Using the uM-FPU64 Matrix Instructions

Release 407

Micromega Corporation

Introduction

The uM-FPU64 chip provides a number of instructions for operating on matrices and vectors. Matrices are defined as a group of sequential 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 32-bit floating point registers, or in RAM 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]

Operation	Instruction	Description
Scalar Set	`MOP, SCALAR_SET`	MA[row, col] = reg[0]
Scalar Add	`MOP, SCALAR_ADD`	MA[row, col] = MA[row, col] + reg[0]
Scalar Subtract	`MOP, SCALAR_SUB`	MA[row, col] = MA[row, col] - reg[0]
Scalar Subtract (reverse)	`MOP, SCALAR_SUBR`	MA[row, col] = reg[0] - MA[r,c]
Scalar Multiply	`MOP, SCALAR_MUL`	MA[r,c] = MA[row, col] * reg[0]
Scalar Divide	`MOP, SCALAR_DIV`	MA[row, col] = MA[row, col] / reg[0]
Scalar Divide (reverse)	`MOP, SCALAR_DIVR`	MA[row, col] = reg[0] / MA[r,c]
Scalar Power	`MOP, SCALAR_POW`	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:

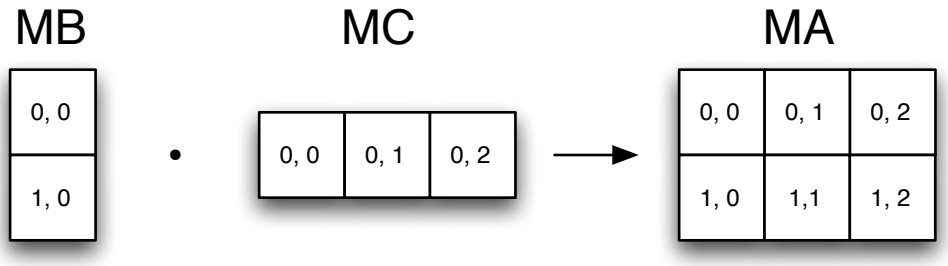
MA[0, 0] = MA[0, 0] + MB[0, 0]
MA[0, 1] = MA[0, 1] + MB[0, 1]

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

Operation	Instruction	Description
Element-wise Set	`MOP, EWISE_SET`	MA[row, col] = MB[row, col]
Element-wise Add	`MOP, EWISE_ADD`	MA[row, col] = MA[row, col] + MB[row, col]
Element-wise Subtract	`MOP, EWISE_SUB`	MA[row, col] = MA[row, col] - MB[row, col]
Element-wise Subtract (reverse)	`MOP, EWISE_SUBR`	MA[row, col] = MB[row, col] - MA[row, col]
Element-wise Multiply	`MOP, EWISE_MUL`	MA[row, col] = MA[row, col] * MB[row, col]
Element-wise Divide	`MOP, EWISE_DIV`	MA[row, col] = MA[row, col] / MB[row, col]
Element-wise Divide (reverse)	`MOP, EWISE_DIVR`	MA[r,c] = MB[row, col] / MA[r,c]
Element-wise Power	`MOP, EWISE_POW`	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.



Operation	Instruction	Description
Matrix Multiply	`MOP, 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

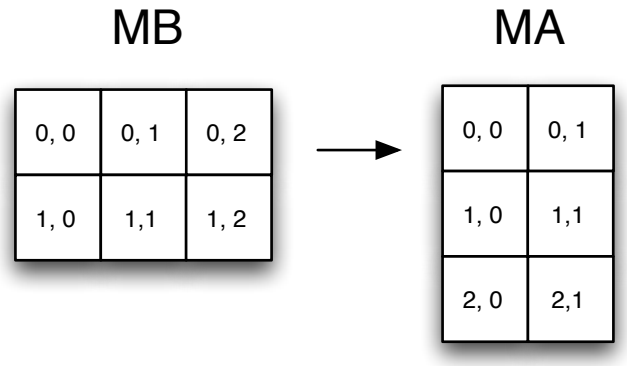
The diagonal operation stores the value contained in register 0 in all elements of matrix MA where the row and column numbers are the same, and stores 0.0 in all other elements.

