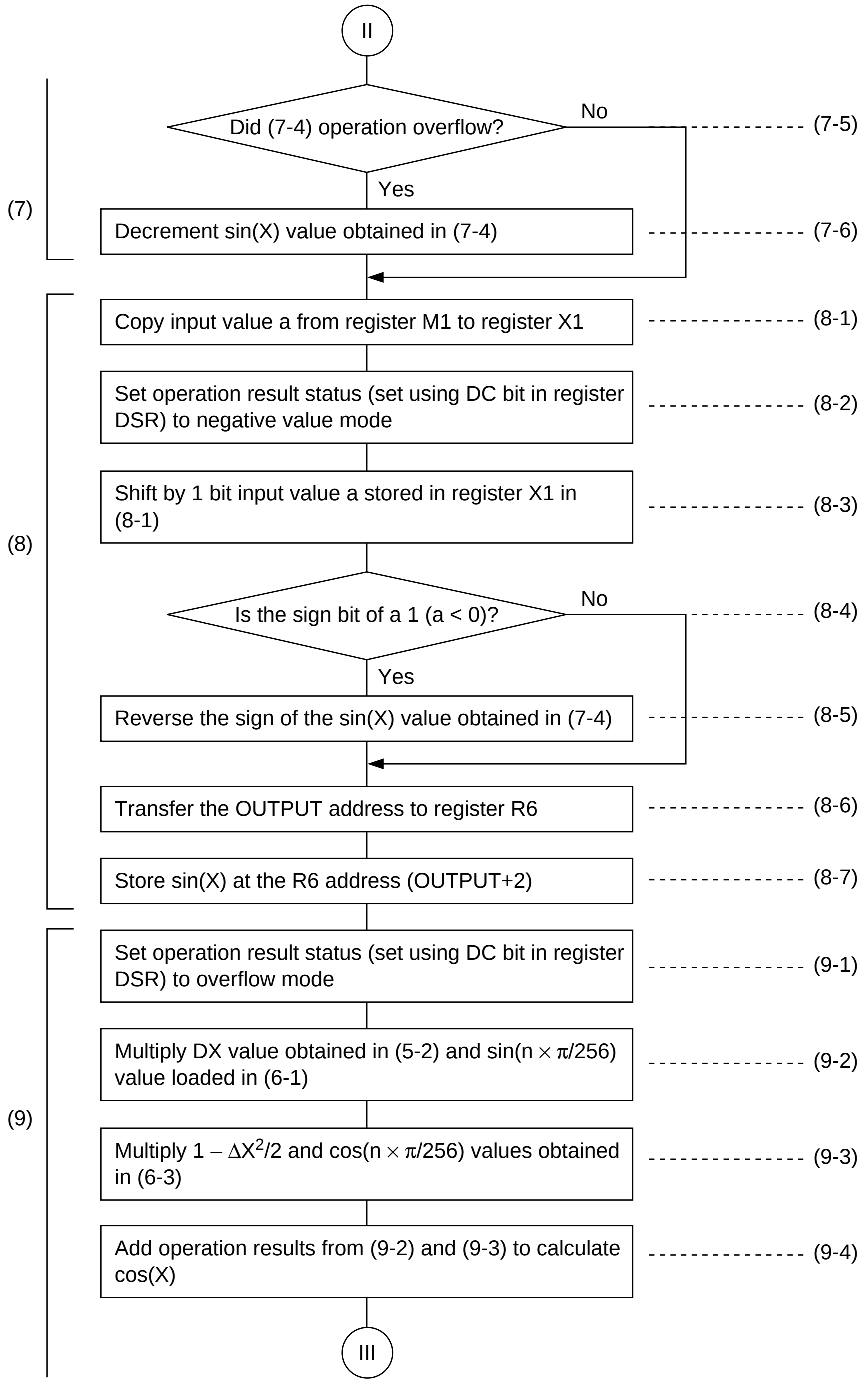
Table 8.3 sin(x), cos(X) Calculation Results

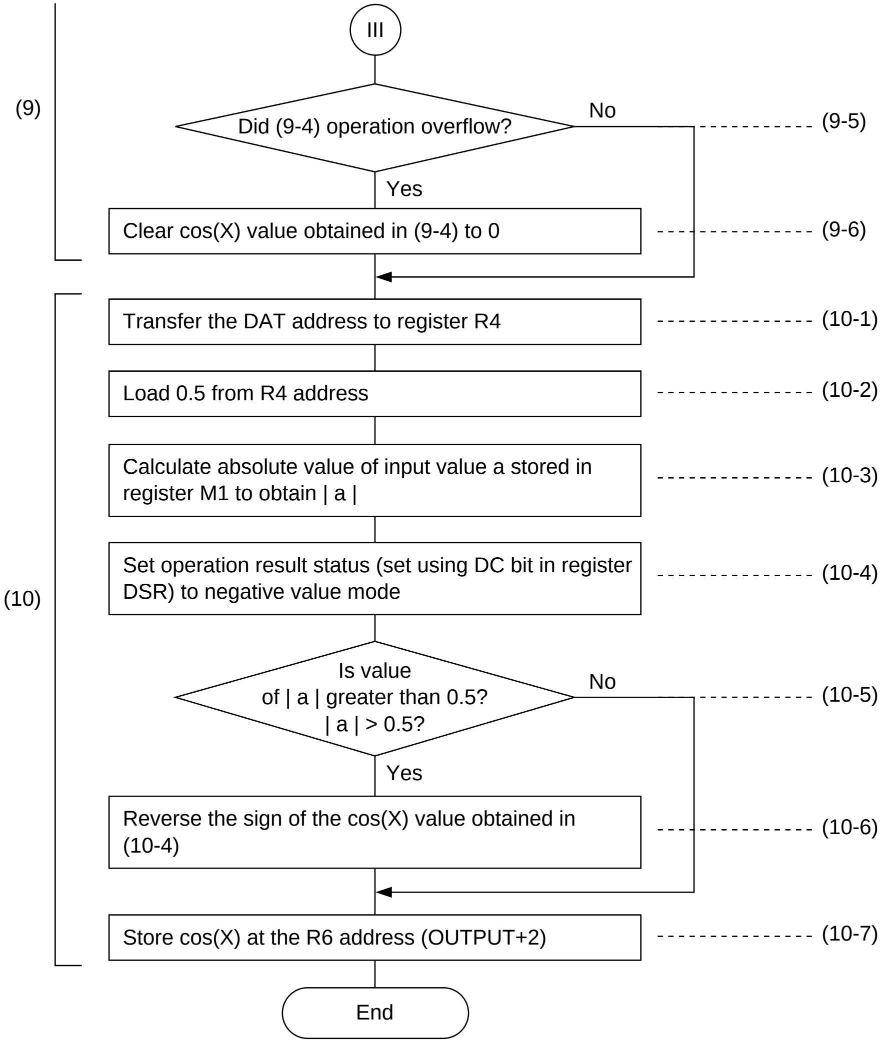
Angle X°	Input Value (a = X/π)	Logical Value (decimal)		Logical Value (hexadecimal)		Output Value (hexadecimal)	
		sin(X)	cos(X)	sin(X)	cos(X)	sin(X)	cos(X)
0	0	0	1	H'0000	H'7FFF	H'0000	H'7FFF
30	0.16667	0.5	0.86603	H'4000	H'6EDA	H'3FFE	H'6ED9
45	0.25	0.70711	0.70711	H'5A82	H'5A82	H'5A82	H'5A82
89.5	0.49722	0.99996	0.00873	H'7FFE	H'011E	H'7FFD	H'011D
152	0.84444	0.46947	–0.88295	H'3C17	H'8EFC	H'3C19	H'8EFD
179.5	0.99722	0.00873	–0.99996	H'011E	H'8002	H'011C	H'8002
–40	–0.22222	–0.64279	0.76604	H'ADB9	H'620D	H'ADBB	H'620F
–75	–0.41667	–0.96593	0.25882	H'845D	H'2121	H'845D	H'2121
–137	–0.76111	–0.681	–0.73135	H'A8B4	H'A263	H'A8B5	H'A263
–180	–1	0	–1	H'0000	H'8000	H'0002	H'8001

Flowchart


Start
Transfer INPUT address to register R4
(1-1)
Transfer WORK address to register R5
(1-2)
Transfer TABLE_SIN address to register R6
(1-3)
Transfer TABLE_COS address to register R7
(1-4)
(1)
Load input value a
(2-1)
Transfer H'FF80 to R5 address (WORK area)
(2-2)
To determine sign, copy a and store value in register M1, load 0.5
(2-3)
Calculate | | a | –0.5 |
(2-4)
Calculate | | | a | –0.5 | –0.5 | to obtain a', load H'FF80 from address R5
(2-5)
(2)
Obtain logical product of a' and H'FF80, calculate upper 9 bits (n/256) of a'
(3-1)
Convert n/256 fixed-point data to integer data by shifting n/256 6 bits to the right
(3-2)
Transfer integer data n obtained in (2-1) to R5 address (WORK area)
(3-3)
Zero-extend integer data n passed to CPU unit via R5 address to long-word size, set Y index register R9
(3-4)
(3)
I


I
(4)
Transfer H'007F to R5 address (WORK area)
(4-1)
Load H'007F from R5 address
(4-2)
Obtain logical product of a' and H'007F, calculate lower seven bits (Δa') of a'
(4-3)
(5)
Calculate 4Δa' by shifting the Δa' value obtained in (4-3) 2 bits to the left
Calculate π/4
(5-1)
Multiply 4Δa' and π/4 to calculate ΔX
(5-2)
(6)
Square (ΔX2) ΔX value obtained in (5-2)
Load sin(n × π/256) from data table in YRAM
(6-1)
Shift ΔX2 value obtained in (6-1) 1 bit to the right to obtain 1/2 (ΔX2/2)
Load –1 from register R4
(6-2)
Subtract ΔX2/2 value obtained in (6-2) from –1 loaded in (6-2) to calculate 1 – ΔX2/2
Load cos(n × π/256) from data table
(6-3)
(7)
Set operation result status (set using DC bit in register DSR) to overflow mode
(7-1)
Multiply ΔX value obtained in (5-2) and cos(n × π/256) value loaded in (6-3)
(7-2)
Multiply sin(n × π/256) value obtained in (6-1) and (1 – ΔX2/2) value obtained in (6-3)
(7-3)
Add operation results from (7-2) and (7-3) to calculate sin(X)
(7-4)
II


II
Did (7-4) operation overflow?
No
(7-5)
Yes
(7)
Decrement sin(X) value obtained in (7-4)
(7-6)
Copy input value a from register M1 to register X1
(8-1)
Set operation result status (set using DC bit in register DSR) to negative value mode
(8-2)
Shift by 1 bit input value a stored in register X1 in (8-1)
(8-3)
(8)
Is the sign bit of a 1 (a < 0)?
No
(8-4)
Yes
Reverse the sign of the sin(X) value obtained in (7-4)
(8-5)
Transfer the OUTPUT address to register R6
(8-6)
Store sin(X) at the R6 address (OUTPUT+2)
(8-7)
Set operation result status (set using DC bit in register DSR) to overflow mode
(9-1)
Multiply DX value obtained in (5-2) and sin(n × π/256) value loaded in (6-1)
(9-2)
(9)
Multiply 1 – ΔX²/2 and cos(n × π/256) values obtained in (6-3)
(9-3)
Add operation results from (9-2) and (9-3) to calculate cos(X)
(9-4)
III


III
(9)
Did (9-4) operation overflow?
No
(9-5)
Yes
Clear cos(X) value obtained in (9-4) to 0
(9-6)
Transfer the DAT address to register R4
(10-1)
Load 0.5 from R4 address
(10-2)
Calculate absolute value of input value a stored in register M1 to obtain | a |
(10-3)
Set operation result status (set using DC bit in register DSR) to negative value mode
(10-4)
(10)
Is value of | a | greater than 0.5? | a | > 0.5?
No
(10-5)
Yes
Reverse the sign of the cos(X) value obtained in (10-4)
(10-6)
Store cos(X) at the R6 address (OUTPUT+2)
(10-7)
End

Main Program

```
;*****************************************************************************************
;*                      Trigonometric function routine
;*
;*                      sinX,cosX
;*
;*****************************************************************************************
;*****************************************************************************************
;*                     Initial setting routine
;*****************************************************************************************
MAIN:
       MOV.L          #INPUT,R4
       MOV.L          #WORK,R5
       MOV.L          #TABLE_SIN,R6
       MOV.L          #TABLE_COS,R7

;*****************************************************************************************
;*                     a calculation routine
;*****************************************************************************************
                                 MOVX.W @R4,X0                ;a load
       MOV.L          #H'FF80,R0                              ;For extracting upper 9 bits
                                                               of a' (N×π/64)
       MOV.W          R0,@R5
       MOV.L          #DAT,R4
       PCOPY  X0,M1              MOVX.W @R4+,X1               ;For determining sign of M1,
                                                               load 0.5
       PCOPY  X1,Y1
       PSUB   X0,Y1,M0
       PABS   M0,A0                                           ;||a|-0.5|
       PSUB   A0,Y1,M0                                        ;|||a|-0.5|-0.5|
       PABS   M0,M0              MOVX.W @R5,X0                ;M0 = a', #H'FF80 load
;*****************************************************************************************
;*                     n calculation, R6 setting routine
;*****************************************************************************************
       PAND   X0,M0,A0                                        ;A1 = n/256
       PSHA   #-6,A0                                          ;Convert fixed-point n to
                                                               integer n
                                 MOVX.W A0,@R5                ;Pass integer n to CPU unit
       MOV.W          @R5,R1
       EXTU.W         R1,R1                                   ;
       MOV.L          R1,R9                                   ;

;*****************************************************************************************
;*                     Δa' calculation routine
;*****************************************************************************************
       MOV.L          #H'007F,R0                              ;For extracting lower 7 bits
                                                               of a' (Δa')
```


