# RISC-V Matrix Specification

Version 0.5b, 07/2024: This document is in development.

# Table of Contents


Preamble . . . 1
Copyright and license information . . . 2
Contributors . . . 3
1. Introduction . . . 4
2. Implementation-defined Constant Parameters. . . . 5
3. Programmer’s Model . . . 6
3.1. Matrix Registers . . . 6
3.2. Matrix Type Register, mtype . . . 7
3.3. Matrix Tile Size Registers, mtilem/mtilek/mtilen . . . 9
3.4. Matrix Start Index Register, mstart . . . 9
3.5. Matrix Control and Status Register, mcsr . . . 9
3.6. Matrix Context Status in mstatus and sstatus . . . 10
4. Instructions . . . 11
4.1. Instruction Formats . . . 11
4.2. Configuration-Setting Instructions . . . 12
4.2.1. mtype Encoding . . . 14
4.2.2. ATM/ATK/ATN Encoding . . . 15
4.2.3. Constraints on Setting mtilem/mtilek/mtilen . . . 16
4.3. Load and Store Instructions . . . 17
4.3.1. Load Instructions . . . 17
4.3.2. Store Instructions . . . 19
4.3.3. Whole Matrix Load & Store Instructions . . . 20
4.4. Data Move Instructions . . . 21
4.4.1. Data Move Instructions between Matrix Registers . . . 21
4.4.2. Data Move Instructions between Matrix and Integer . . . 22
4.4.3. Data Move Instructions between Matrix and Float-point . . . 22
4.4.4. Data Broadcast Instructions . . . 23
4.4.5. Matrix Transpose Instructions. . . . 24
4.5. Arithmetic and Logic Instructions . . . 25
4.5.1. Matrix Multiplication Instructions . . . 25
4.5.2. Element-Wise Instructions . . . 26
4.6. Type-Convert Instructions . . . 28
5. Intrinsic Examples . . . 31
5.1. Matrix multiplication . . . 31
5.2. Matrix multiplication with left matrix transposed . . . 31
5.3. Matrix transpose without multiplication . . . 32
6. Standard Matrix Extensions . . . 33

6.1. Zma*b*: Matrix Mode Extension . . . 33
6.2. Zmi4: Matrix 4-bit Integer Extension . . . 33
6.3. Zmi8: Matrix 8-bit Integer Extension . . . 34
6.4. Zmi16: Matrix 16-bit Integer Extension . . . 35
6.5. Zmi32: Matrix 32-bit Integer Extension . . . 37
6.6. Zmi64: Matrix 64-bit Integer Extension . . . 38
6.7. Zmf8e4m3: Matrix 8-bit E4M3 Float Point Extension . . . 39
6.8. Zmf8e5m2: Matrix 8-bit E5M2 Float Point Extension . . . 40
6.9. Zmf8e3m4: Matrix 8-bit E3M4 Float Point Extension . . . 42
6.10. Zmf16e5m10: Matrix 16-bit Half-precision Float-point (FP16) Extension . . . 43
6.11. Zmf16e8m7: Matrix 16-bit BFloat (BF16) Extension . . . 44
6.12. Zmf32e8m23: Matrix 32-bit Float-point Extension . . . 45
6.13. Zmf19e8m10: Matrix 19-bit TensorFloat32 (TF32) Extension . . . 46
6.14. Zmf64e11m52: Matrix 64-bit Float-point Extension . . . 48
6.15. Zmv: Matrix for Vector operations . . . 49
6.15.1. Load Instructions . . . 50
6.15.2. Store Instructions . . . 50
6.15.3. Data Move Instructions . . . 51
6.15.4. Intrinsic Example: Matrix multiplication fused with element-wise vector operation . . . 52
6.16. Zmi2c: Im2col Extension . . . 53
6.16.1. CSRs . . . 53
6.16.2. Configuration Instructions . . . 55
6.16.3. Load Unfold Instructions . . . 55
6.16.4. Intrinsic Example: Conv2D . . . 55
6.16.5. Intrinsic Example: Conv3D . . . 57
6.16.6. Intrinsic Example: MaxPool2D . . . 58
6.16.7. Intrinsic Example: AvgPool2D . . . 59
6.17. Zmc2i: Col2im Extension . . . 60
6.17.1. CSRs . . . 60
6.17.2. Configuration Instructions . . . 60
6.17.3. Store Fold Instructions . . . 60
6.18. Zmsp*: Matrix Sparsity Extension . . . 60
6.18.1. Configuration Instructions . . . 61
6.18.2. Matrix Multiplication Instructions . . . 61
7. Matrix Instruction Listing . . . 64

