RX Family

Using Multiply-Accumulate Operation Instructions

R01AN0254EJ0100
Rev.1.00
Jun 17, 2011

Introduction

This document describes the use of DSP function instructions on RX Family MCUs.

Target Device

RX Family

Contents

1. Introduction

The RX Family CPU core (RX) includes a 16-bit × 16-bit multiply-accumulate unit. When executing a 32-bit × 32-bit integer-multiply instruction (MUL instruction), which is normally used for arithmetic expressions or address calculations employing multiplication, the lower 32 bits of the 64-bit result of the 32-bit × 32-bit operation are taken as the calculation result. In other words, it is assumed when using the MUL instruction that the calculation result will not exceed 32 bits. However, there is usually valid data contained in the high-order bits of the result of a multiply or multiply-accumulate operation when fixed-point expressions are used for numeric data (see reference [1], for example). As a result, only a narrow range of numeric data can be handled when using the MUL instruction for multiply or multiply-accumulate operations involving numeric data using fixed-point expressions, because the MUL instruction can only be used when 32 bits are sufficient to express the calculation result. As a solution to this problem, the RX supports multiply-accumulate (and multiply) instructions that use a 48-bit accumulator and that can be executed in a single cycle, instructions that perform rounding operations on values stored in the accumulator, and transfer instructions for moving data between the accumulator and the general registers. By combining these multiply-accumulate operation instructions, rounding instructions, etc., a variety of operations involving numeric data using fixed-point expressions can be performed at high speed, resulting in data processing capacity rivaling that of a DSP. Explanations of the use of these instructions, interspersed with examples, are provided below. The description below covers multiply-accumulate operation instruction and related topics. For further details, refer to the *RX Family Software Manual*.

[1] Mori, Natori, Torii, *Iwanami kōza jōhōkagaku: 18 sūchi keisan* [Numerical Computation], pp. 1–27, Iwanami Shoten, 1982

2. Types of DSP Function Instructions

The RX provides DSP function instructions to take advantage of the 16-bit × 16-bit multiply-accumulate unit included in the CPU. Taking advantage of these instruction allows 16-bit × 16-bit multiply-accumulate operations to be performed efficiently. These instructions take two general-purpose register values as inputs and store the result in an accumulator. The four DSP function operation instructions are listed below.

- MULHI src, src2
- MULLO src, src2
- MACHI src, src2
- MACLO src, src2

Here, the letters “MUL” as the first three letters of the mnemonic indicate a multiply instruction, which is to say that the result of a multiply operation is stored in the accumulator. The letters “MAC” as the first three letters of the mnemonic indicate a multiply-accumulate instruction, which is to say that the result of a multiply operation is added to the accumulator value and the result is then stored in the accumulator. The letters “HI” as the final two letters of the mnemonic indicate that the multiply operation uses the upper (high-order) 16 bits of the register value designated by the mnemonic’s first operand as the multiplier and the upper 16 bits of the register value designated by the second operand as the multiplicand. In other words, a 16-bit × 16-bit multiply operation is indicated. The letters “LO” as the final two letters of the mnemonic indicate that the multiply operation uses the lower (low-order) 16 bits of the register value designated by the mnemonic’s first operand as the multiplier and the lower 16 bits of the register value designated by the second operand as the multiplicand. Figure 1 illustrates the functions of the above four instructions.