Operation	Instruction	Description
Identity matrix	`MOP, IDENTITY`	MA = identity matrix
Diagonal matrix	`MOP, DIAGONAL`	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.

Operation	Instruction	Description
Transpose	`MOP, TRANSPOSE`	MA = transpose matrix



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, AVE                calculate average value of elements in MA
```

Operation	Instruction	Description
Count	`MOP, COUNT`	reg[0] = count of all elements in MA
Sum	`MOP, SUM`	reg[0] = sum of all elements in MA
Average	`MOP, AVE`	reg[0] = average of all elements in MA
Minimum	`MOP, MIN`	reg[0] = minimum of all elements in MA
Maximum	`MOP, MAX`	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.

Operation	Instruction	Description
Copy Matrix A to Matrix B	`MOP, COPY_AB`	Matrix B is set to a copy of matrix A.
Copy Matrix A to Matrix C	`MOP, COPY_AC`	Matrix C is set to a copy of matrix A.
Copy Matrix B to Matrix A	`MOP, COPY_BA`	Matrix A is set to a copy of matrix B.
Copy Matrix B to Matrix C	`MOP, COPY_BC`	Matrix C is set to a copy of matrix B.
Copy Matrix C to Matrix A	`MOP, COPY_CA`	Matrix A is set to a copy of matrix C.
Copy Matrix C to Matrix B	`MOP, COPY_CB`	Matrix B is set to a copy of matrix C.

Matrix Determinant

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

calculate the determinant of larger matrices use the LU decomposition or Cholesky decomposition matrix operations.

Instruction	Description
`MOP, DETERM`	reg[0] = determinant of MA

Matrix Inverse

The inverse operation is only valid for 2x2 and 3x3 matrices. Matrix MA is set to the inverse of matrix MB. To calculate the inverse of larger matrices use the LU decomposition or Cholesky decomposition matrix operations.

Instruction	Description
`MOP, INVERSE`	MA = inverse of MA

Load Registers to Matrix

The 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.

Instruction	Description
`MOP, LOAD_RA, byteCount, byte, ...`	Load matrix A from registers.
`MOP, LOAD_RB, byteCount, byte, ...`	Load matrix B from registers.
`MOP, LOAD_RC, byteCount, byte, ...`	Load matrix C from registers.

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.

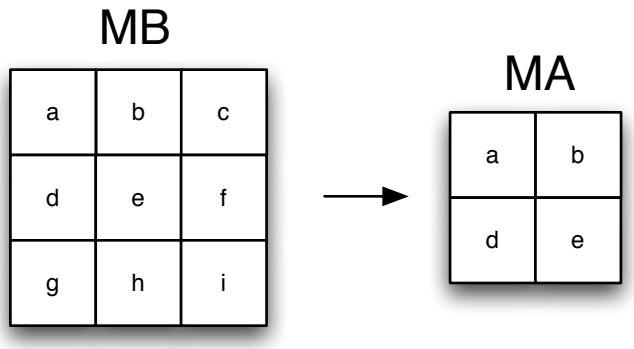
