RX Family

R01AN0227EJ0100
Rev.1.00
Mar 14, 2011

Matrix Operation Programming with DSP Function Instructions

Introduction

This document describes matrix operation programming with DSP function instructions of the RX family.

Target Device

RX Family

Contents

1. General

The RX family CPU core (hereafter referred to as RX) incorporates a 16 × 16-bit multiply-accumulator. The result of executing a typical 32 × 32-bit integer multiplication instruction (MUL instruction) that is used for multiplicative expressions or address calculations is given by the lower 32 bits of the 64-bit result of multiplying two 32-bit numbers. Accordingly, it is assumed that the result of using an MUL instruction does not exceed 32 bits. However, when a numerical value is expressed as a fixed-point number (For example, refer to [1].), it is common that the valid data of the result of a multiplication or a multiply-accumulation is assigned to the upper bits. Therefore, if a multiplication or a multiply-accumulation of fixed-point numbers is carried out using a MUL instruction, the result must be within 32 bits and only a very limited range of numerical values can be dealt with. To solve this problem, the RX supports the instructions to perform the following: multiply-accumulation (or multiplication) by a 48-bit accumulator, rounding operation of the value stored in an accumulator, and data transfer between an accumulator and a general-purpose register. The combination of these multiply-accumulation and rounding operation instructions allows several high-speed operations on fixed-point numbers and data processing performance equal to DSPs. For details on the RX's multiply-accumulation instruction, refer to "RX Family User's Manual; Software" (REJ09B0435). The application note "How to Use Multiply-Accumulation Instruction" (R01AN0254EJ) explains how to use these multiply-accumulation and rounding operation instructions. In addition, the application note "How to Use Intrinsic Functions for Multiply-Accumulation" (R01AN0255EJ) explains how to use these multiply-accumulation and rounding operation instructions through intrinsic functions that are extended functions of the RX Family C/C++ compiler (hereafter referred to as compiler).

This application note describes matrix operation programming with the RX's multiply-accumulation instructions. Specifically, operations on N × N square matrices are to be carried out. The arithmetic operations discussed are addition, subtraction, multiplication, and inverse matrices. However, only the inverses of 2 × 2 square matrices are to be dealt with for the sake of clarity. Also, in program examples, the RX's multiply-accumulation instructions are used through compiler intrinsic functions (intrinsic functions supporting the multiply-accumulation instruction are available at compiler version 1.01 or later).

Note: [1] Mori, Natori, Torii; "Iwanami Koza Joho Kagaku-18 Suchi Keisan", pp.1-27, Iwanami Shoten, (1982)

2. Matrix Data Representation

This section describes matrix data representation. In this application note, operations on N × N square matrices are discussed, and in program examples these operations are carried out mainly with N = 2 or 4. Inversion of matrices is performed with N = 2, that is, 2 × 2 square matrices are to be handled.

Since the RX's multiply-accumulation instructions are intended for 16-bit signed data, an element of a matrix should be represented as 16-bit signed data and a matrix should be constructed from one-dimensional matrices of this 16-bit data. For example, a 4 × 4 square matrix should be represented as a short type one-dimensional array with 4 × 4 = 16 elements. Also, a 2 × 2 square matrix should be represented as a short type one-dimensional array with 2 × 2 = 4 elements. The definitions based on this approach are shown in the program example below.

```
#define N       4   /* Number of rows and columns of square matrix to be
handled */
typedef int16_t Matrix;
typedef int16_t Matrix44[4 * 4];  /* 4×4 square matrix */
typedef int16_t Matrix22[2 * 2];  /* 2×2 square matrix */
```

This matrix data representation allocates sequential addresses to matrix row vector elements. Note that in contrast, unordered addresses are allocated to matrix column vector elements. Examples of where each matrix element is placed within a one-dimensional array are provided below.

