Using the uM-FPU V3.1 Matrix Operations

Micromega Corporation

Introduction

The uM-FPU V3 chip provides a number of instructions for operating on matrices and vectors. Matrices are defined as a group of sequential uM-FPU registers organized by rows and columns. For example, the following diagram shows a matrix of 2 rows and 4 columns.

	Col 0	Col 1	Col 2	Col 3
Row 0	0, 0	0, 1	0, 2	0, 3
Row 1	1, 0	1,1	1, 2	1, 3

Matrices are stored in sequential uM-FPU V3 registers in row major order. The following list shows the storage locations for a matrix of 2 rows and 4 columns starting at register 16.

Register 16	row 0, column 0
Register 17	row 0, column 1
Register 18	row 0, column 2
Register 19	row 0, column 3
Register 20	row 1, column 0
Register 21	row 1, column 1
Register 22	row 1, column 2
Register 23	row 1, column 3

Instruction Summary

The FPU has several special purpose instructions for working with matrices and vectors.

`SELECTMA`	Select matrix A
`SELECTMB`	Select matrix B
`SELECTMC`	Select matrix C
`LOADMA`	Load register 0 with value from matrix A
`LOADMB`	Load register 0 with value from matrix B
`LOADMC`	Load register 0 with value from matrix C
`SAVEMA`	Save register 0 value to matrix A
`SAVEMB`	Save register 0 value to matrix B
`SAVEMC`	Save register 0 value to matrix C
`MOP`	Matrix Operations

Since matrices and vectors are stored in FPU registers, any FPU instructions that reference registers can also be used to access values in a matrix or vector. Register X can be used for sequential access.

Matrix Operations

Matrix operations can involve one, two or three matrices which are referred to as MA, MB and MC. The SELECTMA, SELECTMB and SELECTMC instructions are used to select the registers that define a matrix. The starting register, the number of rows, and the number of columns are specified following the opcode. For example, the following instruction selects MB as a matrix with 2 rows and 4 columns starting at register 16.

```
SELECTMB,16,2,4
```

Register X is also set to the first register of the matrix by the SELECTMA, SELECTMB, and SELECTMC instructions. This makes it easy to use any of the register X instructions to quickly access the sequential registers in the matrix (e.g. READX, WRITEX, XSAVE, LOADX, etc.). The elements in a matrix can be accessed directly by using the specific register address for each element, or they can be accessed by row and column number using the LOADMA, LOADMB, LOADMC, SAVEMA, SAVEMB and SAVEMC instructions.

The LOADMA, LOADMB and LOADMC instructions load register 0 with the value from a selected element in the matrix. The following instruction loads register 0 with the value in row 1, column 2 of matrix MB (row and columns numbers start at 0).

```
LOADMB,1,2
```

The SAVEMA, SAVEMB and SAVEMC instructions store the value in register 0 to the selected element in the matrix. The following instruction saves the value in register 0 to row 1, column 2 of matrix MB.

```
SAVEMB,1,2
```

MOP instruction

The MOP (matrix operation) instruction performs all the matrix operations.

Scalar Operations

A scalar operation takes the single value in register 0 and applies it to each element of the matrix MA. For example, the Scalar Add operation using two 2x2 matrices, will perform the following:

MA[0, 0] = MA[0, 0] + reg[0]
MA[0, 1] = MA[0, 1] + reg[0]
MA[1, 0] = MA[1, 0] + reg[0]
MA[1, 1] = MA[1, 1] + reg[0]

Code	Operation	Description
0	Scalar Set	MA[row, col] = reg[0]
1	Scalar Add	MA[row, col] = MA[row, col] + reg[0]
2	Scalar Subtract	MA[row, col] = MA[row, col] + reg[0]
3	Scalar Subtract (reverse)	MA[row, col] = reg[0] - MA[r,c]
4	Scalar Multiply	MA[r,c] = MA[row, col] * reg[0]
5	Scalar Divide	MA[row, col] = MA[row, col] / reg[0]
6	Scalar Divide (reverse)	MA[row, col] = reg[0] / MA[r,c]
7	Scalar Power	MA[r,c] = MA[row, col] ** reg[0]