RENESAS

```
       MOV.W          R0,@R5
                                MOVX.W @R5,X1                ;#H'007F load
       PAND   X1,M0,Y1                                       ;Δa'
;*******************************************************************************************
;*                    ΔX calculation routine
;*******************************************************************************************
       PSHA   #2,Y1             MOVX.W @R4+,X1               ;4Δa', Δ/4 load
       PMULS  X1,Y1,A1                                       ;Δa'× π
;*******************************************************************************************
;*            1 – (ΔX²)/2calculation, sin(n × π/256) and cos(n × π/256) loading routine
;*******************************************************************************************
       PCOPY  A1,X0                            MOVY.W @R6+R9,Y0 ;copy,dummy load
       PMULS  A1,X0,M0                         MOVY.W @R6,Y0    ;ΔX²,sin(n×π/256) load
       PSHA   #-1,M0            MOVX.W @R4,X1  MOVY.W @R7+R9,Y1 ;ΔX²/2, -1 lode,dummy load
       PSUB   X1,M0,A1                         MOVY.W @R7,Y1    ;1-ΔX²/2,cos(n×π/256) load

;*******************************************************************************************
;*                    sin(X) calculation routine
;*******************************************************************************************
       MOV.L          #H'6,R0
       LDS            R0,DSR                               ;Set overflow mode
       PMULS  X0,Y1,M0                                     ;ΔX·cos(n×π/256)
       PMULS  A1,Y0,A0                                     ;(1-(ΔX²)/2)·sin(n×π/256)
       PABS   A0,A0
       PADD   A0,M0,A0                                     ;A0 = sin(X)
  DCT  PDEC   A0,A0                                        ;If overflow occurs, sin(X) – 1
;*******************************************************************************************
;*                    sin(X) sign processing and storing routine
;*******************************************************************************************
       PCOPY  M1,X1
       MOV.L          #H'0,R0,
       LDS            R0,DSR                                 ;Carry/borrow mode
       PSHA   #1,X1
  DCT  PNEG   A0,A0                                          ;If a < 0, reverse sign
       MOV.L          #OUTPUT,R6
                                               MOVY.W A0,@R6+   ;Store sin(X)
;*******************************************************************************************
;*                    cos(X) calculation routine
;*******************************************************************************************
       MOV.L          #H'6,R0
       LDS            R0,DSR                        ;Set overflow mode
       PMULS  X0,Y0,M0                              ;ΔX·SIN(N×π/64)
       PMULS  A1,Y1,A0                              ;(1-(ΔX·ΔX)/2)·COS(N×π/64)
       PABS   A0,A0
       PSUB   A0,M0,A0
  DCT  PCLR   A0                                    ;If overflow occurs, clear cos(X) to 0
```

```
;;*****************************************************************************************
;*                   cos(X) sign processing and storing routine
;*****************************************************************************************
       MOV.L         #DAT,R4
                               MOVX.W @R4.X0                ;0.5 load
       PABS   M1,M1                                         ;|a|
       MOV.L         #H'2,R0
       LDS           R0,DSR                                 ;Set negative value mode
       PCMP   X0,M1
  DCT  PNEG   A0,A0                                         ;If | a | < 0.5, reverse sign
                                            MOVY.W A0,@R6

EXIT:  BRA    EXIT
       NOP
MAIN_E:NOP
```

Data

```
;*****************************************************************************************
;*              Trigonometric function data routine
;*****************************************************************************************
           .SECTION XRAM,DATA,LOCATE=H'1000FF00
INPUT:          .RES.W      1                              ;External input data storage area
WORK:           .RES.W      1
DAT:            .XDATA.W    0.5,0.78540,-1
                                        ;For calculating a', for calculating π/4 (1 – ΔX²/2)

           .SECTION YRAM,DATA,LOCATE=H'1001F800
TABLE_SIN:      .XDATA.W    0,0.01227,0.02454,0.03681,0.04907,0.06132 ;N/0 - 5
                .XDATA.W    0.07356,0.08580,0.09802,0.11022,0.12241   ;N/6 - 10
                .XDATA.W    0.13458,0.14673,0.15886,0.17096,0.18304   ;N/11 - 15
                .XDATA.W    0.19509,0.20711,0.21910,0.23106,0.24298   ;N/16 - 20
                .XDATA.W    0.25487,0.26671,0.27852,0.29028,0.30201   ;N/21 - 25
                .XDATA.W    0.31368,0.32531,0.33689,0.34842,0.35990   ;N/26 - 30
                .XDATA.W    0.37132,0.38268,0.39400,0.40524,0.41643   ;N/31 - 35
                .XDATA.W    0.42756,0.43862,0.44961,0.46054,0.47140   ;N/36 - 40
                .XDATA.W    0.48218,0.49290,0.50354,0.51410,0.52459   ;N/41 - 45
                .XDATA.W    0.53500,0.54532,0.55557,0.56573,0.57581   ;N/46 - 50
                .XDATA.W    0.58580,0.59570,0.60551,0.61523,0.62486   ;N/51 - 55
                .XDATA.W    0.63439,0.64383,0.65317,0.66242,0.67156   ;N/56 - 60
                .XDATA.W    0.68060,0.68954,0.69838,0.70711,0.71573   ;N/61 - 65
                .XDATA.W    0.72425,0.73265,0.74095,0.74914,0.75721   ;N/66 - 70
                .XDATA.W    0.76517,0.77301,0.78074,0.78835,0.76584   ;N/71 - 75
                .XDATA.W    0.80321,0.81046,0.81758,0.82459,0.83147   ;N/76 - 80
                .XDATA.W    0.83822,0.84485,0.85136,0.85773,0.86397   ;N/81 - 85
                .XDATA.W    0.87009,0.87607,0.88192,0.88764,0.89322   ;N/86 - 90
                .XDATA.W    0.89867,0.90399,0.90917,0.91421,0.91911   ;N/91 - 95
                .XDATA.W    0.92388,0.92851,0.93299,0.93734,0.94154   ;N/96 - 100
                .XDATA.W    0.94561,0.94953,0.95331,0.95694,0.96043   ;N/101 - 105
                .XDATA.W    0.96378,0.96700,0.97003,0.97294,0.97570   ;N/106 - 110
                .XDATA.W    0.97832,0.98079,0.98311,0.98528,0.98730   ;N/111 - 115
                .XDATA.W    0.98918,0.99090,0.99248,0.99391,0.99518   ;N/116 - 120
                .XDATA.W    0.99631,0.99729,0.99812,0.99880,0.99932   ;N/121 - 125
                .XDATA.W    0.99970,0.99992,1                         ;N/126 - 128

TABLE_COS:      .XDATA.W    1,0.99992,0.99970,0.99932,0.99880,0.99812 ;N/0 - 5
                .XDATA.W    0.99729,0.99631,0.99518,0.99391,0.99248   ;N/6 - 10
                .XDATA.W    0.99090,0.98918,0.98730,0.98528,0.98311   ;N/11 - 15
                .XDATA.W    0.98079,0.97832,0.97570,0.97294,0.97003   ;N/16 - 20
                .XDATA.W    0.96700,0.96378,0.96043,0.95694,0.95331   ;N/21 - 25
                .XDATA.W    0.94953,0.94561,0.94154,0.93734,0.93299   ;N/26 - 30
                .XDATA.W    0.92851,0.92388,0.91911,0.91421,0.90917   ;N/31 - 35
                .XDATA.W    0.90399,0.89867,0.89322,0.88764,0.88192   ;N/36 - 40
```

```
                .XDATA.W    0.87607,0.87009,0.86397,0.85773,0.85136   ;N/41 - 45
                .XDATA.W    0.84485,0.83822,0.83147,0.82459,0.81758   ;N/46 - 50
                .XDATA.W    0.81046,0.80321,0.76584,0.78835,0.78074   ;N/51 - 55
                .XDATA.W    0.77301,0.76517,0.75721,0.74914,0.74095   ;N/56 - 60
                .XDATA.W    0.73265,0.72425,0.71573,0.70711,0.69838   ;N/61 - 65
                .XDATA.W    0.68954,0.68060,0.67156,0.66242,0.65317   ;N/66 - 70
                .XDATA.W    0.64383,0.63439,0.62486,0.61523,0.60551   ;N/71 - 75
                .XDATA.W    0.59570,0.58580,0.57581,0.56573,0.55557   ;N/76 - 80
                .XDATA.W    0.54532,0.53500,0.52459,0.51410,0.50354   ;N/81 - 85
                .XDATA.W    0.49290,0.48218,0.47140,0.46054,0.44961   ;N/86 - 90
                .XDATA.W    0.43862,0.42756,0.41643,0.40524,0.39400   ;N/91 - 95
                .XDATA.W    0.38268,0.37132,0.35990,0.34842,0.33689   ;N/96 - 100
                .XDATA.W    0.32531,0.31368,0.30201,0.29028,0.27852   ;N/101 - 105
                .XDATA.W    0.26671,0.25487,0.24298,0.23106,0.21910   ;N/106 - 110
                .XDATA.W    0.20711,0.19509,0.18304,0.17096,0.15886   ;N/111 - 115
                .XDATA.W    0.14673,0.13458,0.12241,0.11022,0.09802   ;N/116 - 120
                .XDATA.W    0.08580,0.07356,0.06132,0.04907,0.03681   ;N/121 - 125
                .XDATA.W    0.02454,0.01227,0                         ;N/126 - 128

OUTPUT:         .RES.W      2                            ;External output data storage area
```

Section 9 Matrix Operations

Overview

Matrix A (3, 3) and matrix B (3, 3) are multiplied to obtain a 32-bit precision matrix product C (3, 3). Matrixes A and B are set in XRAM and YRAM beforehand. Matrix product C is stored beginning at YRAM address H'1001FF00.

Description

1. Method of Expressing Matrixes

   Figure 9.1 shows matrix A (n,m). The element a_{ij} is a component of matrix A. Horizontal rows of components are called rows, which are numbered from the top as row1, row2, row3, ..., row i, ... and so on. Vertical columns of components are called columns, which are numbered from the left as column 1, column 2, column 3, ... column j, ... and so on. The components in the position where row I and column k intersect is called component (i,j). Component (i,j) of matrix A (n,m) is expressed as ai,j.

$$A = \begin{array}{rc} & \begin{array}{cccccc} \phantom{a_{11}} & \phantom{a_{12}} & & \text{(Column } j) & & \phantom{a_{1n}} \end{array} \\ \text{(row } i) & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{pmatrix} \end{array}$$

Figure 9.1 Matrix A

2. Method of Calculating Matrix Product

   Figure 9.2 shows the expression of the components of matrix A × matrix B = matrix product C.

$$\underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}}_{\text{Matrix A}} \times \underbrace{\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}}_{\text{Matrix B}} = \underbrace{\overset{*1}{\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}}}_{\text{Matrix Product C}}$$

*1 $c_{i,j}$: 32-bit components.

Figure 9.2 Expression of Components of Matrix A × Matrix B = Matrix Product C

The components $c_{i,j}$ of matrix product C are obtained using the following equation.

$$C_{n,m} = \sum_{i=1}^{3} (a_{n,i} \times b_{i,m})$$

The components $c_{i,j}$ of matrix product C are obtained by performing a sum of products calculation on row components $a_{n,i}$ of matrix A and column components $b_{i,m}$ of matrix B.

3. Method of Storing Matrix A, Matrix B, and Matrix Product C Components

The components $c_{n,m}$ of matrix product C are obtained by performing a sum of products calculation on row components $a_{n,i}$ of matrix A and column components $b_{i,m}$ of matrix B. The example subroutine, in order to increase the processing speed, stores the elements in XRAM and YRAM as shown in figure 9.3

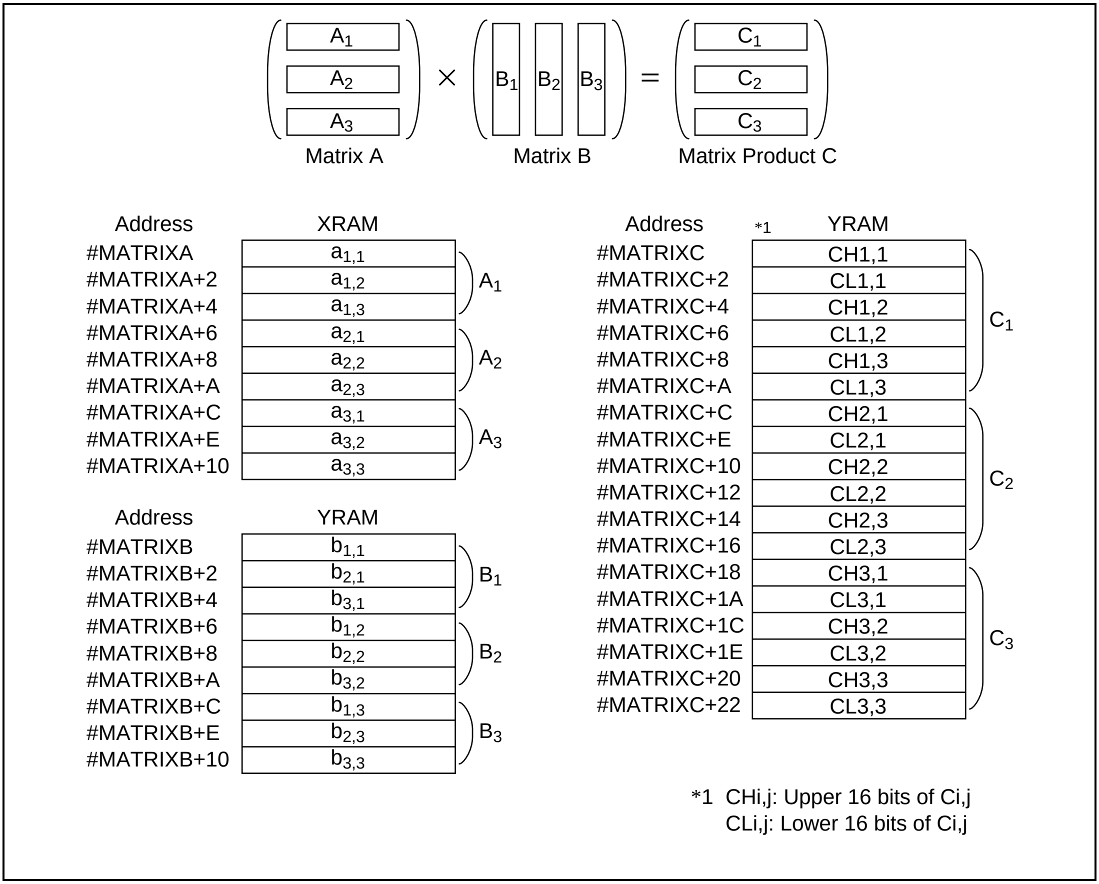


Figure 9.3 Memory Map with Matrix A, Matrix B, and Matrix Product C Components Stored

4. Algorithm for Calculating Matrix Product C

Figure 9.4 shows the algorithm for calculating matrix product C. The details of the algorithm are described below.

(1) Clear counter registers, store matrix A in the X address register (R4) and matrix B in the Y address registers (R6, R7), set the addresses for storing the components of matrix product C.

(2) Perform sum of products calculation on row components $a_{n,i}$ of matrix A and column components $b_{i,m}$ of matrix B.

(3) Store CHn,m (upper 16 bits of matrix product Cn,m) in MATRIXC+2n address and CLn,m (lower 16 bits) in MATRIXC+2n+2 address.

(4) Return matrix A column components to first column.

(5) Determine if one row of matrix product Cn,m has been calculated. If n is not 3, return to process (2). If n is 3, move to process (6).