First, a 2 × 2 square matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The elements are stored in a one-dimensional array in the order below (from left to right).

$$a, b, c, d$$

Next, a 4 × 4 square matrix:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

The elements are stored in a one-dimensional array in the order below (from left to right).

$$a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p$$

Finally, some program examples are given.

$$A = \begin{bmatrix} 2 & -7 \\ -1 & 3 \end{bmatrix}$$

The matrix above corresponds to the program below.

```
Matrix22 A = {
     2, -7,
    -1,  3,
};
```

The matrix B corresponds to the program below.

$$B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ -1 & -2 & -3 & -4 \\ -5 & -6 & -7 & -8 \end{bmatrix}$$

```
Matrix44 B = {
     1,  2,  3,  4,
     5,  6,  7,  8,
    -1, -2, -3, -4,
    -5, -6, -7, -8,
};
```

3. Addition, Subtraction, and Multiplication of Matrices

This section describes addition, subtraction and multiplication of N × N square matrices.

3.1 Addition

Addition of matrices is the operation of adding up the corresponding elements of two matrices. For example, addition of 2 × 2 matrices is carried out as follows:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} + \begin{vmatrix} e & f \\ g & h \end{vmatrix} = \begin{vmatrix} a+e & b+f \\ c+g & d+h \end{vmatrix}$$

Shown below is matrix_add, an addition program for N × N square matrices. matrix_add takes two matrices a and b (both are the start addresses of arrays) as the arguments, and stores the result of adding these matrices into a.

```
/*
  Addition operation of N×N square matrices.
  The result is stored in a.
*/
void matrix_add(Matrix a[], Matrix b[])
{
    int i;

    for (i = 0; i < N * N; i++) {
        a[i] += b[i];
    }
}
```

3.2 Subtraction

Subtraction of matrices, similar to addition, is the operation of calculating the difference between the corresponding elements of two matrices. For example, subtraction of 2 × 2 square matrices is carried out as follows:

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} - \begin{vmatrix} e & f \\ g & h \end{vmatrix} = \begin{vmatrix} a-e & b-f \\ c-g & d-h \end{vmatrix}$$

Shown below is matrix_sub, a subtraction program for N × N square matrices. matrix_sub takes two matrices a and b (both are the start addresses of arrays) as the arguments, and stores the difference between a and b into a.

```
/*
  Subtraction operation of N×N square matrices.
  The result is stored in a.
*/
void matrix_sub(Matrix a[], Matrix b[])
{
    int i;

    for (i = 0; i < N * N; i++) {
        a[i] -= b[i];
    }
}
```

3.3 Multiplication

Multiplication of matrices requires rather complicated computations compared with those for addition and subtraction. A matrix multiplication is carried out by multiplying the left-side matrix's row vector elements by the right-side matrix's column vector elements in order and then adding up the results. Here, it is possible to make use of multiply-accumulation. For example, a multiplication of 2 × 2 square matrices is carried out as follows:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} a \times e + b \times g & a \times f + b \times h \\ c \times e + d \times g & c \times f + d \times h \end{bmatrix}$$

Since, here, multiplications of N × N square matrices are discussed, the resulting matrices are N × N square matrices as well. (In the example above, the resulting matrix is a 2 × 2 square matrix.)

Shown below is matrix_mul, a multiplication program for N × N square matrices. matrix_mul takes three matrices a, b, and res (all these are the start addresses of arrays.) as the arguments, and stores into res the result of multiplying a by b.

```
#include <machine.h>

/*
  Multiplication operation of N×N square matrices.
  The result is stored in res.
*/
void matrix_mul(Matrix a[], Matrix b[], Matrix res[])
{
    int i, j;
    int16_t *q = b;
    int16_t col[N];

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            col[j] = q[N * j];
        }
        q++;
        for (j = 0; j < N * N; j += N) {
            res[j] = (int16_t)macl(a + j, col, N);
        }
        res++;
    }
}
```

This program uses two means for achieving efficient computations. First, in order to use the RX's multiply-accumulation (intrinsic function macl), it allocates sequential addresses to the matrix column vector elements on the right of the multiplication symbol by copying them into the work array col, and calls the intrinsic function macl. Second, in order to efficiently use the column vector elements copied into col, it performs the outer multiplication loop in order of column vectors. If in contrast it performs the outer and inner loops in order of row and column vectors, respectively, it is necessary to copy the same column vector elements into a work array for each loop round.

Note that the multiplication program shown above is designed to work correctly with all N × N square matrices. If N is fixed at a certain small number, it is possible to rewrite the program to run it more quickly by unrolling its inner loop.