`SELECTMA,10,2,2`	select MA as a 2x2 matrix starting at register 10
`MOP,LOAD_RA,4,2,-1,1,2`	create the rotation matrix

Load Matrix to Matrix

The 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.

Instruction	Description
`MOP, LOAD_BA, byteCount, byte, ...`	Load matrix A from matrix B.
`MOP, LOAD_CA, byteCount, byte, ...`	Load matrix A from matrix C.

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



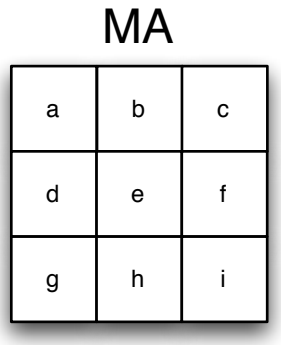
`SELECTMB,20,2,2`	select MB as a 2x2 matrix starting at register 20
`MOP,LOAD_BA,4,0,1,3,4`	copy the 2x2 subset from MA

Save Matrix A to Register

The 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.

Instruction	Description
`MOP, SAVE_AR, byteCount, byte, ...`	Save matrix A values to specified 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,SAVE_AR,9,10,-1,-1,-1,11,-1,-1,-1,12`	save matrix A values to registers

Save Matrix to Matrix

The 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.

Instruction	Description
`MOP, SAVE_AB, byteCount, byte, ...`	Save matrix A values to matrix B
`MOP, SAVE_AC, byteCount, byte, ...`	Save matrix A values to matrix C

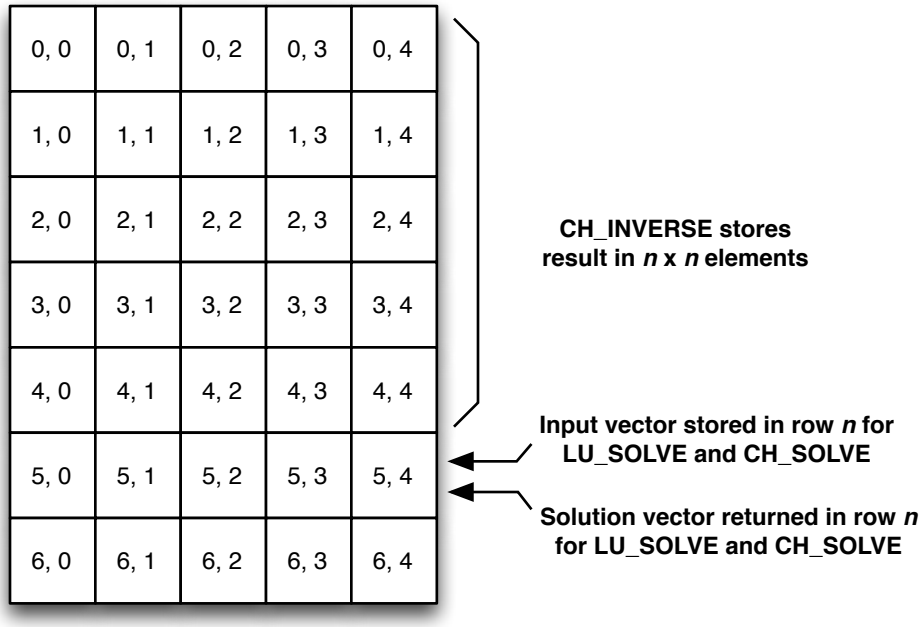
LU and Cholesky Decomposition

The LU and Cholesky decomposition operations can be used to calculate a matrix inverse, matrix determinant, and to solve sets of linear equations for *n* x *n* matrices of any size. The maximum size of matrix will be limited by the available registers or RAM for storing the matrices. An augmented matrix is created by the `MOP,LU_DECOMP` and `MOP,CH_DECOMP` instructions. When allocating matrix storage for matrix C prior to using these instructions, this additional space must be taken into account.

Original *n* x *n* matrix

0, 0	0, 1	0, 2	0, 3	0, 4
1, 0	1, 1	1, 2	1, 3	1, 4
2, 0	2, 1	2, 2	2, 3	2, 4
3, 0	3, 1	3, 2	3, 3	3, 4
4, 0	4, 1	4, 2	4, 3	4, 4

Augmented *n+2* x *n* matrix



Examples:

```
SELECTMC, 10, 5, 5          Select Matrix C
fill matrix with data

MOP, CH_DECOMP              Calculate Cholesky Inverse Matrix
MOP, CH_INVERSE
```

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU64 floating point coprocessor, including:

uM-FPU64Datasheet
uM-FPU64 Instruction Set