(6) Shift matrix A row components down one row.

(7) Determine if all three rows of matrix product C have been calculated. If n is not 3, return to process (2). If n is 3, all of matrix product Cn,m has been calculated and processing ends.

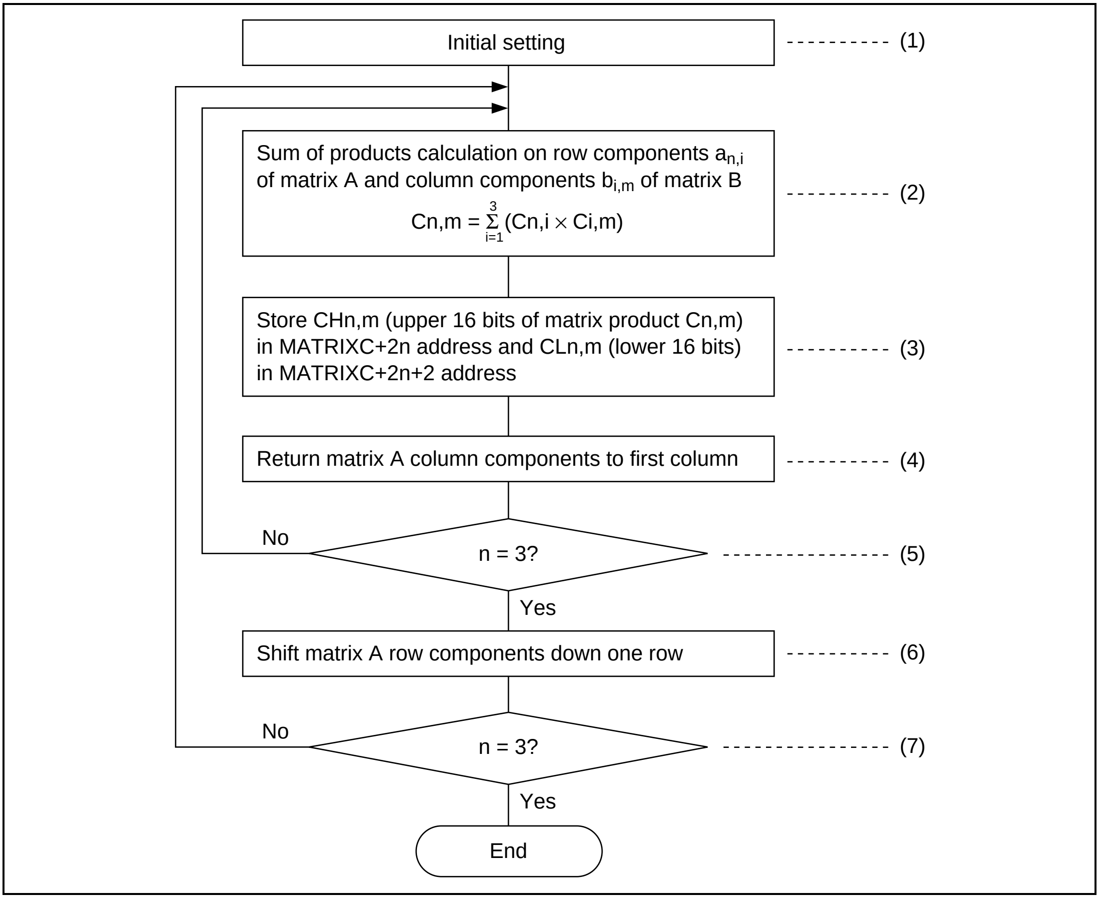
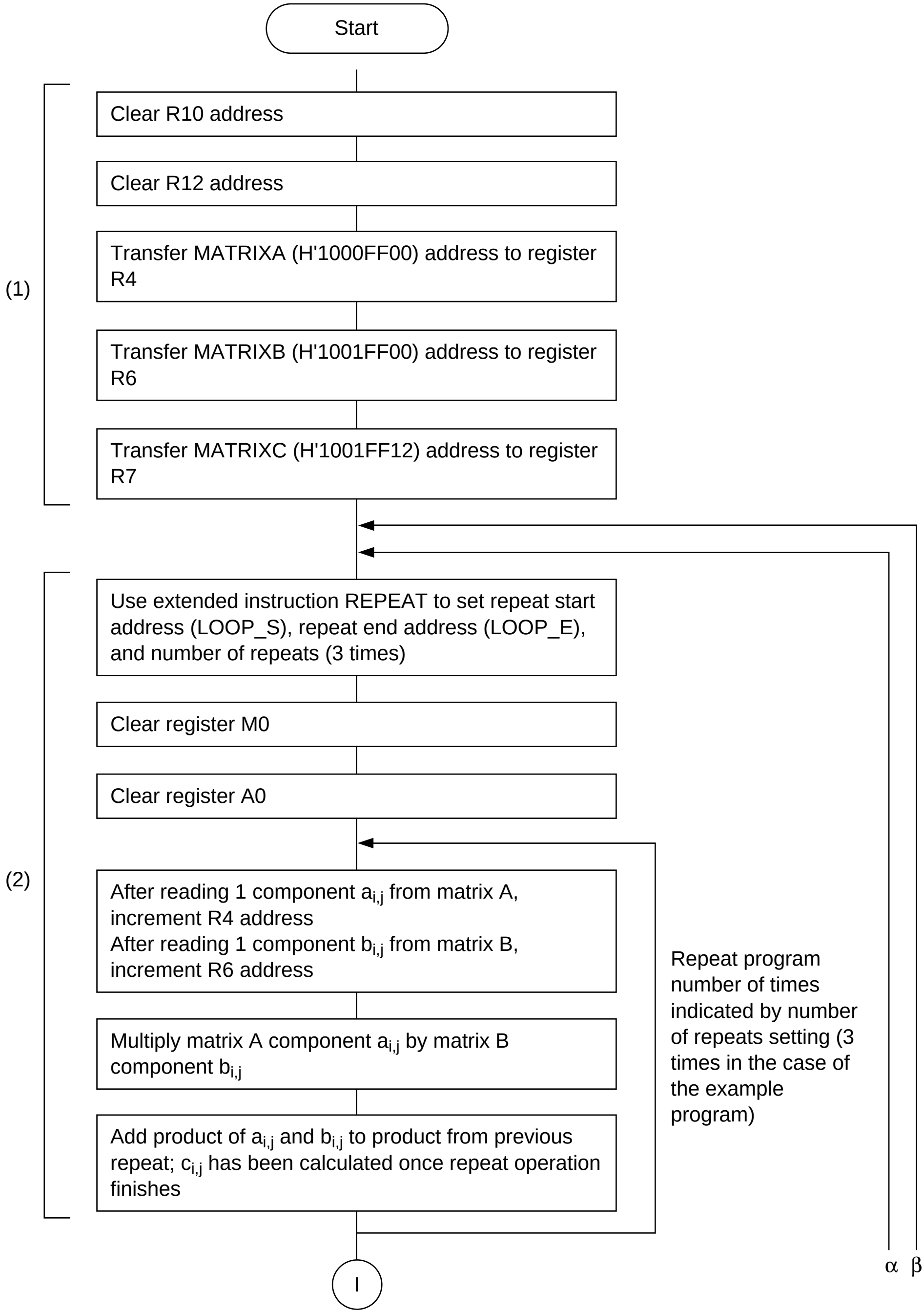
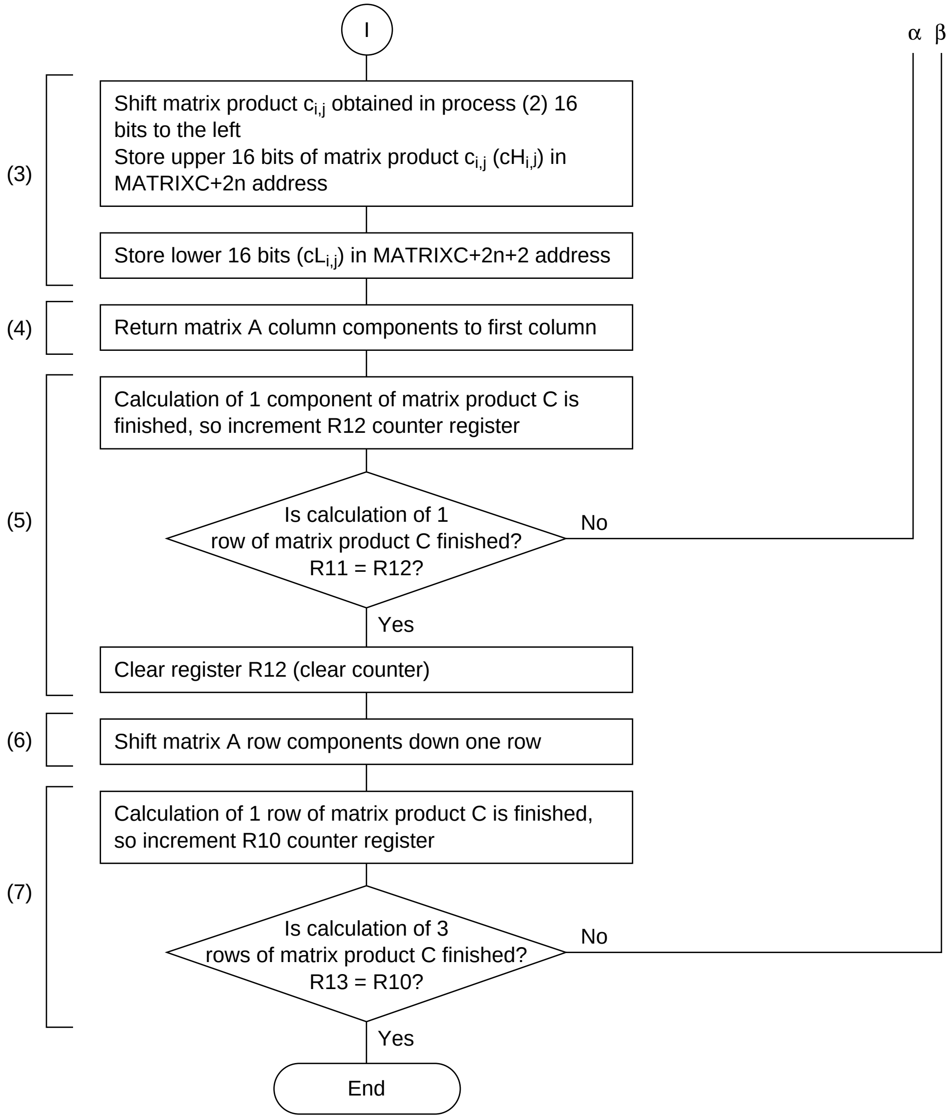


Figure 9.4 Algorithm for Calculating Matrix Product C

RENESAS

Flowchart


Start
(1)
Clear R10 address
Clear R12 address
Transfer MATRIXA (H'1000FF00) address to register R4
Transfer MATRIXB (H'1001FF00) address to register R6
Transfer MATRIXC (H'1001FF12) address to register R7
(2)
Use extended instruction REPEAT to set repeat start address (LOOP_S), repeat end address (LOOP_E), and number of repeats (3 times)
Clear register M0
Clear register A0
After reading 1 component $a_{i,j}$ from matrix A, increment R4 address
After reading 1 component $b_{i,j}$ from matrix B, increment R6 address
Multiply matrix A component $a_{i,j}$ by matrix B component $b_{i,j}$
Add product of $a_{i,j}$ and $b_{i,j}$ to product from previous repeat; $c_{i,j}$ has been calculated once repeat operation finishes
Repeat program number of times indicated by number of repeats setting (3 times in the case of the example program)
I
α
β


I
α β
(3)
Shift matrix product $c_{i,j}$ obtained in process (2) 16 bits to the left
Store upper 16 bits of matrix product $c_{i,j}$ ($cH_{i,j}$) in MATRIXC+2n address
Store lower 16 bits ($cL_{i,j}$) in MATRIXC+2n+2 address
(4)
Return matrix A column components to first column
(5)
Calculation of 1 component of matrix product C is finished, so increment R12 counter register
Is calculation of 1 row of matrix product C finished? R11 = R12?
No
Yes
Clear register R12 (clear counter)
(6)
Shift matrix A row components down one row
(7)
Calculation of 1 row of matrix product C is finished, so increment R10 counter register
Is calculation of 3 rows of matrix product C finished? R13 = R10?
No
Yes
End

Main Program

matrix.src

```
;*******************************************************************************************
;*                        Matrix operation routine
;*
;*                        [A][B]=[C]
;*
;*******************************************************************************************
MAIN:   MOV.L           #0,R10
        MOV.L           #0,R12
        MOV.L           #MATRIXA,R4
        MOV.L           #MATRIXB,R6
        MOV.L           #MATRIXC,R7
;****************************************
;Calculate all components/R10, R13
;****************************************
        MOV.L           #3,R13                    ;Set repeat value (number of rows)
MATORIX:
;**********************************
;Calculate row components of n'th row
;**********************************
        MOV.L           #3,R11                    ;Set repeat value (number of columns)
RETSU:
;****************************
;Calculate 1 component
;****************************
        BSR             SEIBUN
        NOP
        BSR             STORE
        NOP
;****************************
        ADD             #-6,R4                    ;Return address to first column of row i
                                                  of matrix A
        ADD             #1,R12                    ;Increment counter each time 1 component
                                                  of 1 row of matrix product C is
                                                  calculated
        CMP/EQ          R11,R12                   ;Is sum of products calculation for 1 row
                                                  of matrix product C finished?
        BF              RETSU
        MOV.L           #0,R12                    ;Clear counter
;**********************************
        ADD             #6,R4
        MOV.L           #MATRIXB,R6
        ADD             #1,R10                    ;Increment counter when sum of products
                                                  calculation for 1 row of matrix product C
                                                  is finished
        CMP/EQ          R13,R10                   ;Is sum of products calculation for last
                                                  row of matrix product C finished?
```

```
        BF              MATORIX
;****************************************

EXIT:   BRA             EXIT
        NOP

;*******************************************************************************************
;Matrix C 1 component calculation routine
;*******************************************************************************************
SEIBUN:
        REPEAT LOOP_S,LOOP_E,#3                                 ;Number of rows in matrix [A]
                                                                is number of repeats
        PCLR   M0                                               ;Clear for repeat
        PCLR   A0
LOOP_S:
                      MOVX.W @R4+,X0  MOVY.W @R6+,Y0            ;aij,bij load
        PMULS  X0,Y0,M0
LOOP_E: PADD   A0,M0,A0
        RTS
        NOP
;*******************************************************************************************
;Matrix C 1 component storage routine
;*******************************************************************************************
STORE:  PSHA   #16,A0                 MOVY.W A0,@R7+            ;Store upper bits of ci,j
                                      MOVY.W A0,@R7+            ;Store lower bits of ci,j
        RTS
        NOP
;**********************
MAIN_E: NOP
```

Data

```
*********************************************************************************
;*          Matrix operation data (XRAM/YRAM)
;*********************************************************************************
            .SECTION XRAM,DATA,LOCATE=H'1000FF00
MATRIXA:    . XDATA.W     0.5,0.125,0.5,0.125,0.5,0.125,0.5,0.125,0.5

            .SECTION YRAM,DATA,LOCATE=H'1001FF00
MATRIXB:    .RES.W        0.25,0.0625,0.25,0.0625,0.25,0.0625,0.25,0.0625,0.25
MATRIXC:    .RES.W        18
```


RENESAS

Section 10   Inner Product

Overview

The inner product (32-bit precision) of two non-zero n-dimensional space vectors, *a* (16-bit components) and *b* (16-bit components), is calculated. The n-dimensional space vectors *a* and *b* are set in XRAM and YRAM beforehand. The inner product of *a* and *b* is stored in YRAM at address H'1001FF00.