# Preamble

*This document is in the Development state*

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

# Copyright and license information

# Contributors

This RISC-V specification has been contributed to directly or indirectly by:

- Chaoqun Wang
- Chen Chen
- Fujie Fan
- Hui Yao
- Jing Qiu
- Kening Zhang
- Kun Hu
- Mingxin Zhao
- Wengan Shao
- Wenmeng Zhang
- Xianmiao Qu
- Xiaoyan Xiang
- Xin Ouyang
- Xin Yang
- Yunhai Shang
- Zhao Jiang
- Zhiqiang Liu
- Zhiwei liu
- Zhiyong Zhang
- **Your name here**

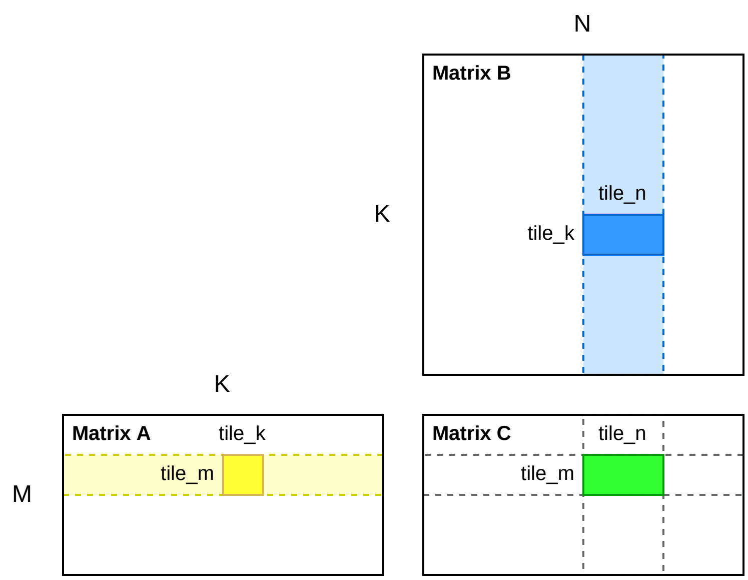
We will be very grateful to the huge number of other people who will have helped to improve this specification through their comments, reviews, feedback and questions.

# Chapter 1. Introduction

This document describes the matrix extension for RISC-V.

Matrix extension implement matrix multiplications by partitioning the input and output matrix into tiles, which are then stored to matrix registers.

Tile size usually refers to the dimensions of these tiles. For the operation C = AB in figure below, the tile size of C is mtilem × mtilen, the tile size of A is mtilem × mtilek and the tile size of B is mtilek × mtilen.




Each matrix multiplication instruction computes its output tile by stepping through the K dimension in tiles, loading the required values from the A and B matrices, and multiplying and accumulating them into the output.

Matrix extension is strongly inspired by the RISC-V Vector "V" extension.