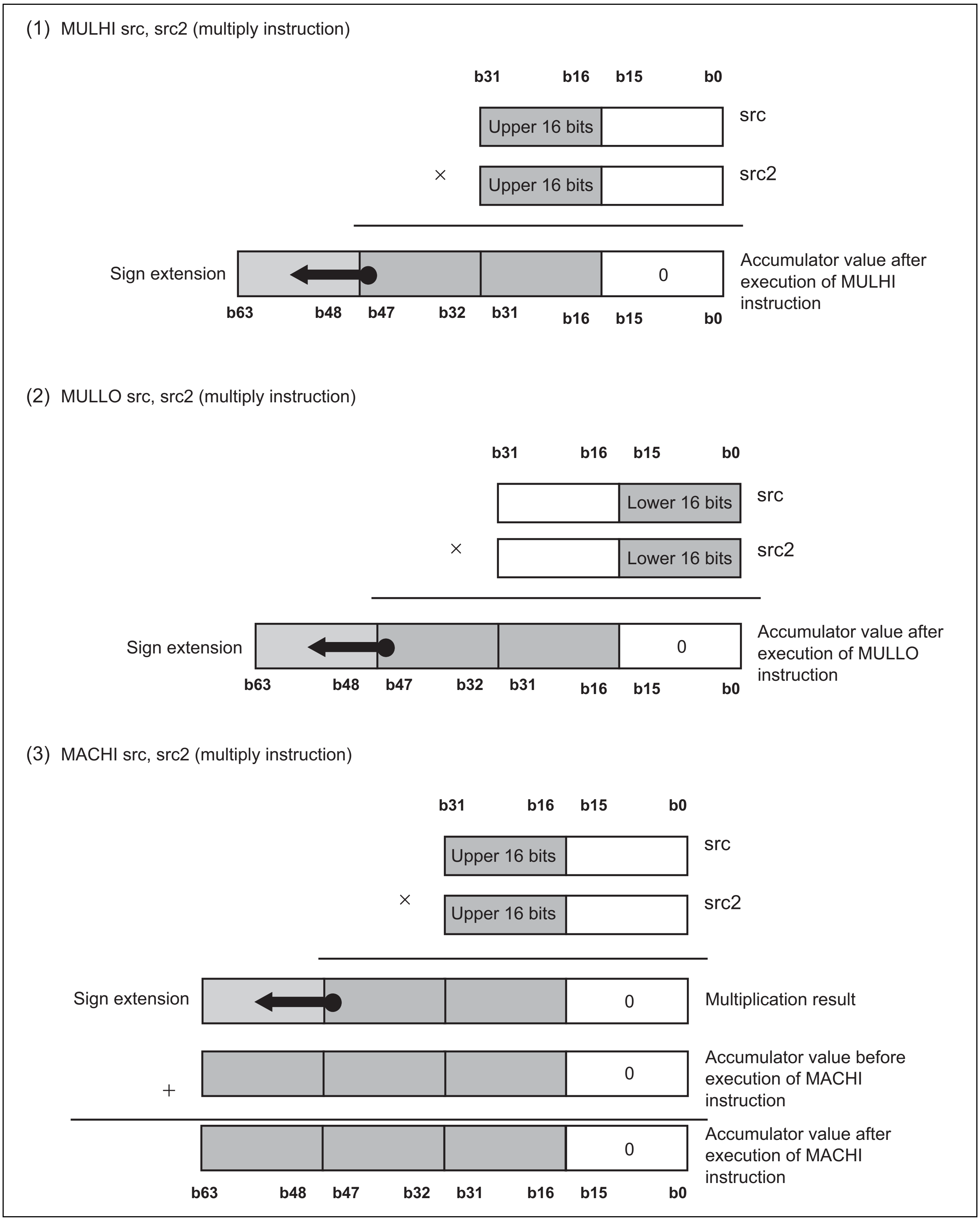

(1) MULHI src, src2 (multiply instruction)
b31 b16 b15 b0
Upper 16 bits
src
×
Upper 16 bits
src2
Sign extension
0
Accumulator value after execution of MULHI instruction
b63 b48 b47 b32 b31 b16 b15 b0
(2) MULLO src, src2 (multiply instruction)
b31 b16 b15 b0
Lower 16 bits
src
×
Lower 16 bits
src2
Sign extension
0
Accumulator value after execution of MULLO instruction
b63 b48 b47 b32 b31 b16 b15 b0
(3) MACHI src, src2 (multiply instruction)
b31 b16 b15 b0
Upper 16 bits
src
×
Upper 16 bits
src2
Sign extension
0
Multiplication result
+
0
Accumulator value before execution of MACHI instruction
0
Accumulator value after execution of MACHI instruction
b63 b48 b47 b32 b31 b16 b15 b0

Figure 1 Functions of 16-Bit × 16-Bit Multiply-Accumulate Operation Instructions

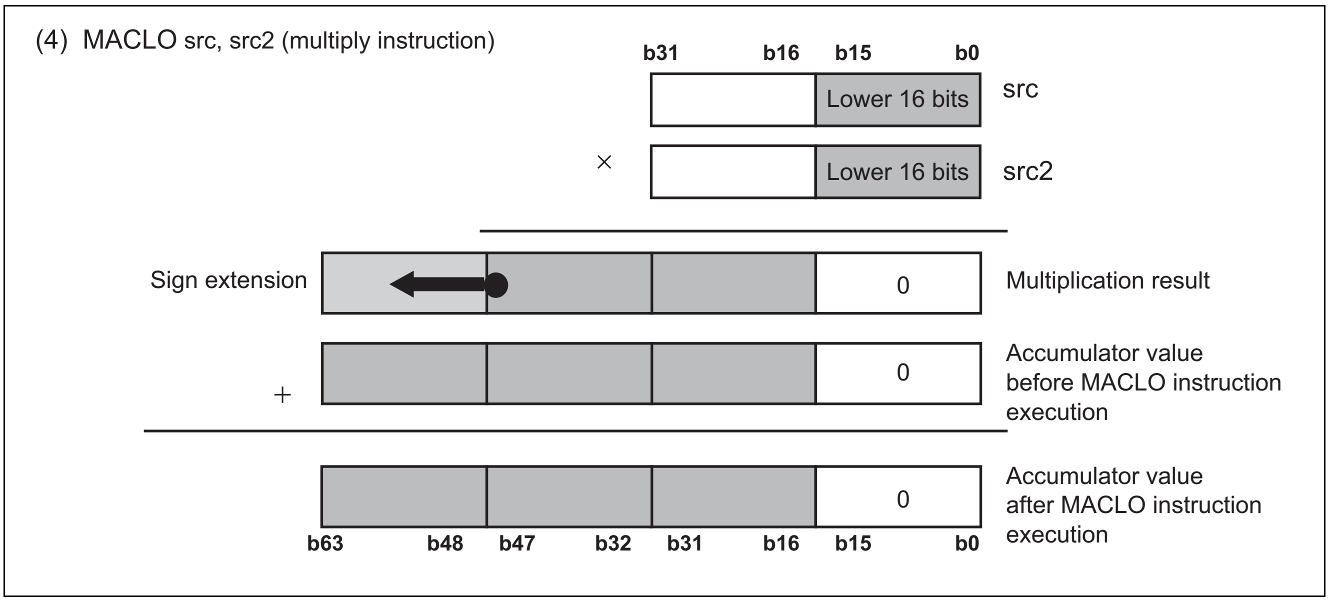

(4) MACLO src, src2 (multiply instruction)
b31
b16
b15
b0
Lower 16 bits
src
×
Lower 16 bits
src2
Sign extension
0
Multiplication result
+
0
Accumulator value before MACLO instruction execution
0
Accumulator value after MACLO instruction execution
b63
b48
b47
b32
b31
b16
b15
b0

Figure 1 Functions of 16-Bit × 16-Bit Multiply-Accumulate Operation Instructions (Continued)

3. Multiply-Accumulate Operation Instructions and Data Packing

The multiply-accumulate operation instructions supported by the RX use 16-bit words as the basic unit of data. In contrast, the bit length of the RX's general registers is 32 bits. Let us consider a case in which a transfer instruction is used to send a 16-bit word of data to a general register, after which it is multiplied by another unit of data that is already stored in a general register. The following expression

$x*a + y*b$

will be used to illustrate the use of multiply-accumulate operation instructions and data packing.