An example of rewriting it for N = 4 is given below. This example is designed for 4 × 4 matrices, and the program can be rewritten for 5 × 5 or 6 × 6 matrices in the same way.

```
#if N == 4
/*
  Multiplication of 4×4 square matrices (For N == 4).
  The result is stored in res.
*/
void matrix44_mul(Matrix a[], Matrix b[], Matrix res[])
{
    int i;
    int16_t *q = b;
    int16_t col[N];

    for (i = 0; i < N; i++) {
        col[0] = q[N * 0];
        col[1] = q[N * 1];
        col[2] = q[N * 2];
        col[3] = q[N * 3];
        q++;
        res[0 * 0] = (int16_t)macl(a + N * 0, col, N);
        res[N * 1] = (int16_t)macl(a + N * 1, col, N);
        res[N * 2] = (int16_t)macl(a + N * 2, col, N);
        res[N * 3] = (int16_t)macl(a + N * 3, col, N);
        res++;
    }
}
#endif /* N == 4 */
```

4. Computation of Inverse Matrices

This section describes computation of inverse matrices. Generally, it is not a simple matter to compute the inverse of a matrix. Particularly in an embedded system, it is not efficient to compute it depending upon circumstances. Therefore, calculate the inverses of matrices in advance as much as possible so that the application can do other necessary calculations by using the calculated inverses and carrying out multiplications.

For example, when A and B are matrices (or vectors), perform the following operation.

$$A^{-1} \times B$$

For the sake of efficiency, identify the matrix C in advance that satisfies the equation below, instead of computing the inverse of A in programs.

$$C = A^{-1}$$

Then, the application should carry out the simple multiplication below:

$$C \times B$$

Since there is a formula for the inverse of a 2 × 2 square matrix, a program example of computing the inverse of a 2 × 2 matrix is given below.

4.1 Computing the Inverse of a 2 × 2 Square Matrix

The 2 × 2 square matrix is given below.

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Then, A^{-1} is determined by following the formula:

$$\mathbf{A}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Note, however, that only if the following equation is satisfied,   is an invertible matrix.

$$ad - bc \neq 0$$

Shown below is matrix22_inv, a program that computes the inverse of a 2 × 2 square matrix using this formula. matrix22_inv computes the inverse of the 2 × 2 square matrix a, and stores it in res. On normal termination, matrix22_inv returns 0. If there is no inverse matrix, it returns -1. Note that since this program handles matrix elements as signed integers, the result may not be correct depending on the input data.

```
/*
  Computation of the inverse of a 2×2 square matrix.
  The result is stored in res, and a value 0 is returned.
  However, if there is no inverse matrix, -1 is returned.
 */
int matrix22_inv(Matrix a[], Matrix res[])
{
    int32_t det = (int32_t)a[0] * (int32_t)a[3] - (int32_t)a[1] *
(int32_t)a[2];

    if (det == 0) {
        return -1;      /* Error: no inverse matrix exists */
    }
    res[0] = (int16_t)(  a[3] / det);
    res[1] = (int16_t)(- a[1] / det);
    res[2] = (int16_t)(- a[2] / det);
    res[3] = (int16_t)(  a[0] / det);
    return 0;
}
```

5. Sample Program

Shown below is an example of using the matrix multiplication and inverse matrix computation programs that are described in this application note. In this example, a multiplication of 4 × 4 matrices is carried out, and the inverse of a 2 × 2 matrix is computed.

```
static Matrix44 a = {
     1,  2,  3,  4,
     5,  6,  7,  8,
    -1, -2, -3, -4,
    -5, -6, -7, -8,
};

static Matrix44 b = {
    5,  6,  7,  8,
    1,  2,  3,  4,
    5,  6,  7,  8,
    1,  2,  3,  4,
};

static Matrix22 c = {
     2, -7,
    -1,  3,
};

void main(void)
{
    Matrix44 d;
    Matrix22 e;

    /* The result of multiplying matrix a by matrix b is stored in d. */
    matrix_mul(a, b, d);
#if N == 4
    /* The result of multiplying matrix a by matrix b is stored in d (for 4×4).
*/
    matrix44_mul(a, b, d);
#endif
    /* The inverse of matrix c is stored in e. */
    matrix22_inv(c, e);
}
```

Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Mar 14, 2011	—	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.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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

© 2011 Renesas Electronics Corporation. All rights reserved.

Colophon 1.1