Description

1. Method of Expressing Space Vectors

   Figure 10.1 shows an expression of the components of n-dimensional space vector *a*. An n-dimensional space vector can be thought of as a vector consisting of a group of n real numbers. There are two ways of expressing the components of a vector: as a row vector and as a column vector.

$$\overset{*1}{\begin{bmatrix} a_1, a_2, \cdots, a_n \end{bmatrix}} \qquad\qquad \overset{*1}{\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}}$$

(a) Row vector   (b) Column vector

*1 a_i: 16-bit

Figure 10.1   Expression of Components of n-dimensional Space Vector *a*

2. Method of Calculating Inner Product

Figure 10.2 shows an expression of the components of the inner product of n-dimensional space vectors *a* and *b*. Here the inner product of vectors *a* and *b* is expressed as (*a*,*b*).

$$\overset{*1}{\begin{bmatrix} a_1, a_2, \cdots, a_i, \cdots, a_n \end{bmatrix}} \times \overset{*1}{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{bmatrix}} = \overset{*2}{a_1b_1 + a_2b_2 + \cdots + a_ib_i + \cdots + a_nb_n}$$

n-dimensional
space vector
Row vector *a*

n-dimensional
space vector
Column vector *b*

*1 a_i: 16-bit
b_i: 16-bit
*2 32-bit

Figure 10.2 Expression of Components of Inner Product of n-dimensional Space Vectors *a* and *b*

The inner product (*a*,*b*) is obtained using the following equation.

$$(a,b) = \sum_{i=1}^{3} a_ib_i$$

Using the above equation, the inner product (*a*,*b*) is obtained by performing a sum of products calculation on components a_i of space vector *a* and components b_i of space vector *b*.

RENESAS

3. Method of Storing Inner Product (*a*,*b*) of n-dimensional Space Vectors *a* and *b*

Figure 10.3 shows the method of storing the inner product (*a*,*b*) components of n-dimensional space vectors *a* and *b*, which are set in XRAM and YRAM.

Address	XRAM
VECTORA	a_1
VECTORA+2	a_2
VECTORA+4	a_3
⋮	⋮
VECTORA+2n–2	a_{n-1}
VECTORA+2n	a_n

Address	YRAM
VECTORB	b_1
VECTORB+2	b_2
VECTORB+4	b_3
⋮	⋮
VECTORB+2n–2	b_{n-1}
VECTORB+2n	b_n

Address	*1 YRAM
#IN_PRO	(*a*,*b*) H
#IN_PRO+2	(*a*,*b*) L

*1 (*a*,*b*)H: Upper 16 bits of (*a*,*b*)
(*a*,*b*)L: Lower 16 bits of (*a*,*b*)

Figure 10.3 Method of Storing Inner Product (*a*,*b*) of n-dimensional Space Vectors *a* and *b*

4. Algorithm for Calculating Inner Product

Figure 10.4 shows the algorithm for calculating the inner product (*a*,*b*). The details of the algorithm are described below.

(1) Set the addresses where the space vector *a* and *b* components are stored as well as the address for storing the inner product of *a* and *b* in X address register (R4) and Y address registers (R6, R7).

(2) Perform a sum of products calculation on components a_i of space vector *a* and components b_i of space vector *b*.

(3) Store (*a*,*b*)H, the upper 16 bits of inner product (*a*,*b*) at the IN_PRO address and (*a*,*b*)L, the lower 16 bits of inner product (*a*,*b*), at the IN_PRO+2 address. This completes the process.

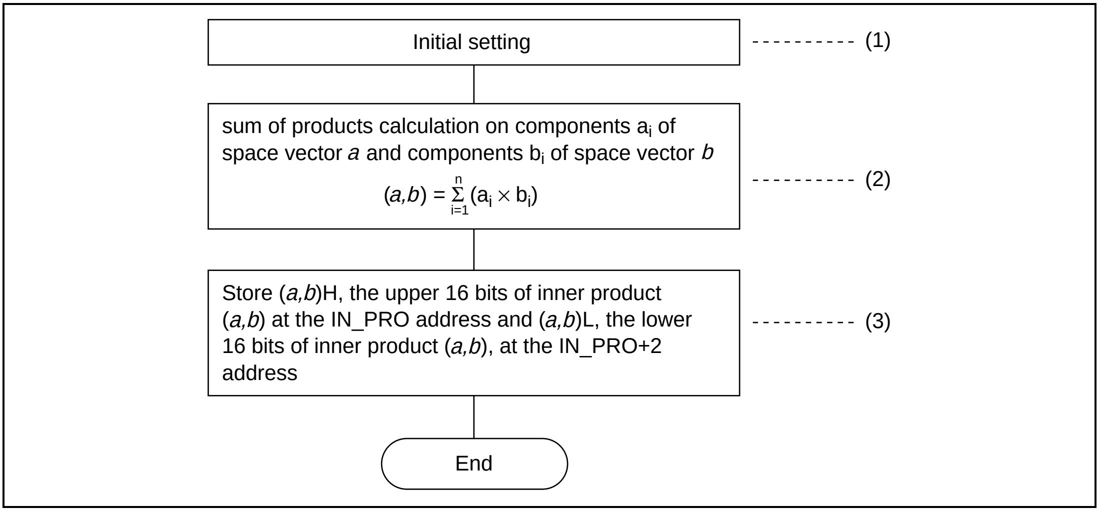
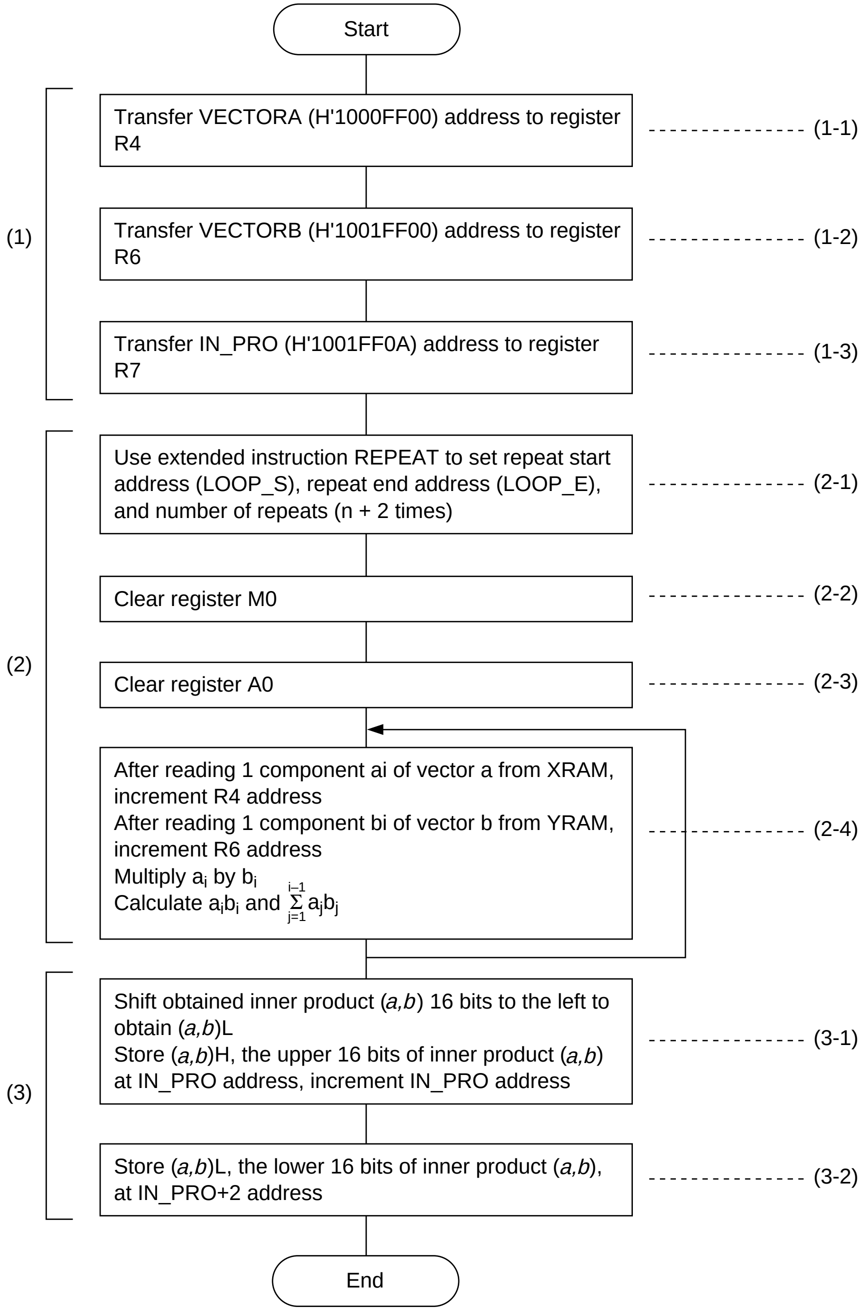


Figure 10.4 Algorithm for Calculating Inner Product

RENESAS

Flowchart



RENESAS

Main Program

This program calculates the inner product for the three-dimensional space vector {ai, bi (i = 1, 2, 3)}.

in_pro.src

```
;*******************************************************
;*                          Inner product calculation routine
;*
;*                          (a,b)=a1b1+a2b2+a3b3
;*
;*******************************************************
;*******************************************************
;*                          Initial setting routine
;*******************************************************
MAIN:   MOV.L               #VECTORA,R4
        MOV.L               #VECTORB,R6
        MOV.L               #IN_PRO,R7

;******************************************************************************************
;*                          Sum of products calculation routine
;******************************************************************************************
        REPEAT  LOOP_S,LOOP_S,#5                        ;Number of components in vector a
                                                        + 2 is number of repeats
        PCLR    A0
        PCLR    M0
        PCLR    X0
        PCLR    Y0
LOOP_S:
        PADD    A0,M0,A0  PMULS  X0,Y0,M0 MOVX.W @R4+,X0  MOVY.W @R6+,Y0 ;ai,bi load

;******************************************************************************************
;*                          Inner product storage routine
;******************************************************************************************
STORE:  PSHA    #16,A0                                  MOVY.W A0,@R7+ ;Store upper bits
of inner product
                                                        MOVY.W A0,@R7  ;Store lower bits
of inner product

EXIT:   BRA     EXIT
        NOP
MAIN_E: NOP
```

Data

```
;*****************************************************************
;*            Inner product calculation data (XRAM/YRAM)
;*****************************************************************
             .SECTION XRAM,DATA,LOCATE=H'1000FF00
VECTORA:     .XDATA.W      0.5,0.125,0.5,0,0

             .SECTION YRAM,DATA,LOCATE=H'1001FF00
VECTORB:     .XDATA.W      0.25,0.0625,0.25,0,0
IN_PRO:      .RES.W        2
```


RENESAS

Section 11 Square Root

Overview

A 16-bit fixed-point square root calculation is performed and a square root with 15-bit precision is obtained.

Description

1. I/O Value Data Format

   Figure 11.1 shows the data format for I/O values. The value, X, whose square root is to be determined is input in 16-bit format with its uppermost bit set to 0. However, it is also necessary to perform normalization on X before calculating the square root.
   The square root, √X, is output in 16-bit (1 word) format with the uppermost bit set to 0.

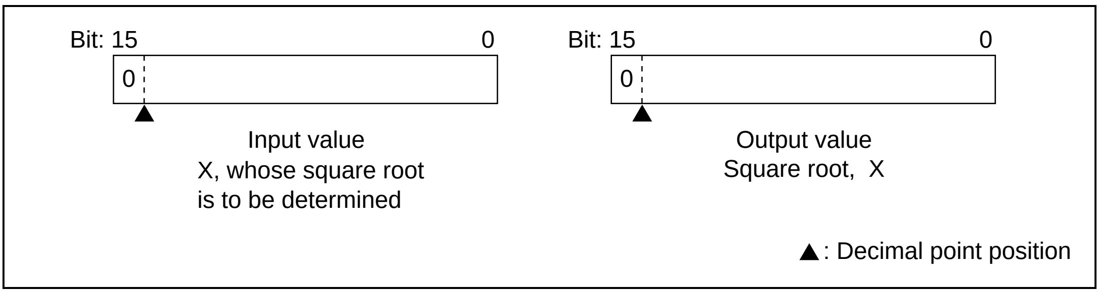


Figure 11.1 I/O Value Data Format

2. Method of Calculating Square Root

   Figure 11.2 illustrates the square root function. The example program calculates an approximate value for the square root of X using a polyline graph of the sort shown in Figure 11.2 Square Root Function. Next, a gradualization equation is used to converge on a more accurate value. This is the method used to calculate the square root, √X.
   Once normalization is performed on X, the range that can be taken by X, the value whose square root is to be calculated, is as follows.

   $0 \leq X < 1.0$
   (H'00000 ≤ X ≤ H'7FFF)

   In the square root function shown in Figure 11.2, the slope of the polyline graph is created by a combination of comparatively gentle sections greater than 0.1 and steep sections less than 0.1, resulting in approximation equations (1) and (2). Using these two equations, an approximate square root value (y0) is obtained.