Perhaps we can employ the RX's multiply-accumulate operation instructions that use the upper 16 bits of general registers for multiply or multiply-accumulate operations in order to utilize the unused upper 16 bits of the registers. If we assign the upper and lower 16 bits of a 32-bit data value to 16-bit data values *x* and *y*, and the upper and lower 16 bits of a second 32-bit data value to 16-bit data values *a* and *b*, we can transfer the 32-bit data values to different general registers and use the MULWHI instruction to calculate the *x*a* portion of the expression. Storing a 32-bit data value as two 16-bit data values in this way is called data packing. It allows us to make effective use of the previously unused upper 16 bits of the general registers.

In addition, data packing makes it possible to use a longword data transfer instruction to transfer data to the registers, allowing two words of data can be transferred at once. Using this method, calculating the expression ($x*a + y*b$) can be performed with four instructions rather than the previous six.

```
mov.l  [Rs_x], Rsrc1   ; transfer x,y
mov.l  [Rs_a], Rsrc2   ; transfer a,b
mullo  Rsrc1, Rsrc2  ;  x*a (multiply-accumulate operation instruction)
maclo  Rsrc1, Rsrc2  ;  +y*b (multiply-accumulate operation instruction)
```

In other words, the 16-bit data values *x* and *y*, and *a* and *b*, respectively, are set in the high- and low-order bits of 32-bit data values, and 32-bit data transfer instructions are used, thereby eliminating two transfer instructions. Figure 2 shows the sequence of 16-bit × 16-bit multiply-accumulate operation instructions used to perform the multiply-accumulate operations. When using 16-bit × 16-bit multiply-accumulate operation instructions, it is important to keep in mind that the two multiplicands must either both be assigned to the high-order bits or both to the low-order bits of the general registers. Also note that the storage positions of *x* and *y* and of *a* and *b* will differ depending on the endian mode.

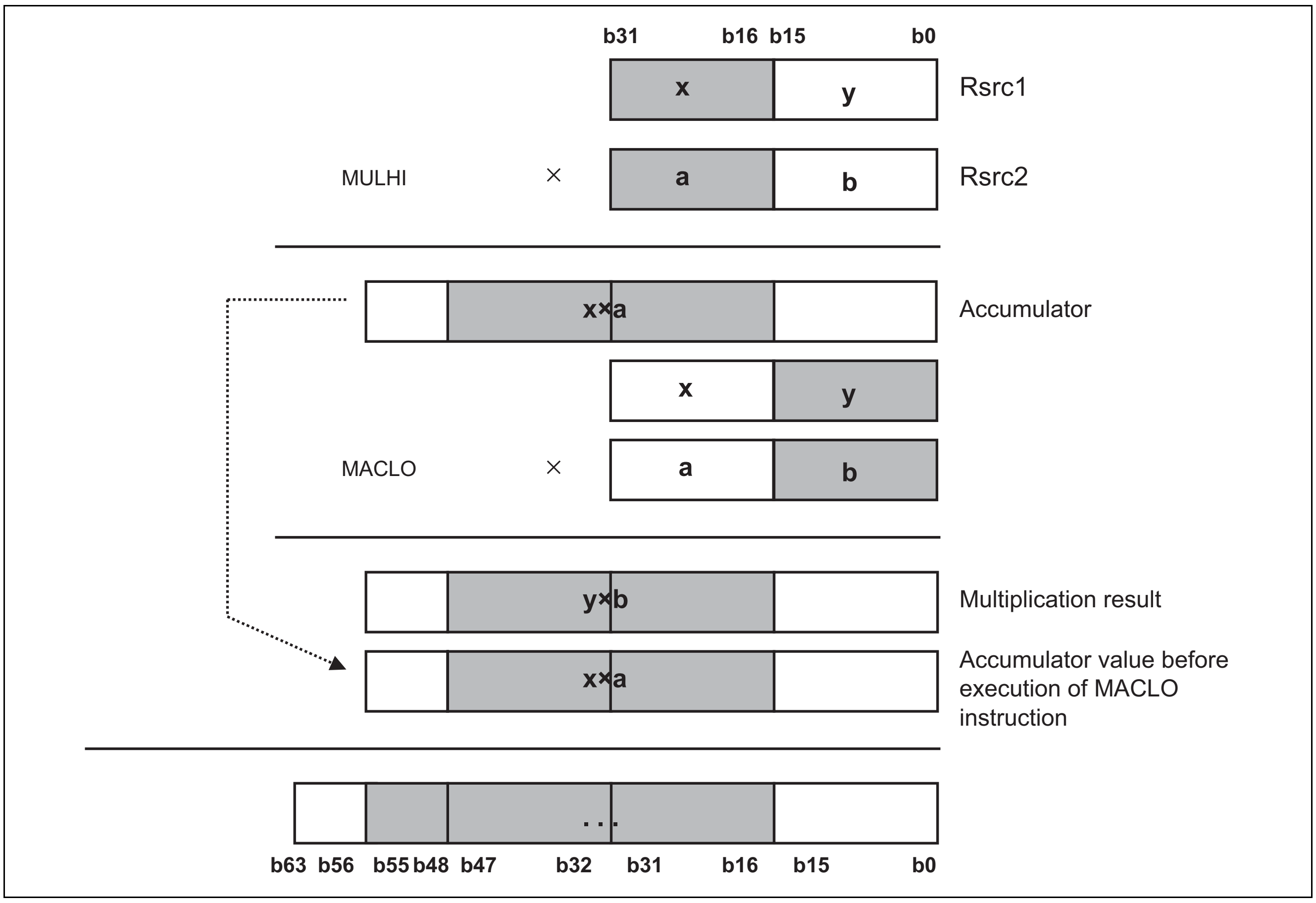


Figure 2 Multiply-Accumulate Operation Sequence Using Multiply-Accumulate Operation Instructions