Element-wise Operations

An Element-wise operation performs each operation on corresponding elements of matrix MA and MB. For example, the Element-wise Add operation using two 2x2 matrices, will perform the following:

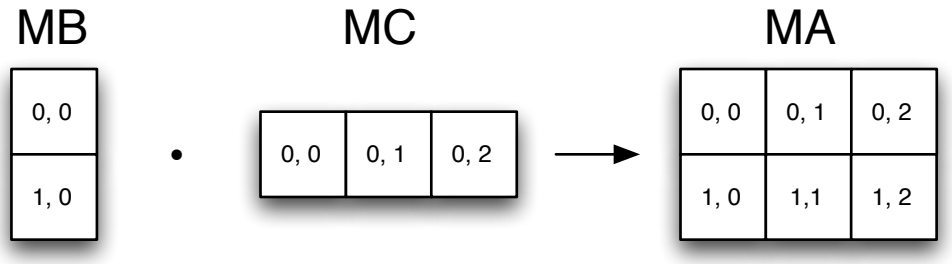
MA[0, 0] = MA[0, 0] + MB[0, 0]
MA[0, 1] = MA[0, 1] + MB[0, 1]

MA[1, 0] = MA[1, 0] + MB[1, 0]
MA[1, 1] = MA[1, 1] + MB[1, 1]

Code	Operation	Description
8	Element-wise Set	MA[row, col] = MB[row, col]
9	Element-wise Add	MA[row, col] = MA[row, col] + MB[row, col]
10	Element-wise Subtract	MA[row, col] = MA[row, col] - MB[row, col]
11	Element-wise Subtract (reverse)	MA[row, col] = MB[row, col] - MA[row, col]
12	Element-wise Multiply	MA[row, col] = MA[row, col] * MB[row, col]
13	Element-wise Divide	MA[row, col] = MA[row, col] / MB[row, col]
14	Element-wise Divide (reverse)	MA[r,c] = MB[row, col] / MA[r,c]
15	Element-wise Power	MA[row, col] = MA[row, col] ** MB[row, col]

Matrix Multiplication

The matrix multiplication performs a matrix multiply of MB times MC and stores the result in MA. The number of columns in MB must be the same as the number of rows in MC, or the multiply will not be done. The size of matrix MA will be updated to reflect the rows and columns of the resulting matrix.



Code	Operation	Description
16	Matrix Multiply	Calculate: MA = MB * MC

Identity and Diagonal Matrix

The identity operation stores the value 1.0 in all elements of matrix MA where the row and column numbers are the same, and stores 0.0 in all other elements. The following diagram shows 3x3 identity matrix:

	Col 0	Col 1	Col 2
Row 0	1.0	0.0	0.0
Row 1	0.0	1.0	0.0
Row 2	0.0	0.0	1.0

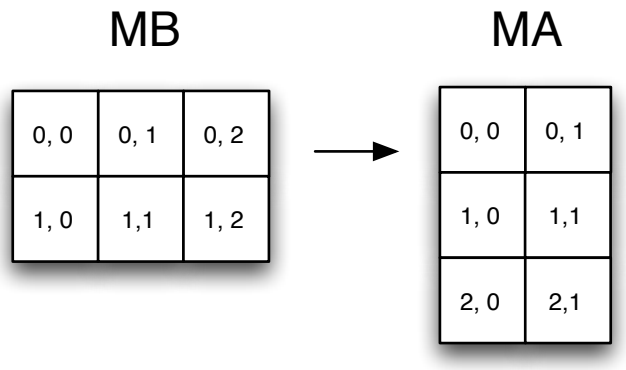
The diagonal operation stores the value contained in register 0 in all elements of matrix MA where the row and column numbers are the same, and stores 0.0 in all other elements.

Code	Operation	Description
17	Identity matrix	MA = identity matrix
18	Diagonal matrix	MA = diagonal matrix

Transpose

The transpose operation turns rows into columns and columns into rows. The following diagram shows the transpose of a 2x3 array to a 3x2 array. The size of matrix MA will be updated to reflect the rows and columns of the resulting matrix.