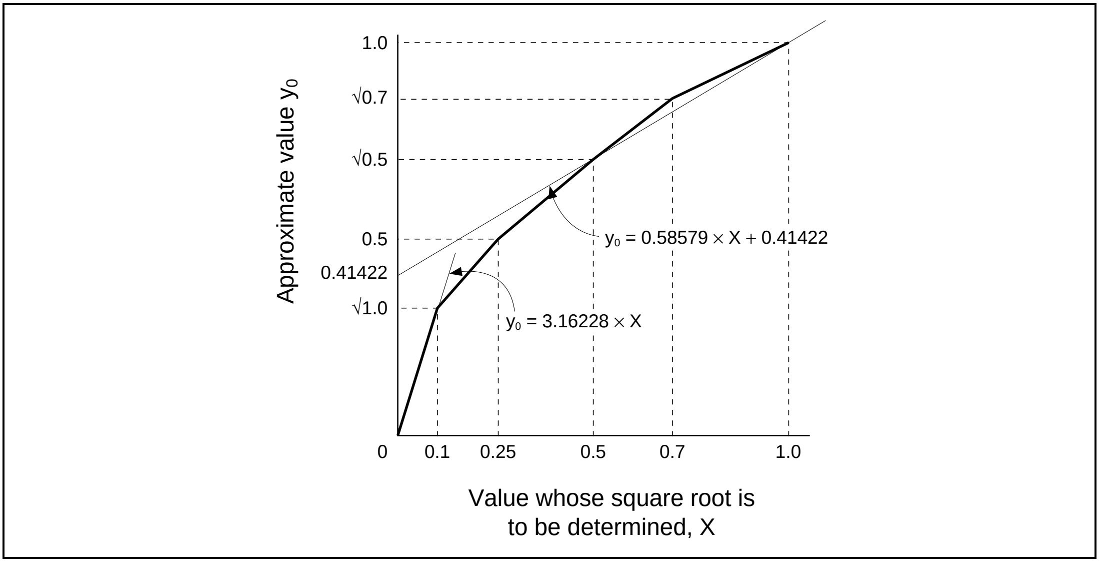


Figure 11.2 Square Root Function

Input value $X > 0.1$

$y_0 = 0.58579 \times X + 0.41422$ -- (1)

Input value $X \leq 0.1$

$y_0 = 3.16228 \times X$ -- (2)

(The actual program uses $y_0 = 0.79057 \times X \times 2^2$.)

Note that equation (2) cannot be used without modification for fixed-point calculation. Therefore, normalization is performed and it is used as $y_0 = 0.79057 \times X \times 2^2$.

Next, the value y_0 obtained with approximation equations (1) and (2) is assigned to gradualization equation (3) to obtain a more accurate square root value, $\sqrt{X}$.

$y_0 = \sqrt{X} = 1/2\ (y_0 + X/y_0)$ -- (3)

Here, in item 2 of equation (3), since the value whose square root is being calculated, X, has been normalized, X/y_0 must be a normalized value in order to $y_0 > X$ after the calculations of equations (1) and (2). In the sample program gradualization equation (3) is performed three times, resulting in a square root value with 15-bit precision.

3. Algorithm for Fixed-point Square Root Calculation

The algorithm for fixed-point square root calculation is described below.

(1) Initial settings are performed.

(2) It is determined whether X, the value whose square root is to be calculated, is not 0. If X is 0, the square root, $\sqrt{X}$, is given as 0 and processing ends.

(3) It is determined whether X, the value whose square root is to be calculated, is a negative number. If X is a negative number, the square root, $\sqrt{X}$, is given as H'FFFF and processing ends.

(4) X, the value whose square root is to be calculated, is compared to H'7FFB to determine whether it is larger or smaller. If X > H'7FFB, the square root, $\sqrt{X}$, is given as $\sqrt{X}(=X)$ and processing ends.

(5) X, the value whose square root is to be calculated, is compared to 0.1 to determine whether it is larger or smaller. If X > 0.1, processing continues with (6). If X ≤ 0.1, processing continues with (6)'.

(6) Equation (1) is used to calculate approximate square root y_0. Processing continues with (7).

(6)' Equation (2) is used to calculate approximate square root y_0. Processing continues with (7).

(7) Approximate square root y_0 is compared to X, the value whose square root is being calculated, to determine whether it is larger or smaller. If y_0 = X, approximate square root y_0 is divided by 2, 0.5 (H'4000) is added, the result is given as the square root, $\sqrt{X}$, and processing ends.

(8) If the comparison in (7) shows that X, the value whose square root is being calculated, is greater than approximate square root y_0, gradualization equation X/y_0 is not performed. In this case the square root, $\sqrt{X}$, is given as H'FFFF and processing ends.

(9) Gradualization equation (3) is used to calculate square root value y, which is given as the square root, $\sqrt{X}$, and processing ends.

Figure 11.3 shows the algorithm used for calculating the square root.

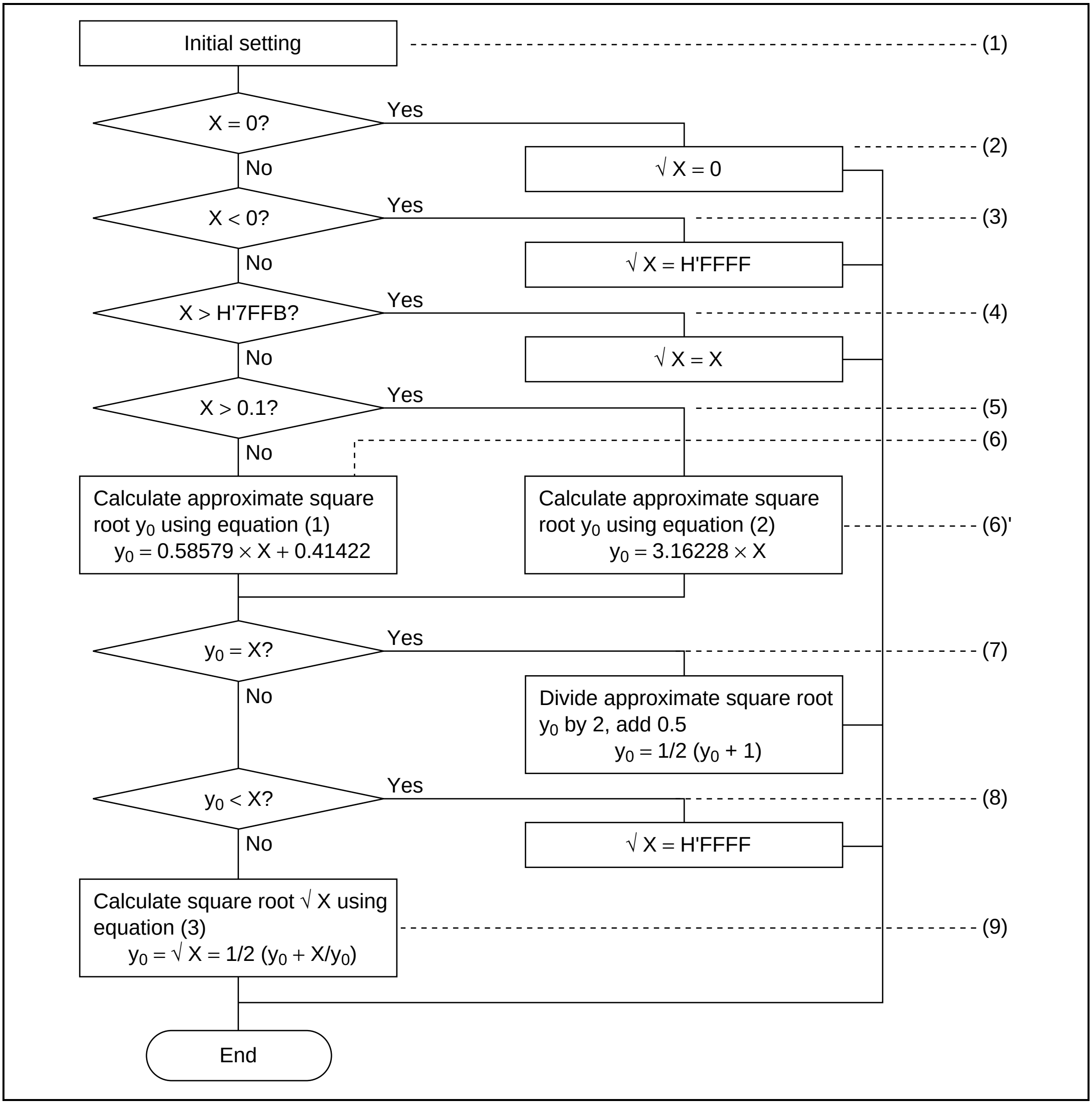
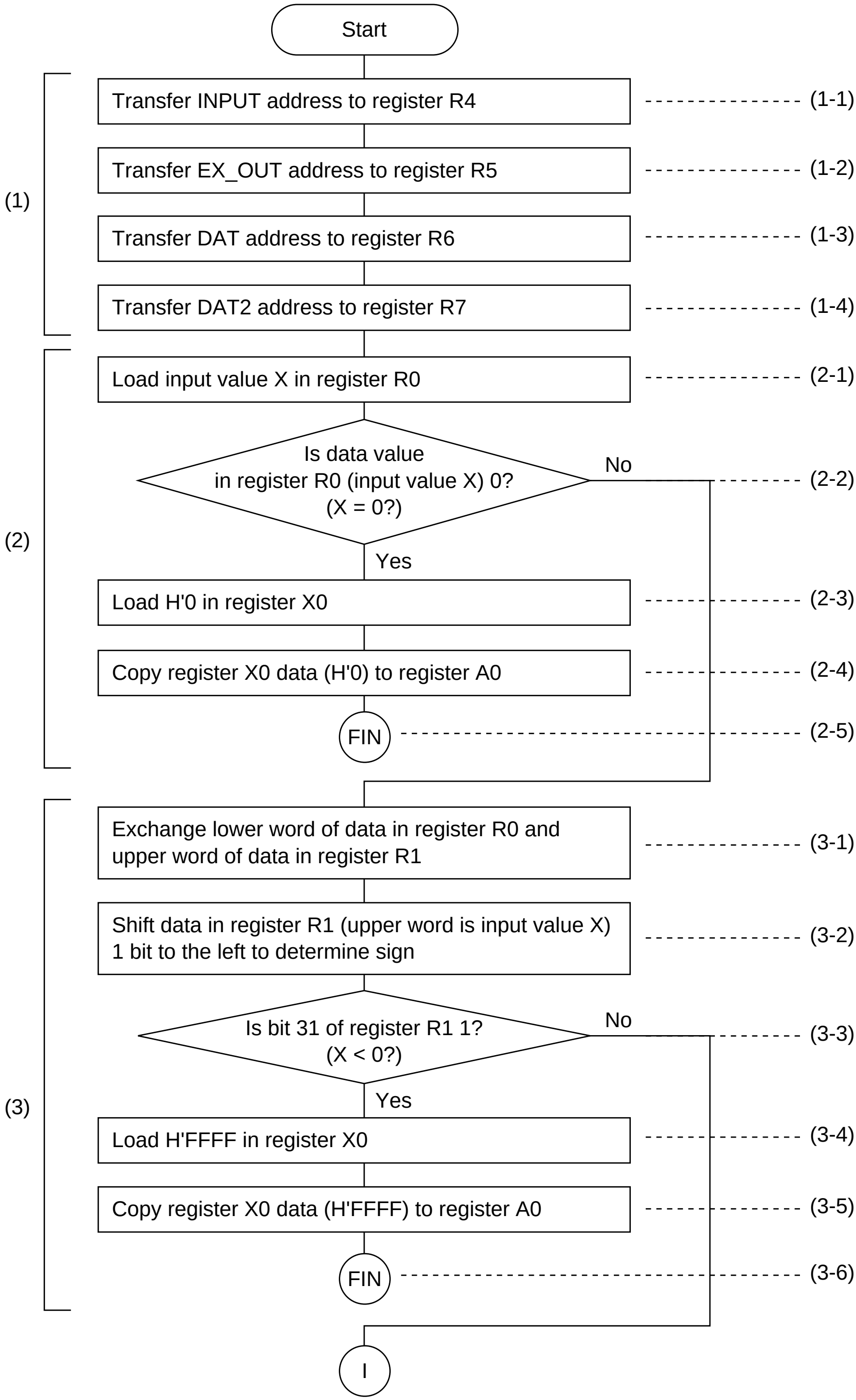
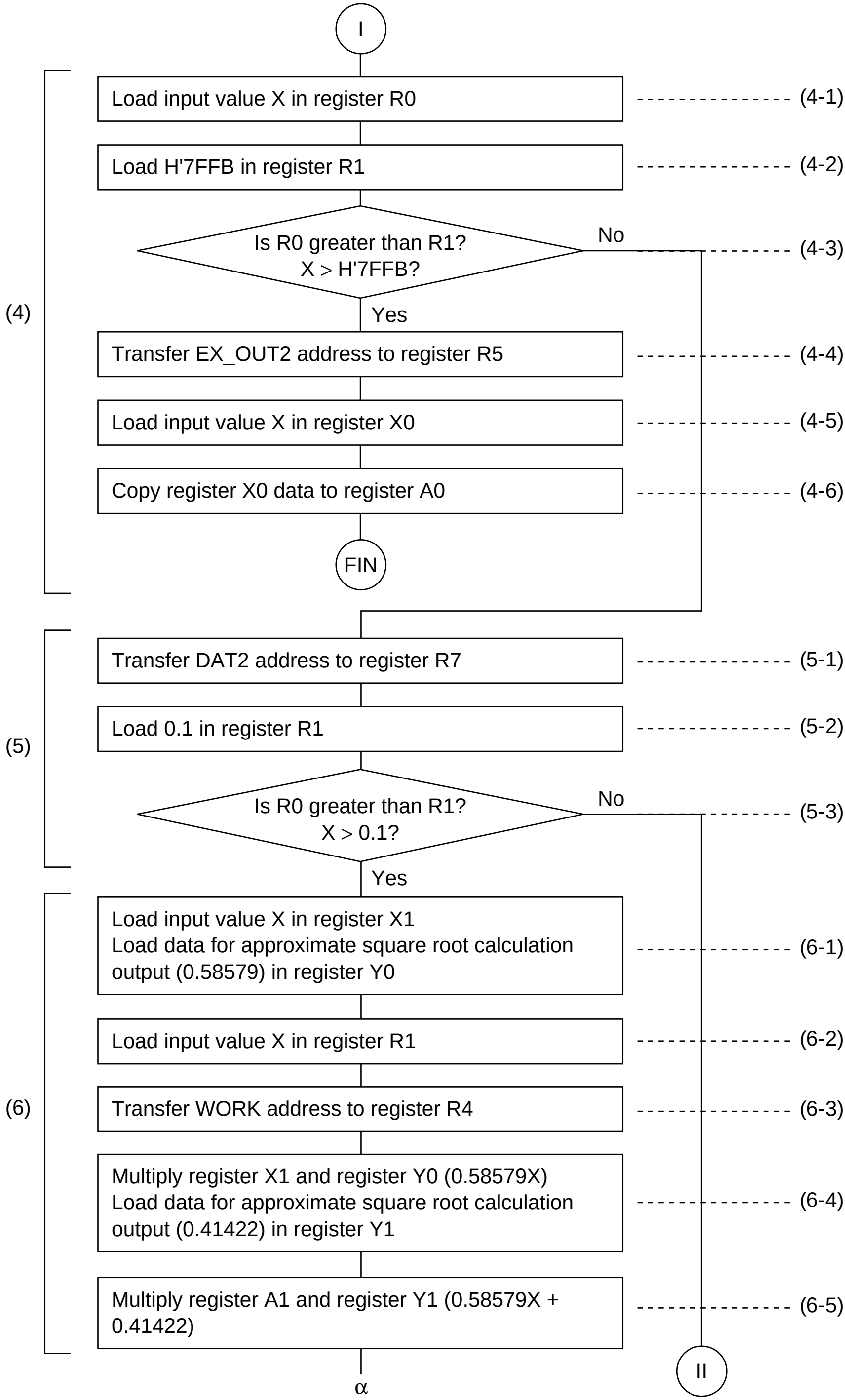