4. Rounding

As we noted in the previous section, multiply-accumulate operation results are stored in the accumulator. The instructions MVFACHI (transfer from upper 32 bits of accumulator) and MVFACMI (transfer from middle 32 bits of accumulator) can be used to transfer to a general register the desired portion of the calculation result. For some calculations, it may be necessary to perform rounding on the multiply-accumulate operation result. The RX provides the 16-bit signed accumulator rounding instruction RACW (16-bit signed accumulator round) for such applications. The RACW instruction rounds off the value in the accumulator as shown in figure 3, after which the result can be extracted to a general register by using the MVFACHI instruction.

Before using rounding, it is necessary to have a good understanding of how rounding is performed by the RACW instruction. In other words, it is necessary to make sure that the result of rounding by the RACW instruction will match the expected calculation result. Figure 4 illustrates the processing performed by the RACW instruction.

- The accumulator value is rounded to word size, and that result is stored in the accumulator. This operation is illustrated below.

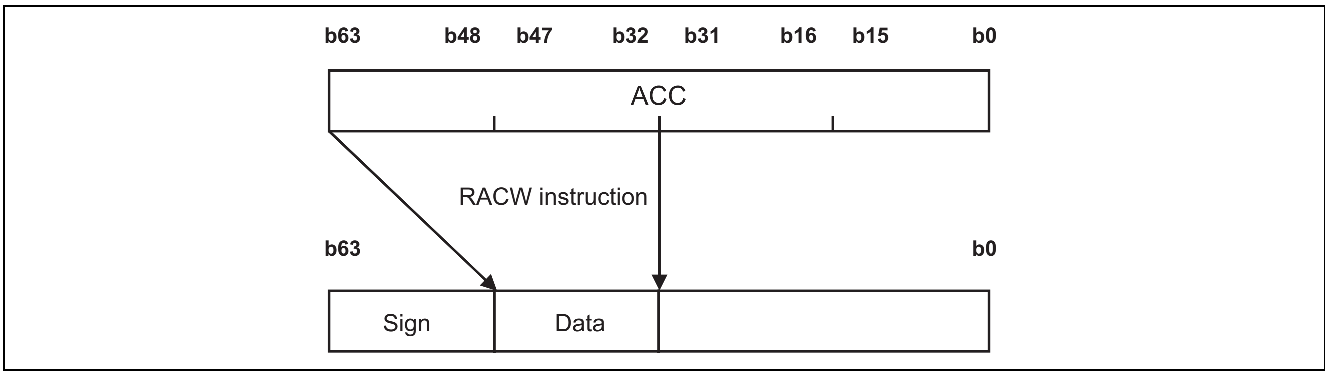


Figure 3 Outline of RACW Instruction Processing

- The RACW instruction uses the following procedures.
  Processing step 1. The accumulator value is left-shifted by the number of bits (1 or 2) specified by src.

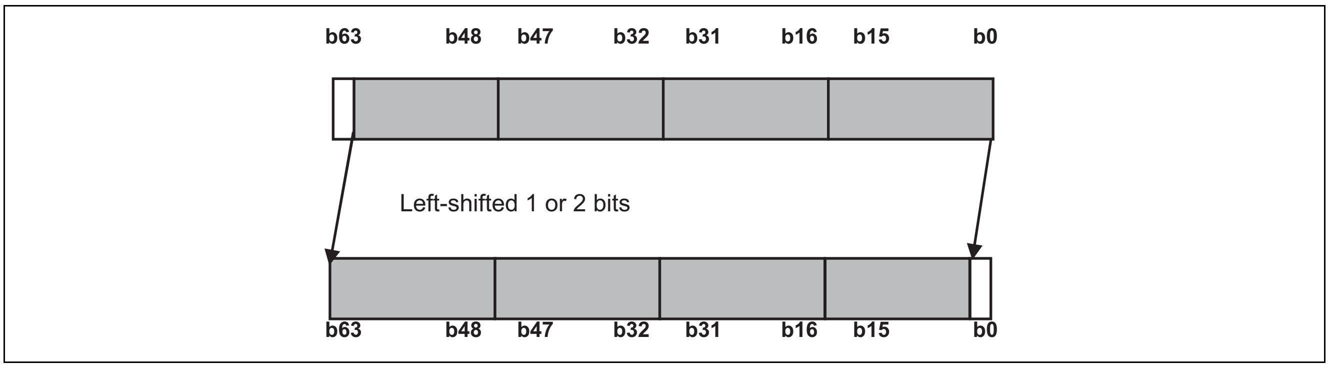


Figure 4 Rounding by RACW Instruction

Processing step 2. The accumulator value is changed according to the 1-bit or 2-bit left-shifted value.