Code	Operation	Description
19	Transpose	MA = transpose MB



Statistics

The statistical operations provide a fast way of calculating values for a group of registers. The following example calculates the average value of registers 16 to 31.

```
SELECTMA,16,16,1        select MA as a 16x1 vector starting at register 16
MOP, 22                 calculate average value of elements in MA
```

Code	Operation	Description
20	Count	reg[0] = count of all elements in MA
21	Sum	reg[0] = sum of all elements in MA
22	Average	reg[0] = average of all elements in MA
23	Minimum	reg[0] = minimum of all elements in MA
24	Maximum	reg[0] = maximum of all elements in MA

Matrix copy

The copy operations provide a convenient way to copy the contents of one matrix to another. The size of the destination matrix will be updated to reflect the rows and columns of the resulting matrix. If there is not sufficient space at the destination, the copy will not be done.

Code	Operation	Description
25	CopyAB	Copy matrix A to matrix B.
26	CopyAC	Copy matrix A to matrix C.
27	CopyBA	Copy matrix B to matrix A.
28	CopyBC	Copy matrix B to matrix C.
29	CopyCA	Copy matrix C to matrix A.
30	CopyCB	Copy matrix C to matrix B.

Matrix Determinant

The determinant operation is only valid for 2x2 and 3x3 matrices. It returns the determinant in register 0.

Code	Operation	Description
31	Matrix Determinant	reg[0] = determinant of MA (2x2 or 3x3 matrices only)

Matrix Inverse

The inverse operation is only valid for 2x2 and 3x3 matrices. Matrix MA is set to the inverse of matrix MB.

Code	Operation	Description
32	Matrix Inverse	reg[0] = inverse of MA (2x2 or 3x3 matrices only)

Indexed Load Registers to Matrix

The indexed load register to matrix instructions can be used to quickly load a matrix by copying register values to a matrix. The byte immediately following the matrix operation specifies the number of index values to follow. An index value is a signed 8-bit integer specifying one of the registers from 0 to 127. If the index is positive, the value of the indexed register is copied to the matrix. If the index is negative, the absolute value is used as an index, and the negative value of the indexed register is copied to the matrix. Register 0 is cleared to zero before the register values are copied, so index 0 will always store a zero value in the matrix. The values are stored sequentially, beginning with the first register in the destination matrix.

Code	Operation	Description
33	Indexed Load Registers to MA	load register values to matrix MA
34	Indexed Load Registers to MB	load register values to matrix MB
35	Indexed Load Registers to MC	load register values to matrix MC

Example:

Suppose you wanted to create a 2-dimensional rotation matrix as follows:

cos A	-sin A
sin A	cos A

Assuming register 1 contains the value sin A, and register 2 contains the value cos A, the following instructions create the matrix.

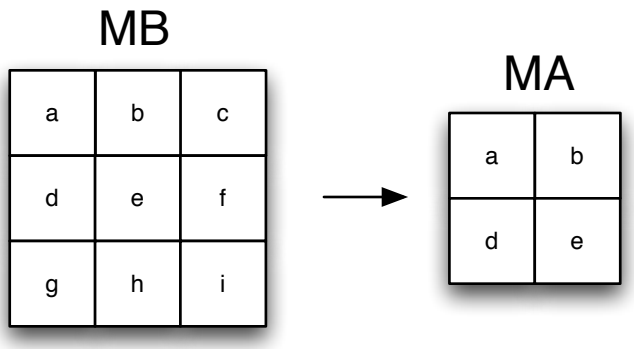
```
SELECTMA,10,2,2          select MA as a 2x2 matrix starting at register 10
MOP,33,4,2,-1,1,2        create the rotation matrix
```

Indexed Load Matrix to Matrix

The indexed load matrix to matrix instructions can be used to quickly copy values from one matrix to another. The byte immediately following the matrix operation specifies the number of index values to follow. An index value is a signed 8-bit integer specifying the offset of the desired matrix element from the start of the matrix. If the index is positive, the matrix element is copied to matrix MA. If the index is negative, the absolute value is used as an index, and the negative value of the matrix element is copied to the destination matrix. Register 0 is cleared to zero before the register values are copied, so index 0 will always store a zero value in matrix MA. The values are stored sequentially, beginning with the first register in matrix MA.