Figure 11.3 Algorithm for Calculating Square Root

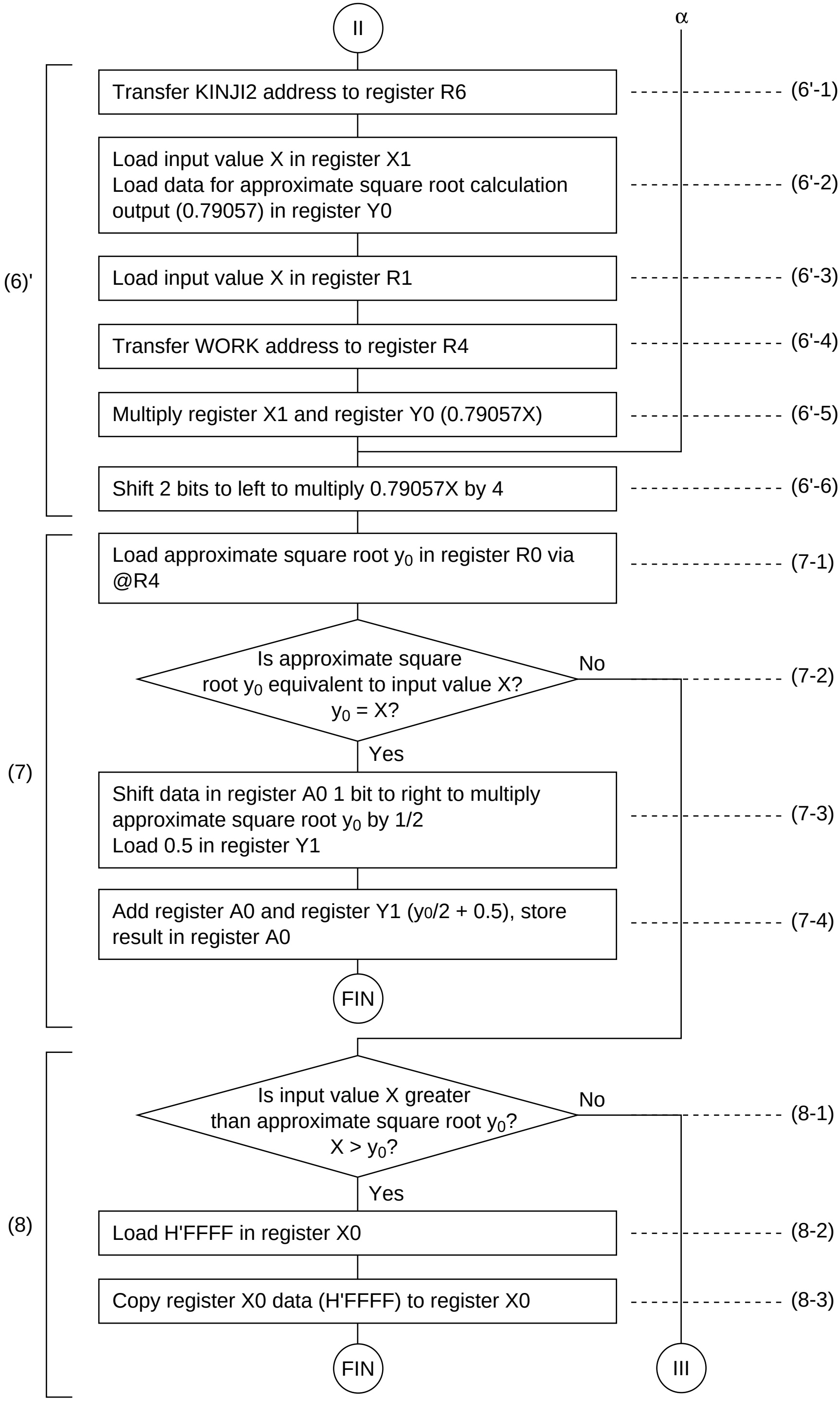
Flowchart

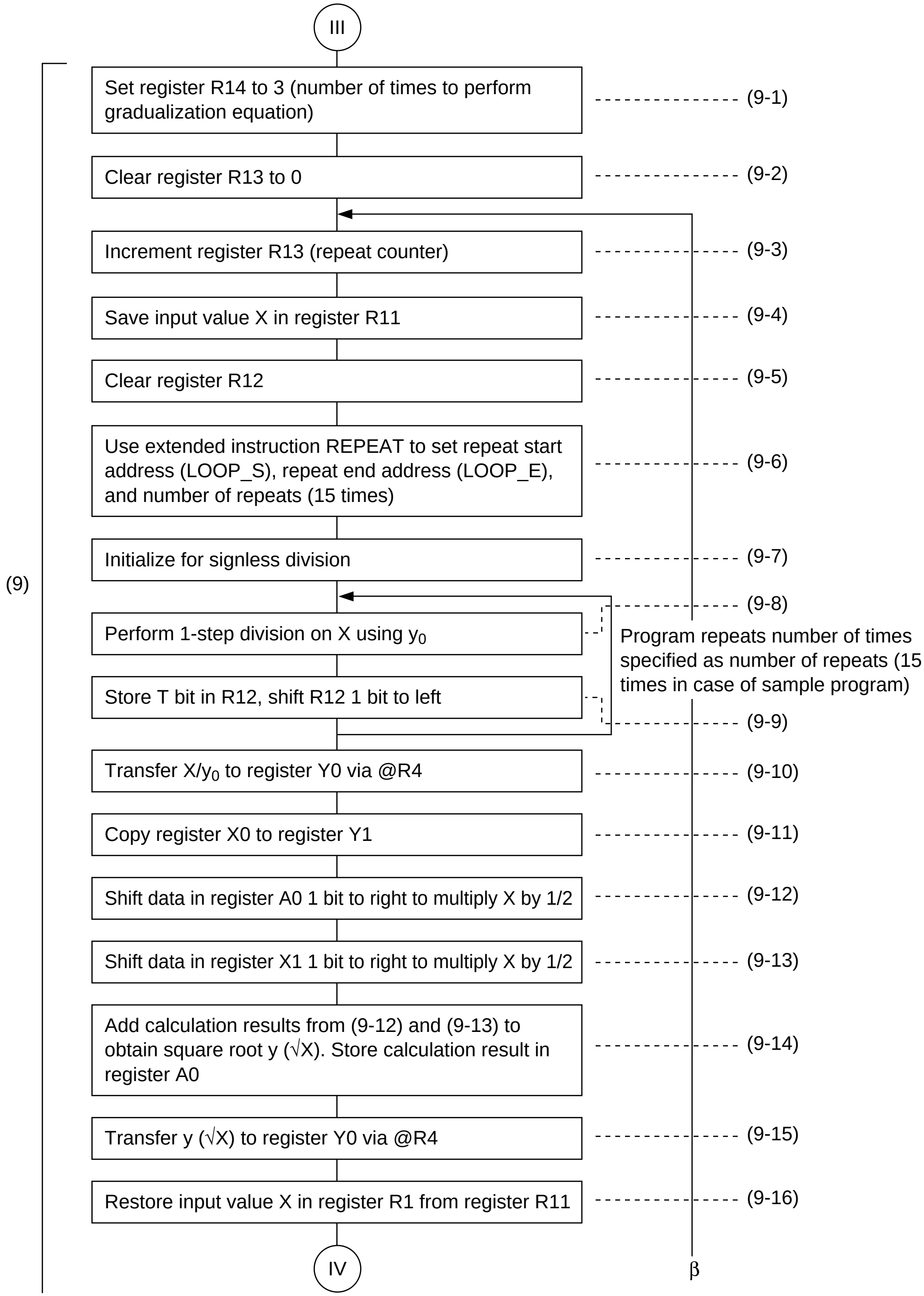


RENESAS

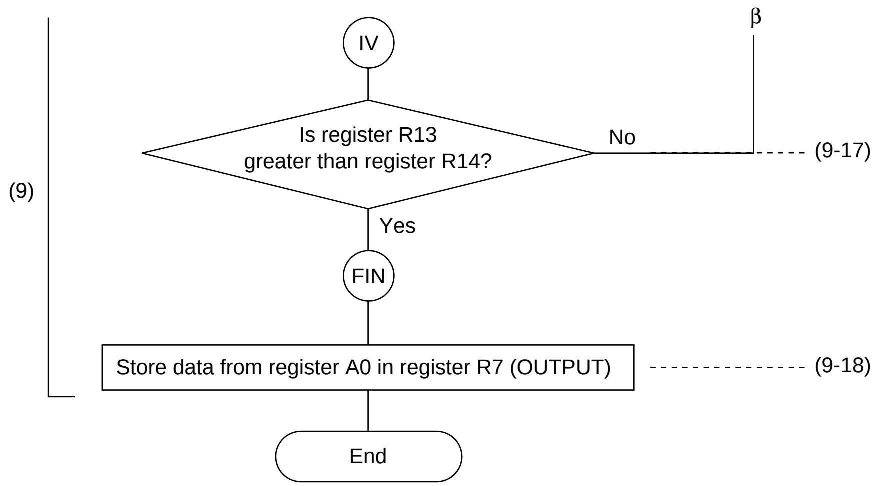

I
Load input value X in register R0
(4-1)
Load H'7FFB in register R1
(4-2)
Is R0 greater than R1?
X > H'7FFB?
No
(4-3)
Yes
(4)
Transfer EX_OUT2 address to register R5
(4-4)
Load input value X in register X0
(4-5)
Copy register X0 data to register A0
(4-6)
FIN
Transfer DAT2 address to register R7
(5-1)
Load 0.1 in register R1
(5-2)
(5)
Is R0 greater than R1?
X > 0.1?
No
(5-3)
Yes
Load input value X in register X1
Load data for approximate square root calculation output (0.58579) in register Y0
(6-1)
Load input value X in register R1
(6-2)
(6)
Transfer WORK address to register R4
(6-3)
Multiply register X1 and register Y0 (0.58579X)
Load data for approximate square root calculation output (0.41422) in register Y1
(6-4)
Multiply register A1 and register Y1 (0.58579X + 0.41422)
(6-5)
α
II

RENESAS


II
α
Transfer KINJI2 address to register R6
(6'-1)
Load input value X in register X1
Load data for approximate square root calculation output (0.79057) in register Y0
(6'-2)
(6)'
Load input value X in register R1
(6'-3)
Transfer WORK address to register R4
(6'-4)
Multiply register X1 and register Y0 (0.79057X)
(6'-5)
Shift 2 bits to left to multiply 0.79057X by 4
(6'-6)
Load approximate square root y_0 in register R0 via @R4
(7-1)
Is approximate square root y_0 equivalent to input value X? y_0 = X?
No
(7-2)
Yes
(7)
Shift data in register A0 1 bit to right to multiply approximate square root y_0 by 1/2
Load 0.5 in register Y1
(7-3)
Add register A0 and register Y1 (y_0/2 + 0.5), store result in register A0
(7-4)
FIN
Is input value X greater than approximate square root y_0? X > y_0?
No
(8-1)
Yes
(8)
Load H'FFFF in register X0
(8-2)
Copy register X0 data (H'FFFF) to register X0
(8-3)
FIN
III


III
Set register R14 to 3 (number of times to perform gradualization equation)
(9-1)
Clear register R13 to 0
(9-2)
Increment register R13 (repeat counter)
(9-3)
Save input value X in register R11
(9-4)
Clear register R12
(9-5)
Use extended instruction REPEAT to set repeat start address (LOOP_S), repeat end address (LOOP_E), and number of repeats (15 times)
(9-6)
Initialize for signless division
(9-7)
(9)
(9-8)
Perform 1-step division on X using y_0
Program repeats number of times specified as number of repeats (15 times in case of sample program)
Store T bit in R12, shift R12 1 bit to left
(9-9)
Transfer X/y_0 to register Y0 via @R4
(9-10)
Copy register X0 to register Y1
(9-11)
Shift data in register A0 1 bit to right to multiply X by 1/2
(9-12)
Shift data in register X1 1 bit to right to multiply X by 1/2
(9-13)
Add calculation results from (9-12) and (9-13) to obtain square root y (√X). Store calculation result in register A0
(9-14)
Transfer y (√X) to register Y0 via @R4
(9-15)
Restore input value X in register R1 from register R11
(9-16)
IV
β

RENESAS


β
IV
Is register R13
greater than register R14?
No
(9-17)
(9)
Yes
FIN
Store data from register A0 in register R7 (OUTPUT)
(9-18)
End

RENESAS

Main Program

rout.src

```
;*******************************************************************************************
;*                                  Square root calculation routine
;*
;*                                  √X
;*
;*******************************************************************************************
;*******************************************************************************************
;*                      Initial setting routine
;*******************************************************************************************
MAIN:
        MOV.L           #INPUT,R4
        MOV.L           #EX_OUT,R5
        MOV.L           #KINJI1,R6
        MOV.L           #DAT1,R7

;*******************************************************************************************
;*                      Zero check of value to have square root calculated routine
;*******************************************************************************************
        MOV.W           @R4,R0
        CMP/EQ          #0,R0
        BF              ZERO_CH                                 ;If zero, do following
processing
                                    MOVX.W @R4,X0
        PCOPY   X0,A0
        BRA             FIN                                     ;End of processing
        NOP

;*******************************************************************************************
;*                      Negative value check of value to have square root calculated routine
;*******************************************************************************************
ZERO_CH:
        SWAP            R0,R1
        SHAL            R1
        BF              MINUS_CH                                ;If negative, do following
                                                                processing
                                    MOVX.W @R5,X0
        PCOPY           X0,A0
        BRA             FIN                                     ;End of processing
        NOP

;;******************************************************************************************
;*                      Comparison of value to have square root calculated and F'7FFB routine
;*******************************************************************************************
MINUS_CH:
```