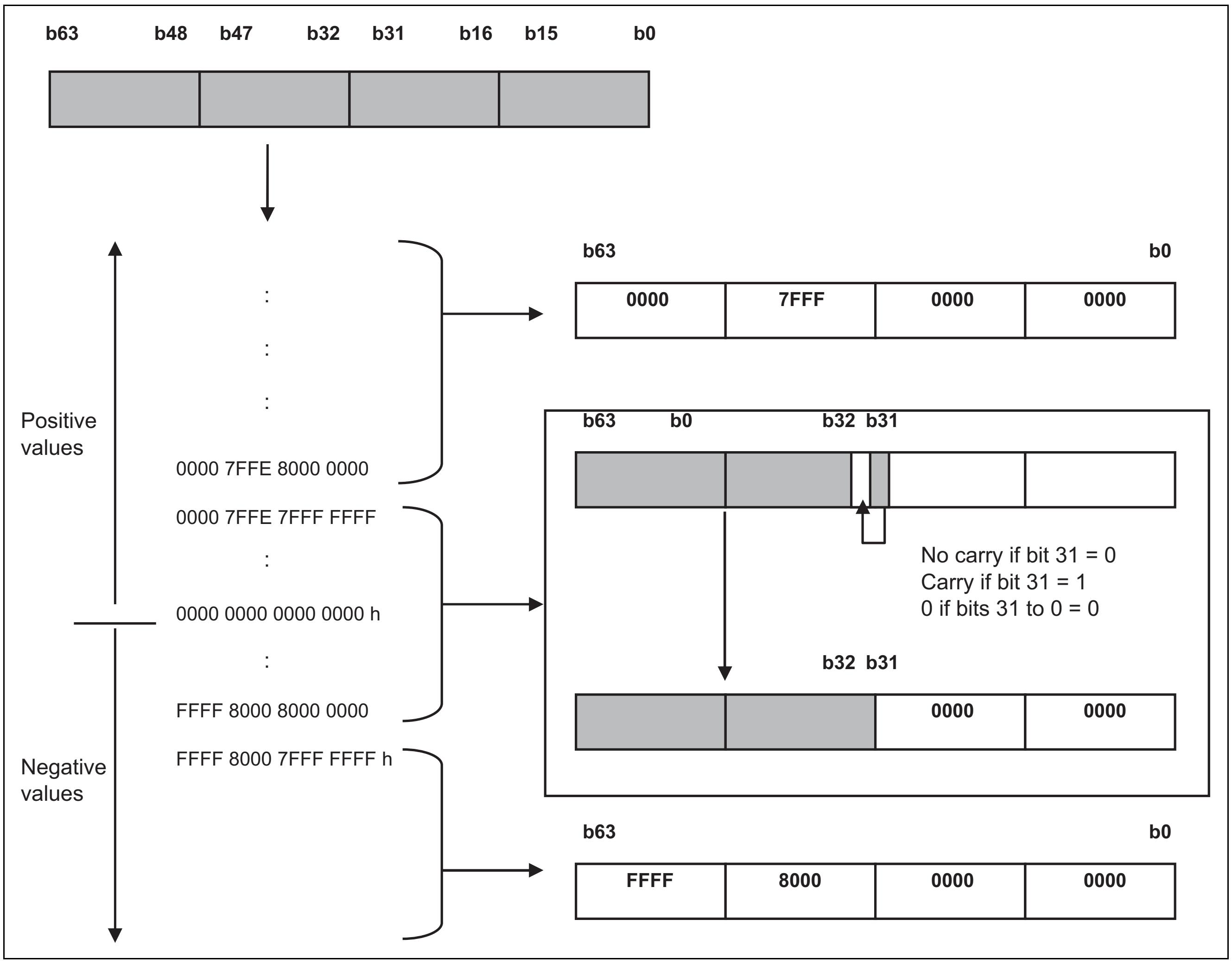


Figure 4 Rounding by RACW Instruction (Continued)

RACW src

Note that the above instruction causes the accumulator value to be left-shifted by the number of bits (1 or 2) specified by src, after which it is rounded according to the value of bit 31 (0 or 1).

5. Fixed-Point Expressions and Rounding

The RX rounding instruction RACW left-shifts the accumulator value by the number of bits (1 or 2) specified by src and then rounds the accumulator according to the value of bit 31. (See figure 4.) That is, when using the rounding instruction, the accumulator value that is the result of the multiply-accumulate operation instruction must be a fixed-point expression with the decimal point between bits 31 and 30 (when 1-bit shift is specified for the RACW instruction) or between bits 29 and 30 (when 2-bit shift is specified for the RACW instruction). Consequently, the 15 accumulator bits from bit 30 to bit 16 (when 1-bit shift is specified for the RACW instruction) or the 14 accumulator bits from bit 29 to bit 16 (when 2-bit shift is specified for the RACW instruction) are treated as the fractional part of the value. Since the multiplication result of a 16-bit × 16-bit multiply-accumulate operation instruction is stored in bits 47 to 16 of the accumulator, when using a rounding operation the decimal point of the multiplicand must be considered to ensure that the lower 15 or 14 bits of the multiplication result are the fractional portion of the value. Figure 5 shows examples of multiplicand expressions when using rounding. Specific examples are presented in the following section.