# Chapter 2. Implementation-defined Constant Parameters

Each hart supporting a matrix extension defines four parameters:

1. The maximum size in bits of a matrix element that any operation can produce or consume, $\mathrm{ELEN} \geq 8$, which must be a power of 2.
2. The number of bits in a single matrix tile register, MLEN, which must be a power of 2, and must be no greater than $2^{32}$.
3. The number of bits in a row of a single matrix tile register, RLEN, which must be a power of 2, and must be no greater than $2^{16}$.
4. The multiple of length for matrix accumulation registers, AMUL, where the number of bits in a row of a single matrix accumulation register is $\mathrm{RLEN} \times \mathrm{AMUL}$, and the number of bits in a single matrix accumulation register is $\mathrm{MLEN} \times \mathrm{AMUL}$.

Some constraints on these parameters are defined as following.

1. $\mathrm{ELEN} \leq \mathrm{RLEN} \leq \mathrm{MLEN}$, this supports matrix tile size from $1 \times 1$ to $2^{16} \times 2^{16}$.
2. For implmentations without widening accumulation space, $\mathrm{AMUL} = 1$.
3. For implmentations with double-widening accumulation space, $\mathrm{AMUL} = 2$.
4. For implmentations with quadruple-widening accumulation space, $\mathrm{AMUL} = 4$.
5. For implmentations with octuple-widening accumulation space, $\mathrm{AMUL} = 8$.
6. AMUL with any other value is not allowed.

# Chapter 3. Programmer's Model

The matrix extension adds 8 unprivileged CSRs and 16 matrix registers to the base scalar RISC-V ISA.

*Table 1. Matrix CSRs*

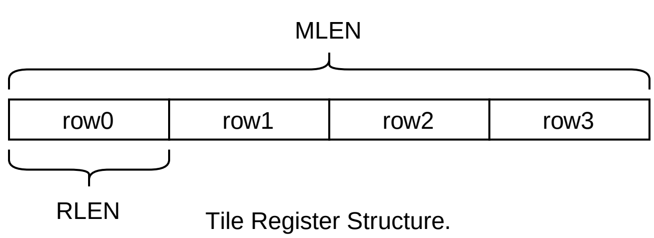
| Address | Privilege | Name | Description |
|---|---|---|---|
| 0xC40 | URO | mtype | Matrix tile data type register. |
| 0xC41 | URO | mtilem | Tile length in m direction. |
| 0xC42 | URO | mtilen | Tile length in n direction. |
| 0xC43 | URO | mtilek | Tile length in k direction. |
| 0xC44 | URO | mlenb | MLEN/8 (matrix tile register length in bytes). |
| 0xC45 | URO | mrlenb | RLEN/8 (matrix tile register row length in bytes). |
| 0xC46 | URO | mamul | AMUL. |
| 0x040 | URW | mstart | Start element index. |
| 0x041 | URW | mcsr | Matrix control and status register. |

## 3.1. Matrix Registers

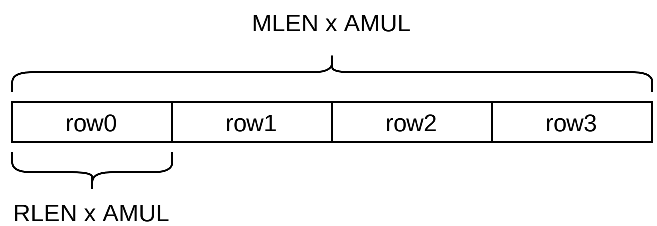
The matrix extension adds 8 architectural **Tile Registers** (tr0-tr7) for input tile matrices and 8 architectural **Accumulation Registers** (acc0-acc7) for output accumulation matrices.

A **Tile Register** has a fixed MLEN bits of state, where each row has RLEN bits. As a result, there are MLEN/RLEN rows for each tile register in logic.