RENESAS

```
        MOV.W           @R4,R0                                    ;X load
        MOV.W           @R7,R1                                    ;H'7FFB load
        CMP/GT          R1,R0                                     ;R0 > R1 ?
        BF              EQU_SEL                                   ;If X > F'7FFB, do following
                                                                  processing
        MOV.L           #EX_OUT2,R5
                                    MOVX.W @R5,X0                 ;X load
        PCOPY   X0,A0
        BRA             FIN
        NOP

;*******************************************************************************************
;*                      Approximation equation selection routine
;*******************************************************************************************
EQU_SEL:
        MOV.L           #DAT2,R7
        MOV.W           @R7,R1
        CMP/GT          R1,R0
        BF              Y0_PRO2                                   ;If X ≤ 0.1, jump

********************************************************************************************
;*                      Approximate square root y0 calculation routine
;*******************************************************************************************
Y0_PRO1:
                                    MOVX.W @R4,X1  MOVY.W @R6+,Y0 ;Load input value X (value to
                                                                  have square root calculated)
                                                                  for use in calculating
                                                                  approximate square root
        MOV.W           @R4,R1                                    ;Keep input value X (value to
                                                                  have square root calculated)
                                                                  in R1
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A1                   MOVY.W @R6+,Y1 ;0.58579X,0.41422 load
        PADD    A1,Y1,A0                                          ;0.58579X+0.41422 -> y0
        BRA             HIKAKU
        NOP

;********************************************************************************************
;*                      Approximation equation (2) y0 calculation routine
;*******************************************************************************************
Y0_PRO2:
        MOV.L           #KINJI2,R6
                                    MOVX.W @R4,X1  MOVY.W @R6+,Y0 ;Load input value X (value to
                                                                  have square root calculated)
                                                                  for use in calculating
                                                                  approximate square root
        MOV.W           @R4,R1                                    ;Keep input value X (value to
                                                                  have square root calculated)
                                                                  in R1
        MOV.L           #WORK,R4
```

```
        PMULS  X1,Y0,A1                      MOVY.W @R6+,Y1 ;0.58579X,0.41422 load
        PSHA   #2,A0                                        ;0.58579X+0.41422 -> y0

**************************************************************************************
;*                 Comparison of approximate square root and value to have square root
calculated routine/Part 1
;**************************************************************************************
HIKAKU:
                                 MOVX.W A0,@R4              ;Pass to CPU unit
        MOV.W          @R4,R0
        CMP/EQ         R0,R1                                ;Approximate square root y0 =
                                                            input value X (value to have
                                                            square root calculated)?
        BF             NOT_EQ                               ;If y0 ≠ X, do following
                                                            processing
        PSHA   #-1,A0                        MOVY.W @R6,Y1  ;y0/2,0.5 load
        PADD   A0,Y1,A0                                     ;y0/2-0.5
        BRA            FIN                                  ;End of processing
        NOP

;**************************************************************************************
;*                 Comparison of approximate square root and value to have square root
calculated routine/Part 2
;**************************************************************************************
NOT_EQ:
        CMP/GT         R0,R1
        BF             NOT_GT                               ;If y0 < X, do following
                                                            processing
                                             MOVX.W @R5,X0  ;H'FFFF load
        PCOPY  X0,A0
        BRA            FIN
        NOP

;**************************************************************************************
;*                 Square root y calculation using gradualization equation routine
;**************************************************************************************
NOT_GT:
        MOV.L          #3,R14                               ;Set number of repeats
        MOV.L          #0,R13
LENEAR_LP:
        ADD            #1,R13                               ;Increment counter

        MOV            R1,R11                               ;push X
        MOV.L          #0,R12                               ;Clear register R12
        REPEAT         LOOP_S,LOOP_E,#15
        DIV0U                                               ;Signless initialization
LOOP_S:
        DIV1           R0,R1                                ;R1/R0
LOOP_E:
```

```
        ROTCL          R12                                  ;Store T bit
        MOV.W          R12,@R4
                                    MOVX.W @R4,X0
        PCOPY  X0,Y1
        PSHA   #-1,A0                                       ;y0/2
        PSHA   #-1,Y1                                       ;(X/y0)/2
        PADD   A0,Y1,A0
                                    MOVX.W A0,@R4
        MOV.W          @R4,R0
        MOV            R11,R1                               ;pop X

        CMP/GT         R14,R13
        BF             LENEAR_LP                            ;If set number of repeats has
                                                            been performed, escape

FIN:    MOV.L          #OUTPUT,R7
                                          MOVY.W A0,@R7 ;Store square root √X

EXIT:   BRA            EXIT
        NOP
MAIN_E: NOP
```

Data

```
;*********************************************************************************************
;*                        Square root calculation data (XRAM/YRAM)
;*********************************************************************************************
             .SECTION XRAM,DATA,LOCATE=H'1000FF00
INPUT:       .RES.W       1                       ;External input data storage area
WORK:        .RES.W       1                       ;Work area
EX_OUT:      .DATA.W      H'FFFF                  ;Output value if input value X < 0
EX_OUT2:     .XDATA.W     1                       ;Output value if input value X > H'7FFB

             .SECTION YRAM,DATA,LOCATE=H'1001FF00
KINJI1:      .XDATA.W     0.58579,0.41422,0.5     ;Approximation equation (1)
KINJI2:      .XDATA.W     0.79057                 ;Approximation equation (2)
DAT1:        .DATA.W      H'7FFB
DAT2:        .XDATA.W     0.1
OUTPUT:      .RES.W       1                       ;External output data storage area
```

Execution Example

The input values for X (INPUT) and the square root $\sqrt{X}$ values calculated (OUTPUT) are shown in table 11.1.

Table 11.1 Square Root $\sqrt{X}$ Calculation Results (3 Executions of Gradualization Equation)

Input Value X (decimal)	Input Value X (hexadecimal)	Logical Value (decimal) $\sqrt{X}$	Logical Value (hexadecimal) $\sqrt{X}$	Output Value (hexadecimal) $\sqrt{X}$
0.9999	H'7FFC	0.99995	H'7FFE	H'7FFF
0.99987	H'7FFB	0.99993	H'7FFD	H'7FFD
0.85	H'6CCD	0.92195	H'7602	H'7602
0.523	H'42F1	0.72319	H'5C91	H'5C90
0.34	H'2BB5	0.5831	H'4AA3	H'4AA2
0.136	H'1168	0.36878	H'2F34	H'2F33
0.087	H'0B23	0.29496	H'25C1	H'25C1
0.01	H'0147	0.1	H'0CCD	H'0CC9
0	H'0000	0	H'0000	H'0000
–0.7	H'A667	—	—	H'FFFF

Section 12 Square Mean Error

Overview

The square mean error of two variables, a[i] (16-bit components) and b[i] (16-bit components), is calculated.

$(i = 1, 2, ..., n)$

Description

1. Method of Obtaining Square Mean Error

In order to obtain the square mean error, first the error e[i] for the two variables, a[i] and b[i], must be considered. The relevant equation is given as equation (1) below.

$$^{*1}\ e[i] = a[i] - b[i] \quad \text{---} \quad (1)$$

$(i = 1, 2, ..., n)$

Next, the error distribution Se^2 is obtained. The error distribution Se^2 can be calculated by dividing the sum total of the squares of the errors e[i] by the number of components (n). The components of the squares of the errors e[i] can be expressed as follows.

$$1/n \cdot \Sigma e[i]^2 = 1/n \cdot (a[1] - b[1])^2 + (a[2] - b[2]^2 + \cdots + (a[n] - b[n])^2$$

The error distribution Se^2 can be obtained using equation (2) below.

$$Se^2 = 1/n \cdot \sum_{i=1}^{n} (a[i] - b[i])^2 \quad \text{---} \quad (2)$$

The square mean error $E[Se^2]$ is expressed as the square root of the error distribution Se^2. The relevant equation for obtaining the square mean error $E[Se^2]$ is shown as equation (3) below.

$$E[e^2] = \sqrt{1/n \cdot \sum_{i=1}^{n} (a[i] - b[i])^2} \quad \text{---} \quad (3)$$

*1 a[i]: 16-bit
b[i]: 16-bit
e[i]: 16-bit

2. Method of Storing Components of Variables a[i] and b[i]

On order to obtain the square mean error, it is first necessary to calculate the sum total of the squares of the errors e[i]. To increase processing speed, the components of a[i] and b[i] are stored in XRAM and YRAM ahead of time as shown in figure 12.1. Note that 0 is stored in VECTORA+2n, VECTORA+2n+2, VECTORB+2n, and VECTORB+2n+2 of XRAM and YRAM. The example program will not run properly if zeros are not stored in these locations.

For division by the number of components n, the numeric value 1/n is stored in XRAM. The actual program does not use a DSP instruction, but rather multiplies values by 1/n.

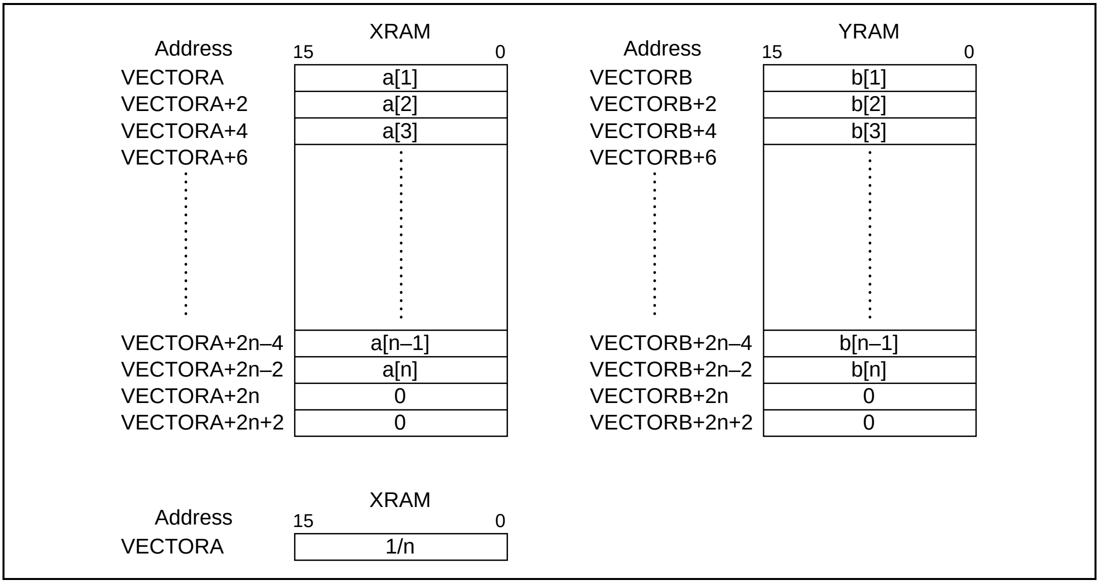


Figure 12.1 Memory Map of Storage of Variables a[i] and b[i], Etc.

3. Algorithm for Calculating Square Mean Error

The algorithm used to calculate the square mean error is described below.

(1) Perform initial settings.

(2) Set items (2) and (3) so that the number of repeats is number of elements n + 2. Two extra repeats are added since the following four instructions run in parallel.

Calculate $e[i]^2 + \sum_{j=1}^{i-1} e[j]^2$, calculate e[i], load a[i], load b[i]

(3) Calculate the error e[i] for a[i] and b[i].

(4) Divide $\sum_{i=1}^{n} (a[i] - b[i])^2$, which was obtained using processes (2) and (3), by n.

(5) Calculate the square root of the input error distribution Se^2. This yields the square mean error and completes the processing. (For details, see 3. Algorithm for Fixed-point Square Root Calculation in 11. Square Root.)