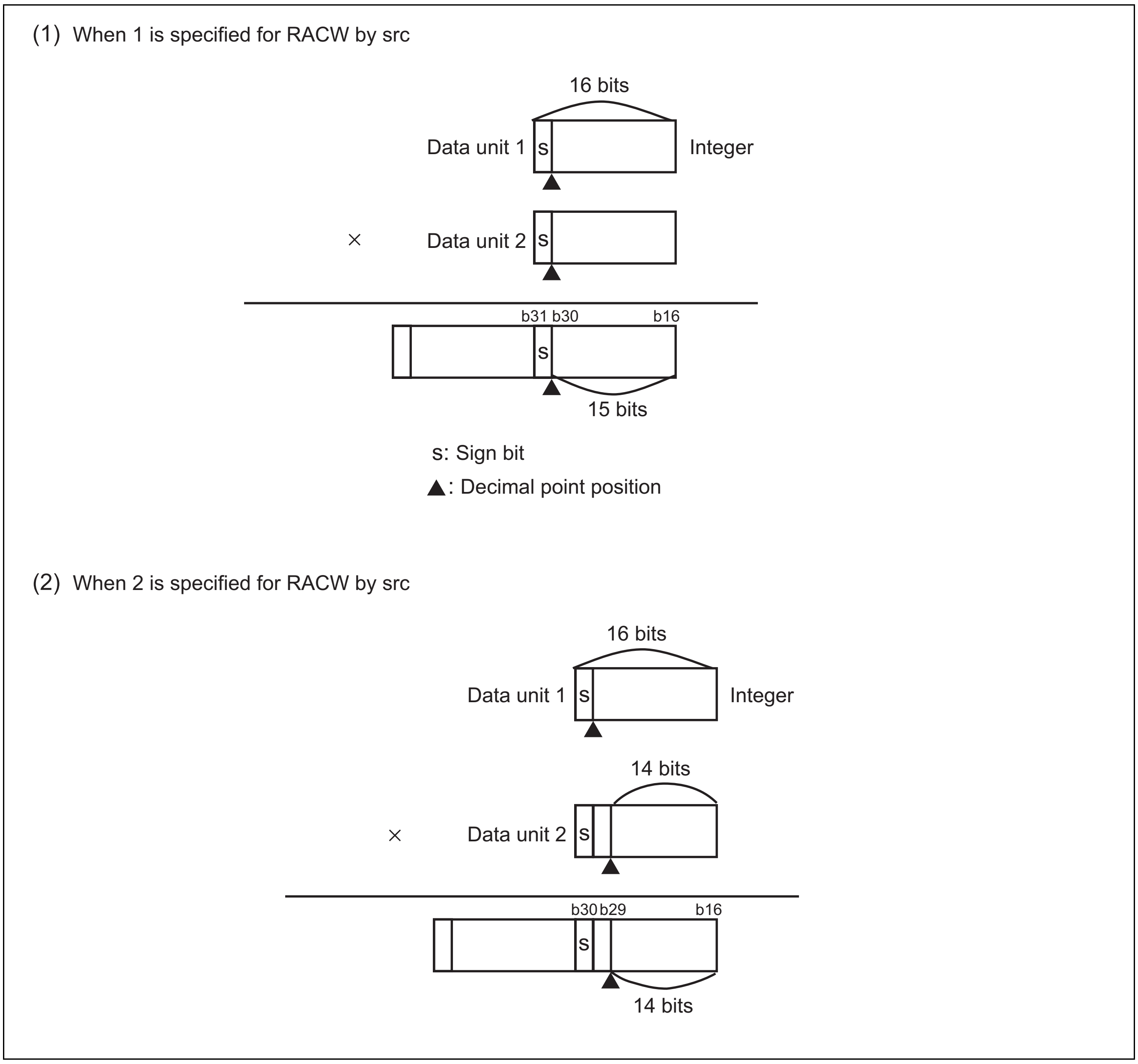


Figure 5 Example Multiplicand Expression when Using Rounding

6. Processing Sequence when Using the DSP Function Instructions

This section describes the basic processing sequence when using the DSP function instructions. In other words, a series of processes is described consisting of executing a multiply-accumulate operation using data represented as fixed-point expressions, performing rounding on the result, and transferring the desired data from the accumulator to a general register.

Multiply-Accumulate Operation Instructions

The following multiply-accumulate operation instructions are available for series processing. Namely, multiply-accumulate operation, rounding operation, and data transfer from the accumulator to a general register.

- MULHI (multiply instruction)
- MULLO (multiply instruction)
- MACHI (multiply-accumulate instruction)
- MACLO (multiply-accumulate instruction)
- RACW (accumulator rounding instruction)
- MVFACHI (accumulator to general register transfer instruction)
- MVFACMI (accumulator to general register transfer instruction)
- MVTACHI (general register to accumulator transfer instruction)
- MVTACLO (general register to accumulator transfer instruction)

6.1 Operation Sequence

When using multiply-accumulate operation instructions, the sequence of operations is as follows.

(1) Accumulator initialization
(2) Multiply-accumulate operation
(3) Rounding
(4) Transfer of calculation result from accumulator to general register

Figure 6 shows a typical multiply-accumulate operation sequence. In figure 6, operation (1) is the initial multiply operation using MULHI. This sets the multiplication result in the accumulator and causes the accumulator to be initialized. To set a specific value in the accumulator as the initial value rather than a multiplication result, it is necessary to transfer the contents of a general register to the accumulator by using the MVTACHI and MVTACLO instructions, thereby initializing the accumulator value. Operation (2), a multiply-accumulate operation, is executed only once in the figure, but of course it can be executed multiple times. Next, the sequence for performing rounding, according to the position of the decimal point, is shown in the figure. Operation (3) is not needed if it is not necessary to perform rounding, and the required data can simply be transferred from the accumulator to a general register. Note that when no rounding is performed it is necessary to choose either the MVFACHI or the MVFACMI instruction, according to whether the upper 32 bits or the middle 32 bits of the accumulator value is required.