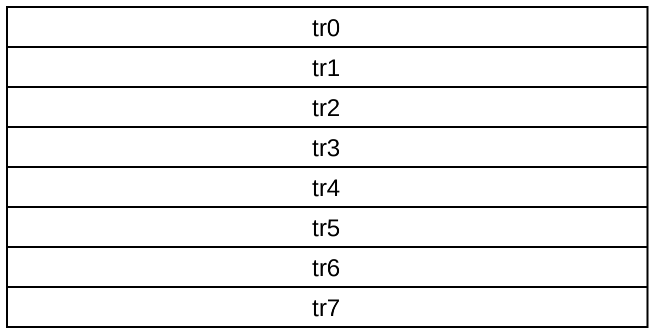
An **Accumulation Register** has a fixed $\mathrm{MLEN} \times \mathrm{AMUL}$ bits of state, where each row has $\mathrm{RLEN} \times \mathrm{AMUL}$ bits. As a result, there are MLEN/RLEN rows for each accumulation register in logic.



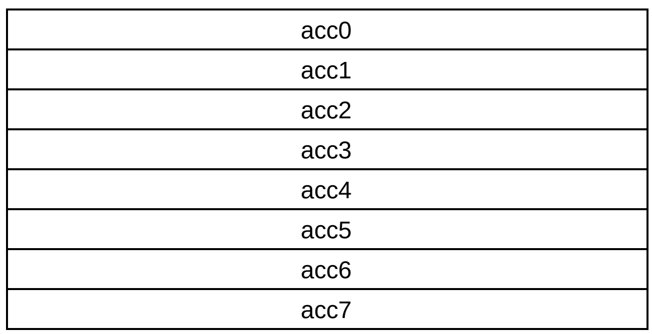

Tile Register Structure.




Accumulation Register Structure.




Tile Register File.




Accumulation Register File.

An input matrix of matrix multiplication instruction only uses one tile register, and large matrix must be split according to the size of tile defined by MLEN and RLEN.

For widening instructions, each output element is wider than input one. To match the width of input and output, an output matrix may be written back to a wider accumulation register whose length are specified by MLEN x AMUL.

## 3.2. Matrix Type Register, mtype

The read-only XLEN-wide *matrix type* CSR, `mtype`, provides the default type used to interpret the contents of the matrix register file, and can only be updated by `msettype{i|hi}` and field-set instructions. The matrix type determines the organization of elements in each matrix register.

> Allowing updates only via type-set or field-set instructions simplifies the maintenance of `mtype` register state.

The `mtype` register has an `mill` field, an `msew` field, an `mba` field and several type fields. Bits `mtype[XLEN-2:16]` should be written with zero, and non-zero values of this field are reserved.

*Table 2. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

The `msew` field is used to specify the element width of source operands. It is used to calculate the maximum values of matrix size.

For each type field, a value 0 means the corresponding type is disabled. Write non-zero value to enable matrix multiplication operation of the specified type. 0 will be returned and `mill` will be set if the type is not supported.

For `mint4` field, write 1 to enable 4-bit integer where a 8-bit integer will be treated as a pair of 4-bit integers. 1'b0 will be returned if 4-bit integer is not supported.

For `mint8` field, write 1 to enable 8-bit integer.

For `mint16` field, write 1 to enable 16-bit integer.

For `mint64` field, write 1 to enable 64-bit integer.

For `mfp8` field, write 2'b01 to enable E4M3, 2'b10 to enable E5M2, and 2'b11 to enable E3M4. `mfp8[1:0]` always returns 2'b00 if FP8 is not supported.

For `mfp16` field, write 2'b01 to enable IEEE-754 half-precision float point (E5M10), and write 2'b10 to enable BFloat16 (E8M7). 2'b11 is reserved.

For `mfp32` field, write 2'b01 to enalbe IEEE-754 single-precison float point (E8M23), and write 2'b10 to enable TensorFloat32 (E8M10). 2'b11 is reserved.

For `mfp64` field, write 1 to enable 64-bit double-precision float point. To support FP64 format, the implementation should support "D" extension at the same time. 0 will be returned if FP64 is not supported.