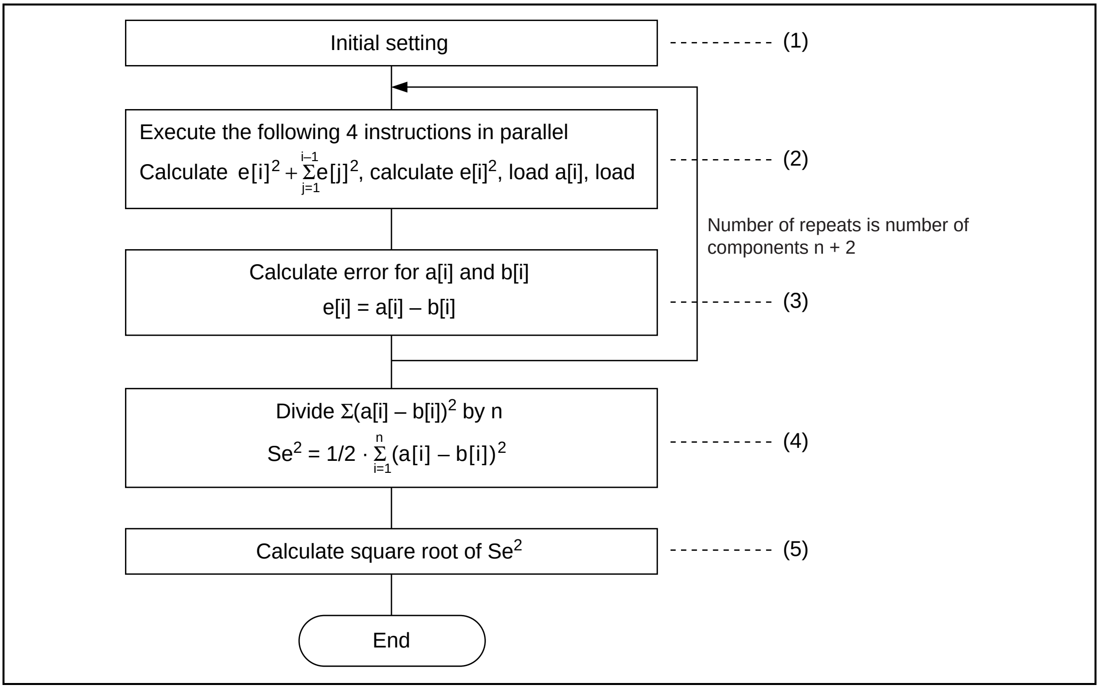
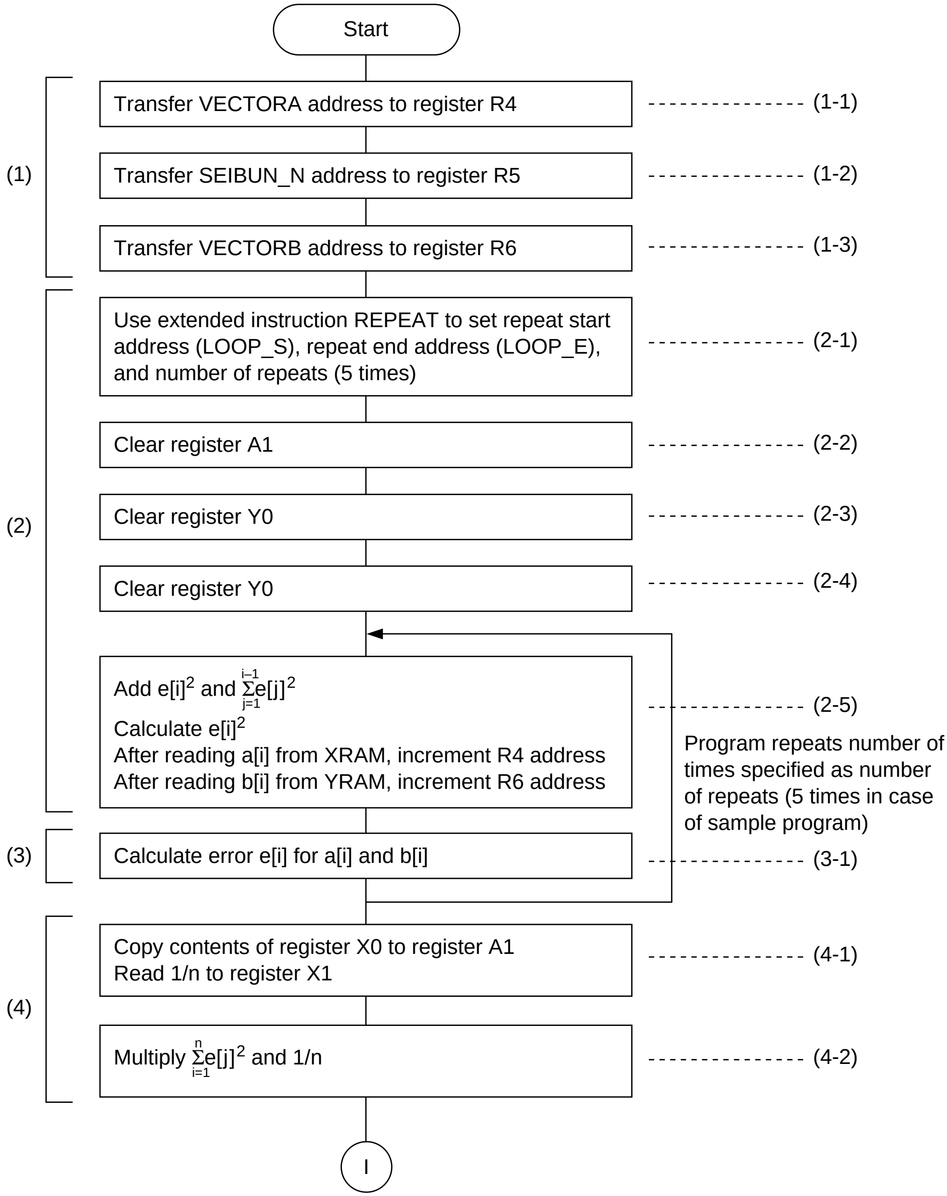
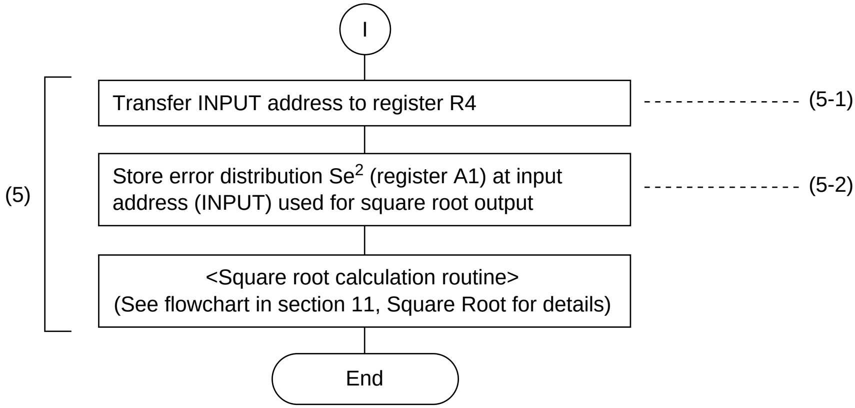


Figure 12.2

Flowchart


Start
(1)
Transfer VECTORA address to register R4
(1-1)
Transfer SEIBUN_N address to register R5
(1-2)
Transfer VECTORB address to register R6
(1-3)
(2)
Use extended instruction REPEAT to set repeat start address (LOOP_S), repeat end address (LOOP_E), and number of repeats (5 times)
(2-1)
Clear register A1
(2-2)
Clear register Y0
(2-3)
Clear register Y0
(2-4)
Add $e[i]^2$ and $\sum_{j=1}^{i-1} e[j]^2$
Calculate $e[i]^2$
After reading a[i] from XRAM, increment R4 address
After reading b[i] from YRAM, increment R6 address
(2-5)
Program repeats number of times specified as number of repeats (5 times in case of sample program)
(3)
Calculate error e[i] for a[i] and b[i]
(3-1)
(4)
Copy contents of register X0 to register A1
Read 1/n to register X1
(4-1)
Multiply $\sum_{i=1}^{n} e[j]^2$ and 1/n
(4-2)
I


I
Transfer INPUT address to register R4
(5-1)
(5)
Store error distribution Se^2 (register A1) at input address (INPUT) used for square root output
(5-2)
<Square root calculation routine>
(See flowchart in section 11, Square Root for details)
End

Main Program

The example program calculates the square mean error using three components {a[i], b[i] (i = 1, 2, 3)}

squ_ave.src

```
;*********************************************************************************************
;*                      Square mean routine
;*
;*                      a[i],b[i]
;*
;*********************************************************************************************
;*********************************************************************************************
;*                      Initial setting routine
;*********************************************************************************************
MAIN:
       MOV.L            #VECTORA,R4
       MOV.L            #SEIBUN_N,R5
       MOV.L            #VECTORB,R6

;*********************************************************************************************
;*                      Error distribution calculation routine
;*********************************************************************************************
       REPEAT LOOP_S,LOOP_E,#5                             ;Number of repeats is number of
                                                           vector a components + 2
       PCLR   A1
       PCLR   Y0
       PCLR   A0
LOOP_S:
       PADD   A0,Y0,Y0  PMULS   A1,A1,A0  MOVX.W @R4+,X0  MOVY.W @R6+,Y1 ;a[i],b[i]load
LOOP_E:
       PSUB   X0,Y1,A1

       PCOPY  Y0,A1                       MOVX.W @R5,X1     ;1/3 load
       PMULS  X1,A1,A1                                      ;0.33333 × Σ(a[i] - b[i])²

;*********************************************************************************************
;*                      Value to have square root calculated storage routine
;*********************************************************************************************
       MOV.L            #INPUT,R4
                                          MOVX.W A1,@R4   ;

;*********************************************************************************************
;*                      Square root calculation routine
;*********************************************************************************************
;*********************************************************************************************
;*                      Initial setting routine
```


RENESAS

```
;*****************************************************************************************
SEMI_MAIN:
        MOV.L           #EX_OUT,R5
        MOV.L           #DAT,R6
        MOV.L           #DAT2,R7

;*****************************************************************************************
;*                      Zero check of value to have square root calculated routine
;*****************************************************************************************
        MOV.W           @R4,R0
        CMP/EQ          #0,R0
        BF              ZERO_CH
                                     MOVX.W @R4,X0                ;H'0 load
        PCOPY  X0,A0                                              ;
        BRA             FIN                                       ;End of processing
        NOP

;*****************************************************************************************
;*                      Negative value check of value to have square root calculated routine
;*****************************************************************************************
ZERO_CH:
        SWAP            R0,R1
        SHAL            R1
        BF              MINUS_CH                                  ;If negative, do
following processing
                                     MOVX.W @R5,X0                ;H'FFFF load
        PCOPY           X0,A0
        BRA             FIN                                       ;End of processing
        NOP

;*****************************************************************************************
;*                      Comparison of value to have square root calculated and F'7FFB
routine
;*****************************************************************************************
MINUS_CH:
        MOV.W           @R4,R0                                    ;X load
        MOV.W           @R7,R1                                    ;H'7FFB load
        CMP/GT          R1,R0                                     ;R0 > R1 ?
        BF              EQU__SEL                                  ;If R1 is greater, jump
        MOV.L           #EX_OUT2,R5
                                     MOVX.W @R5,X0                ;X load
        PCOPY  X0,A0
        BRA             FIN
        NOP

;*****************************************************************************************
;*                      Approximation equation selection routine
```

```
;*****************************************************************************************
EQU_SEL:
        MOV.L           #DAT2,R7
        MOV.W           @R7,R1
        CMP/GT          R1,R0
        BF              Y0_PRO2

;*****************************************************************************************
;*                      Approximation equation (1) y0 calculation routine
;*****************************************************************************************
Y0_PRO1:
                                     MOVX.W @R4,X1  MOVY.W @R6+,Y0 ;Load input value X
                                                                    (value to have square
                                                                    root calculated) for use
                                                                    in calculating
                                                                    approximate square root
        MOV.W           @R4,R1                                      ;Keep input value X
                                                                    (value to have square
                                                                    root calculated) in R1
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A1                            MOVY.W @R6+,Y1 ;0.58579X,0.41422 load
        PADD    A1,Y1,A0                                            ;0.58579X+0.41422-> y0
        BRA             HIKAKU
        NOP

;*****************************************************************************************
;*                      Approximation equation (2) y0 calculation routine
;*****************************************************************************************
Y0_PRO2:
        MOV.L           #KINJI2,R6   MOVX.W @R4,X1  MOVY.W @R6+,Y0 ;Load input value X
                                                                    (value to have square
                                                                    root calculated) for use
                                                                    in calculating
                                                                    approximate square root
        MOV.W           @R4,R1                                      ;Keep input value X
                                                                    (value to have square
                                                                    root calculated) in R1
        MOV.L           #WORK,R4
        PMULS   X1,Y0,A0                                            ;0.79057 × X
        PSHA    #2,A0                                               ;(0.79057 × X) × 4

;*****************************************************************************************
;*                      Comparison of approximate square root and value to have square root
calculated routine/Part 1
;*****************************************************************************************
HIKAKU:
                                     MOVX.W A0,@R4                  ;Pass to CPU unit
        MOV.W           @R4,R0
        CMP/EQ          R0,R1                                       ;Approximate square root
                                                                    = input value X (value
                                                                    to have square root
                                                                    calculated)?
```

```
        BF              NOT_EQ
        PSHA    #-1,A0                          MOVY.W @R6,Y1   ;y0/2,0.5 load
        PADD    A0,Y1,A0                                        ;y0/2-0.5
        BRA             FIN
        NOP
;*******************************************************************************************
;*                      Comparison of approximate square root and value to have square root
calculated routine/Part 2
;*******************************************************************************************
NOT_EQ:
        CMP/GT          R0,R1
        BF              NOT_GT
                                MOVX.W @R5,X0                   ;H'FFFF load
        PCOPY   X0,A0
        BRA             FIN
        NOP
;

;*******************************************************************************************
;*                      Square root y calculation using gradualization equation routine
;*******************************************************************************************
NOT_GT:
        MOV.L           #3,R14                                  ;Set number of repeats
        MOV.L           #0,R13
LENEAR_LP:
        ADD             #1,R13                                  ;Increment counter

        MOV             R1,R11
        MOV.L           #0,R12
        REPEAT          DIV_S,DIV_E,#15
        DIV0U                                                   ;Signless initialization
DIV_S:
        DIV1            R0,R1                                   ;R1/R0
DIV_E:
        ROTCL           R12                                     ;Store T bit
        MOV.W           R12,@R4
                                MOVX.W @R4,X0
        PCOPY   X0,Y1
        PSHA    #-1,A0                                          ;y0/2
        PSHA    #-1,Y1                                          ;(X/y0)/2
        PADD    A0,Y1,A0
                                MOVX.W A0,@R4
        MOV.W           @R4,R0
        MOV             R11,R1

        CMP/GT          R14,R13
        BF              LENEAR_LP
```


RENESAS

```
FIN:    MOV.L           #OUTPUT,R7
                                                         MOVY.W A0,@R7   ;Store square root √X



EXIT:   BRA    EXIT
        NOP
MAIN_E: NOP
```

Data

```
;*********************************************************************************************
;*                           Square mean calculation data (XRAM/YRAM)
;*********************************************************************************************
             .SECTION XRAM,DATA,LOCATE=H'1000FF00
VECTERA:     .XDATA.W      0.5,0.125,0.5,0,0
SEIBUN_N:    .XDATA.W      0.33333                ;1/number of components (n)

;* For calculating square root *
INPUT:       .RES.W        1
WORK:        .RES.W        1
EX_OUT:      .DATA.W       H'FFFF
EX_OUT2:     .XDATA.W      1

             .SECTION YRAM,DATA,LOCATE=H'1001FF00
VECTERB:     .XDATA.W      0.25,0.0625,0.25,0,0

;; * For calculating square root *
KINJI1:      .XDATA.W      0.58579,0.41422,0.5    ;Approximation equation (1)
KINJI2:      .XDATA.W      0.79057                ;Approximation equation (2)
DAT1:        .DATA.W       H'7FFB
DAT2:        .XDATA.W      0.1
OUTPUT:      .RES.W        1
```

Section 13 Effects of DSP Instructions on Program Performance

The number of execution cycles required by each function program file is listed in tables 13.1 and 13.2.

The test conditions used for table 13.1 were as follows: an E8000 (SH7612) emulator was used, the main program of each program file was allocated to XRAM, and the data was allotted to XRAM and YRAM.

The test conditions used for table 13.2 were as follows: a simulator (SH-DSP) was used, the main program of each program file was allocated to XROM, and the data was allotted to XRAM and YRAM.

Table 13.1 Performance of Programs Employing DSP Instructions

Program Filename	Function	No. of Execution Cycles	Notes
pmuls32.src	32-bit multiplication	116	
tri_fun.src	Trigonometric function	62	
matrix.src	Matrix operation	238	3 × 3 matrix operation
in_pro.src	Inner product	15	3-dmensional space vectors
rout.src	Square root	104	
squ_ave.src	Square mean error	114	n = 3 (3 components)

Table 13.2 Performance of Programs Employing DSP Instructions

Program Filename	Function	No. of Execution Cycles	Notes
pmuls32.src	32-bit multiplication	172	
tri_fun.src	Trigonometric function	80	
matrix.src	Matrix operation	378	3 × 3 matrix operation
in_pro.src	Inner product	21	3-dmensional space vectors
rout.src	Square root	272	
squ_ave.src	Square mean error	292	n = 3 (3 components)

SH-DSP Software Application Note

Publication Date: 1st Edition, September 1999
Published by: Electronic Devices Sales & Marketing Group
Semiconductor & Integrated Circuits
Hitachi, Ltd.
Edited by: Technical Documentation Group
UL Media Co., Ltd.
Copyright © Hitachi, Ltd., 1999. All rights reserved. Printed in Japan.