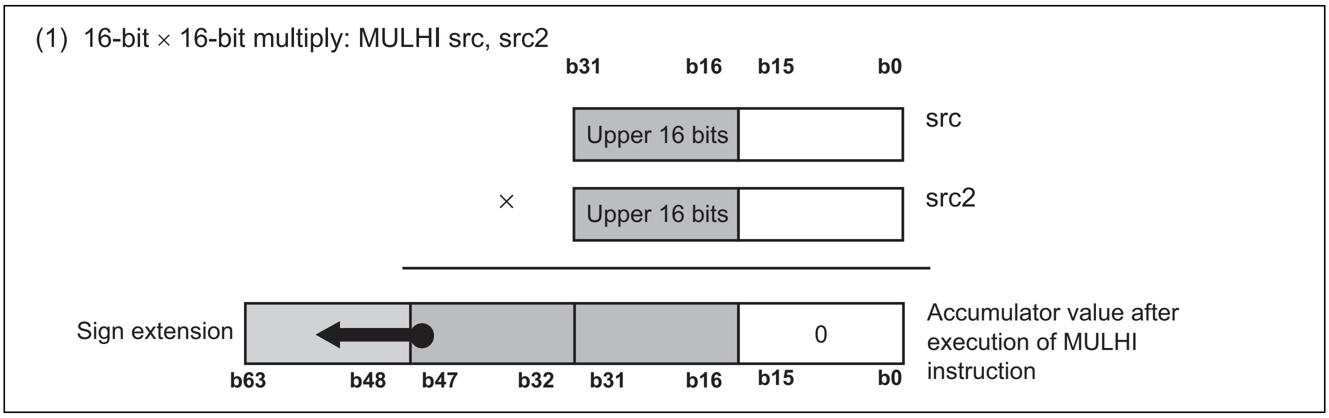

(1) 16-bit × 16-bit multiply: MULHI src, src2
b31
b16
b15
b0
Upper 16 bits
src
×
Upper 16 bits
src2
Sign extension
0
Accumulator value after execution of MULHI instruction
b63
b48
b47
b32
b31
b16
b15
b0

Figure 6 Operation Sequence Using Multiply-Accumulate Operation Instructions

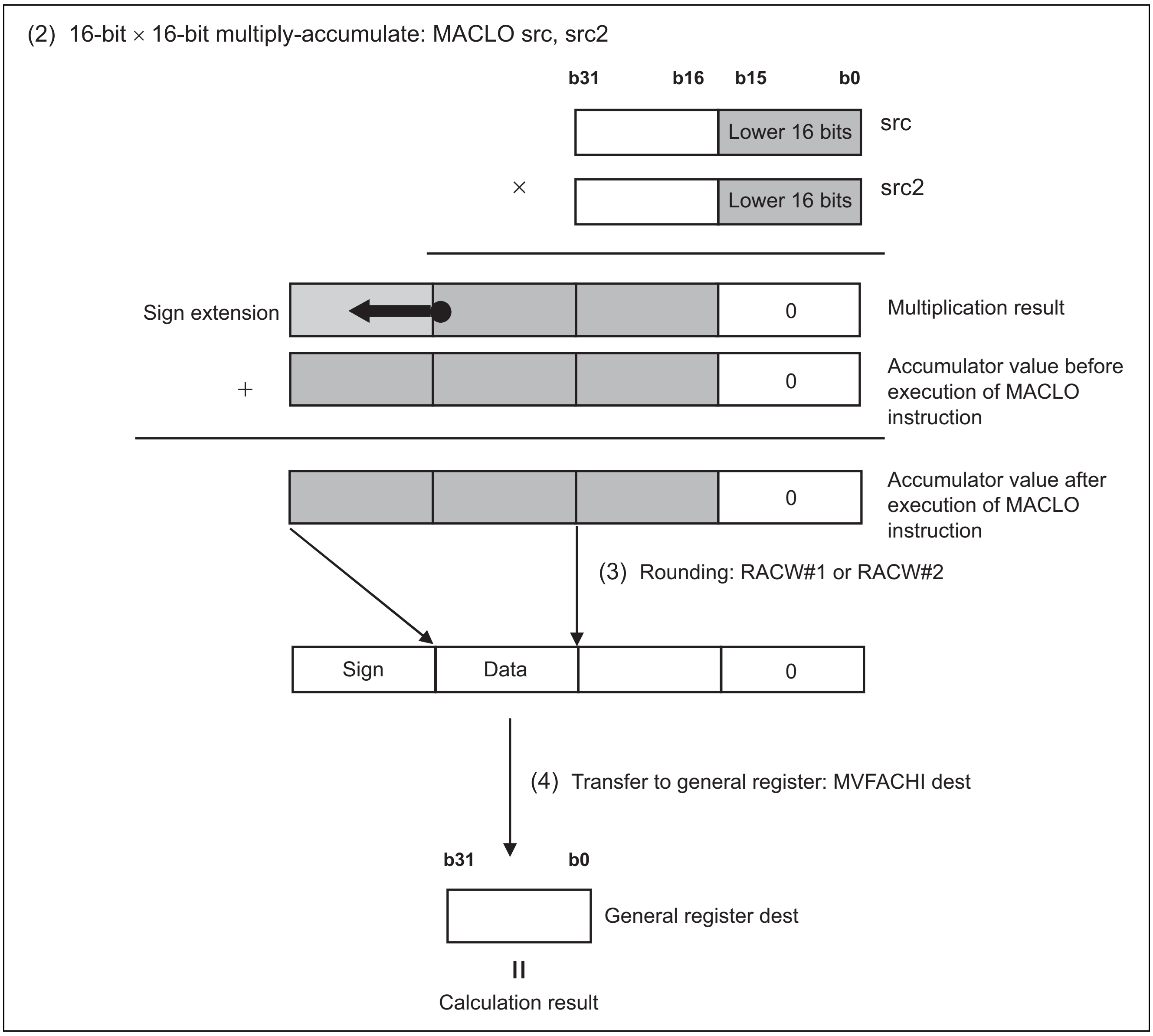

(2) 16-bit × 16-bit multiply-accumulate: MACLO src, src2
b31
b16
b15
b0
Lower 16 bits
src
×
Lower 16 bits
src2
Sign extension
0
Multiplication result
+
0
Accumulator value before execution of MACLO instruction
0
Accumulator value after execution of MACLO instruction
(3) Rounding: RACW#1 or RACW#2
Sign
Data
0
(4) Transfer to general register: MVFACHI dest
b31
b0
General register dest
II
Calculation result

Figure 6 Operation Sequence Using Multiply-Accumulate Operation Instructions (Continued)

7. Sample Program

The sample program described below uses multiply-accumulate operation instructions to perform calculations using matrix elements.