The `mba` field indicates that the out-of-bound elements is undisturbed or agnostic. When `mba` is marked undisturbed (`mba=0`), the out-of-bound elements in a matrix register retain the value it previously held. Otherwise, the out-of-bound elements can be overwritten with any values.

## 3.3. Matrix Tile Size Registers, mtilem/mtilek/mtilen

The XLEN-bit-wide read-only `mtilem/mtilek/mtilen` CSRs can only be updated by the `msettile{m|k|n}{i}` instructions. The registers holds 3 unsigned integers specifying the tile shapes for tiled matrix.

## 3.4. Matrix Start Index Register, mstart

The `mstart` read-write CSR specifies the index of the first element to be executed by load/store and element-wise arithmetic instructions. The CSR can be written by hardware on a trap, and its value represents the element on which the trap was taken. The value is the sequential number in row order.

Any legal matrix instruction can reset the `mstart` to zero at the end of excution.

## 3.5. Matrix Control and Status Register, mcsr

The `mcsr` register has 2 fields, and other bits with non-zero value are reserved.

*Table 3. `mcsr` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1:3 | 0 | Reserved if non-zero. |
| 2:1 | mmode[1:0] | The mode of matrix multiplication. |
| 0 | msat | Integer arithmetic instruction accrued saturation flag. |

`mmode` field indicates the mode of matrix multiplication. `mmode = 00` means `C = A x B`, where the source matrices, `A` and `B`, are both organized as the original order. `mmode = 01` means `C = A x BT`, where `B` is transposed. `mmode = 10` means `C = AT x B`, where `A` is transposed.

An implementation can support any combination of these modes, with extensions Zmab, Zmabt and

Zmatb.

If an unsupported `mmode` is set, then any attempt to execute a matrix multiplication instruction will raise an illegal instruction exception.

## 3.6. Matrix Context Status in mstatus and sstatus

A 2-bit matrix context status field should be added to mstatus and shadowed in status. It is defined analogously to the vector context status field, VS.

# Chapter 4. Instructions

## 4.1. Instruction Formats

The instructions in the matrix extension use 64-bit encoding with prefix 0111111 at the lowest 7 bits and a major opcode xxyyy11 at [38:32], where $\text{yyy} \neq 111$.

Instruction formats are listed below.

Configuration instructions, where `funct3` field, inst[14:12], is fixed to 000. The `imm` field supports 32-bit immediate operand.

| 63–43 | 42–39 | 38–32 |
|---|---|---|
| imm[31:11] | mtf | xxyyy11 |

| 31–26 | 25–15 | 14–12 | 11–7 | 6–0 |
|---|---|---|---|---|
| funct6 | imm[10:0] | funct3 | rd | 0111111 |

Configuration Immediate Instructions.

| 63–43 | 42–39 | 38–32 |
|---|---|---|
| 0...00 | funct4 | xxyyy11 |

| 31–26 | 25–20 | 19–15 | 14–12 | 11–7 | 6–0 |
|---|---|---|---|---|---|
| funct6 | 000000 | rs1 | funct3 | rd | 0111111 |

Configuration Instructions.

Load & store instructions, where `funct3` field is fixed to 001, and `ls` field indicates the direction of memory access (load or store). For higher 32 bits, `eew` field indicates the effective element width, `ba` field indicates if the out-of-bound elements are agnostic, and `mt` field indicates the type of matrix (00 for output/accumulation matrix, 01 for left matrix and 10 for right matrix).

| 63–50 | 49–48 | 47 | 46–44 | 43–39 | 38–32 |
|---|---|---|---|---|---|
| 0...00 | mt | ba | eew | funct5 | xxyyy11 |

| 31–26 | 25 | 24–20 | 19–15 | 14–12 | 11–7 | 6–0 |
|---|---|---|---|---|---|---|
| funct6 | ls | rs2 | rs1 | funct3 | ms3/md | 0111111 |

Load/Store Instructions.