Code	Operation	Description
36	Indexed Load MB to MA	load matrix MB values to matrix MA
37	Indexed Load MC to MA	load matrix MC values to matrix MA

Example:
Suppose MB is a 3x3 array and you want to create a 2x2 array from the upper left corner as follows:



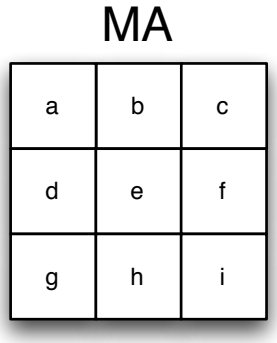

```
SELECTMB,20,2,2          select MB as a 2x2 matrix starting at register 20
MOP,36,4,0,1,3,4         copy the 2x2 subset from MA
```

Indexed Save Matrix to Register

The indexed save matrix to register instructions can be used to quickly extract values from a matrix. The byte immediately following the matrix operation specifies the number of index values to follow. An index value is a signed 8-bit integer specifying one of the registers from 0 to 127. The values are stored sequentially, beginning with the first element in matrix MA. If the index is positive, the matrix value is copied to the indexed register. If the index is negative, the matrix value is not copied.

Code	Operation	Description
38	Indexed Save MA to Registers	save matrix MA values to registers

Example:
Suppose matrix MA is a 3x3 matrix containing the following values:



The following instruction stores the value `a` to register 10, `e` to register 11 and `i` to register 12.

```
MOP,38,9,10,-1,-1,-1,11,-1,-1,-1,12      save matrix values to registers
```

Indexed Save Matrix to Matrix

The indexed save matrix to matrix instructions can be used to quickly extract values from a matrix. The byte immediately following the matrix operation specifies the number of index values to follow. An index value is a signed 8-bit integer specifying the offset of the desired matrix element from the start of matrix MA. The values are stored sequentially in the destination matrix, beginning with the first element in matrix MA. If the index is positive, the matrix value is copied to the destination matrix. If the index is negative, the matrix value is not copied.

Code	Operation	Description
39	Indexed Save MA to MB	save matrix MA values to matrix MB
40	Indexed Save MA to MC	save matrix MA values to matrix MC

uM-FPU V3 IDE support for the MOP instruction

The uM-FPU V3 IDE doesn't provide high level support for matrix operations, they must be specified using assembler. There are predefined symbols for the matrix operations that can be used with the `MOP` instruction. For example the following instructions initialize all elements of a 2x2 matrix to 1.0.

e.g.

```
#asm
      SELECTMA, 10, 2, 2
      LOADBYTE, 1
      MOP, SCALAR_SET
#endasm
```

A list of the predefined symbols for matrix operations are as follows:

```
0     SCALAR_SET          21    SUM
1     SCALAR_ADD          22    AVE
2     SCALAR_SUB          23    MIN
3     SCALAR_SUBR         24    MAX
4     SCALAR_MUL          25    COPYAB
5     SCALAR_DIV          26    COPYAC
6     SCALAR_DIVR         27    COPYBA
7     SCALAR_POW          28    COPYBC
8     EWISE_SET           29    COPYCA
9     EWISE_ADD           30    COPYCB
10    EWISE_SUB           31    DETERM
11    EWISE_SUBR          32    INVERSE
12    EWISE_MUL           33    ILOADRA
13    EWISE_DIV           34    ILOADRB
14    EWISE_DIVR          35    ILOADRC
15    EWISE_POW           36    ILOADBA
16    MULTIPLY            37    ILOADCA
17    IDENTITY            38    ISAVEAR
18    DIAGONAL            39    ISAVEAB
19    TRANSPOSE           40    ISAVEAC
20    COUNT
```

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU V3.1 floating point coprocessor, including:

uM-FPU V3.1 Datasheet
uM-FPU V3.1 Instruction Set