7.1 Multiplication of a Matrix of 3 Rows and 4 Columns by a Matrix of 4 Rows and 1 Column

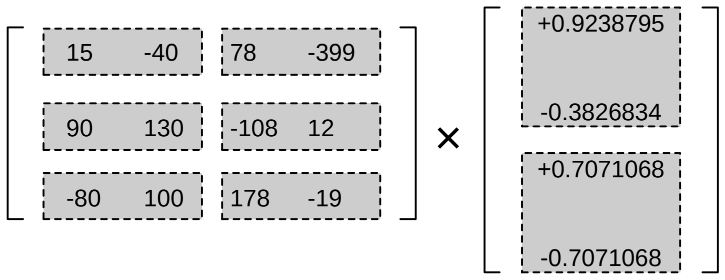
Below is described a program that multiplies a matrix consisting of 3 rows and 4 columns by a matrix consisting of 4 rows and 1 column.

$$\begin{pmatrix} 16 & -40 & 78 & -399 \\ 90 & 130 & -108 & 12 \\ -80 & 100 & 178 & -19 \end{pmatrix} \times \begin{pmatrix} +0.9238795 \\ -0.3826834 \\ +0.7071068 \\ -0.7071068 \end{pmatrix}$$

Since the elements of the matrices are represented by word size data, the elements of the 4 row by 1 column matrix must be fixed-point values. Since the data values are all within the range –1.0 to 1.0, we adopt a data representation in which we consider the decimal point to be between bits 15 and 14 of the 16 data bits and see bits 14 to 0 as the fractional part of the value. The data is expressed by multiplying the floating-point value by 2^15 to obtain a fixed-point data word. A value of +0.9238795, for example, becomes 0.9238795 × 2^15 = 30274, or 0x7642 in hexadecimal notation. All of the data values are converted into word size data in preparation for using the multiply-accumulate instruction in this way.

7.2 Data Packing

The rows times columns block calculation unit for this matrix multiplication is four words of data. That is, four multiply-accumulate operations are performed on data words. Since two words of data can be packed into a single 32-bit unit, the number of data transfers can be decreased. In addition, it is possible to make use of instructions for multiplying only the high-order bits, and only the low-order bits, respectively, of each set of 32-bit data units. In the following figure, the pairs of word-size data items packed into a single 32-bit unit are indicated by gray boxes.



7.3 Sample Program

A code listing of the sample program described in 7.1 and 7.2 above is presented on the following pages.

The union *matrix* represents a matrix consisting of 3 rows and 4 columns, and the union *vector* represents a matrix consisting of 4 rows and 1 column. The typedef *dataUnit* is initialized with data words, using the member *word*. When employing multiply-accumulate instructions, the member *longWord* is used to transfer data longwords, and it was defined for this purpose.

The function *matrixmultiply4* performs four multiply-accumulate calculations, performs rounding on the result, and returns the final result as its return value. In the sample program the function *matrixmultiply4* uses multiply-accumulate instructions. Since *matrixmultiply4* is an inline function written in assembly language, *#pragma inline_asm* is used.

```
/***********************************************************************/
/*                                                                     */
/*  FILE        :RXmatrix.c                                            */
/*  DATE        :Mon, Jul 19, 2010                                     */
/*  DESCRIPTION :Main Program                                          */
/*  CPU TYPE    :RX610                                                 */
/*                                                                     */
/*  This file is generated by Renesas Project Generator (Ver.4.50).    */
/*  NOTE:THIS IS A TYPICAL EXAMPLE.                                    */
/*                                                                     */
/***********************************************************************/

#include <stdint.h>

//#include "typedefine.h"
#ifdef __cplusplus
//#include <ios>                        // Remove the comment when you use ios
//_SINT ios_base::Init::init_cnt;       // Remove the comment when you use ios
#endif

void main(void);
#ifdef __cplusplus
extern "C" {
void abort(void);
}
#endif

#define CONST_BITS  15
#define ONE ((long int) 1)
#define CONST_SCALE (ONE << CONST_BITS)

#define MATRIX_SIZE (4)

typedef union {
  int16_t word[MATRIX_SIZE];
  int32_t longWord[MATRIX_SIZE/2];
} dataUnit;

dataUnit vector = {
  0x7642,       // +0.9238795
  0xcf05,       // -0.3826834
  0x5a82,       // +0.7071068
  0xa57f        // -0.7071068
};

#define COLUMN (3)

dataUnit matrix[COLUMN] = {
  {16, -40, 78, -399},
  {90, 130, -108, 12},
  {-80, 100, 178, -19},
};

#pragma inline_asm matrixmultiply4
int16_t matrixmultiply4(dataUnit *, dataUnit *);

void main(void)
{
  int i;
  int16_t results[COLUMN];

  for (i=0; i<COLUMN; ++i) {
    results[i] = matrixmultiply4(&matrix[i], &vector);
  }
  return;
}

int16_t matrixmultiply4(dataUnit *a, dataUnit *b)
{
    mov.l [r1+], r3    ; transfer 2 words of data
    mov.l [r2+], r4    ; transfer 2 words of data
    mulhi r3, r4
    maclo r3, r4
    mov.l [r1+], r3    ; transfer 2 words of data
    mov.l [r2+], r4    ; transfer 2 words of data
    machi r3, r4
    maclo r3, r4
    racw  #1     ; rounding
    mvfachi  r1
}

#ifdef __cplusplus
void abort(void)
{

}
#endif
```

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jun.17.11	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141