Data move instructions, where `funct3` field is fixed to 010, `di` field indicates the moving direction, and `rc` field indicates the moving dimension (in row, in column or in element). The `mks` field specifies the source mask register and `mkm` field indicates the mask mode (to mask rows, columns or elements).

| 63–59 | 58–57 | 56–52 | 51–50 | 49–48 | 47 | 46–44 | 43–39 | 38–32 |
|---|---|---|---|---|---|---|---|---|
| mks | mkm | 00000 | rc | mt | ba | eew | funct5 | xxyyy11 |

| 31–26 | 25 | 24–20 | 19–15 | 14–12 | 11–7 | 6–0 |
|---|---|---|---|---|---|---|
| funct6 | di | rs2/imm | rs1/ms1 | funct3 | rd/md | 0111111 |

Data Move Instructions.

Matrix multiplication instructions, where `funct3` field is fixed to 100, and `fp` field indicates if the operation is float-point type. `typ*` field indicates the data type of source or destination elements (000-011 for 8b-64b, 111 for 4b, and 100 to use `msew` field of `mtype` CSR). `frm` field indicates the rounding mode of float-point result, where the encoding is the same as RVF.`sps` field specifies the source sparsity index register and `spm` field indicates the sparsing mode (01 for left matrix sparsing, 10 for right matrix sparsing, and 00 without sparsing).

| 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|
| sps | spm | typ2 | typ1 | typd | ba | frm | funct5 | xxyyy11 |

| 31 26 | 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|
| funct6 | fp | ms2 | ms1 | funct3 | md | 0111111 |

Matrix Multiplication Instructions.

Element-wise instructions, where `funct3` field is fixed to 101, and `fp` field indicates if the operation is float-point type. The `typ*` and `frm` fields are the same as matrix multiplication instructions. The `mks` and `mkm` fields are the same as data move instructions.

| 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|
| mks | mkm | typ2 | typ1 | typd | ba | frm | funct5 | xxyyy11 |

| 31 26 | 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|
| funct6 | fp | ms2 | ms1 | funct3 | md | 0111111 |

Element-wise Immediate Instructions.

Type-convert instructions, where `funct3` field is fixed to 111, `fd` field indicates if the destination is float point, and `fs` field indicates if the source is float point. Other fields are the same as element-wise instructions.

| 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|
| mks | mkm | enw | typ1 | typd | ba | frm | funct5 | xxyyy11 |

| 31 26 | 25 | 24 | 23 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|---|
| funct6 | fd | fs | funct4 | ms1 | funct3 | md | 0111111 |

Type-convert Instructions.

## 4.2. Configuration-Setting Instructions

Due to hardware resource constraints, one of the common ways to handle large-sized matrix multiplication is "tiling", where each iteration of the loop processes a subset of elements, and then continues to iterate until all elements are processed. The Matrix extension provides direct, portable support for this approach.

The block processing of matrix multiplication requires three levels of loops to iterate in the direction of the number of rows of the left matrix (m), the number of columns of the left matrix (k, also the number of rows of the right matrix), and the number of columns of the right matrix (n), given by the application.

The shapes of the matrix tiles to be processed, m (application tile length m or `ATM`), k (`ATK`), n (`ATN`), is used as candidates for `mtilem` / `mtilek` / `mtilen`. Based on microarchitecture implementation and `mmode` setting, hardware returns a new `mtilem` / `mtilek` / `mtilen` value via a general purpose register (usually smaller), also stored in `mtilem` / `mtilek` / `mtilen` CSR, which is the shape of tile per iteration handled by hardware.

For a simple matrix multiplication example, check out the Section Intrinsic Example, which describes how the code keeps track of the matrices processed by the hardware each iteration.

A set of instructions is provided to allow rapid configuration of the values in `mtile*` and `mtype` to match application needs.

The `msettype[i]` instructions set the `mtype` CSR based on their arguments, and write the new value of mtype into rd.

```
msettypei  rd, imm      # rd = new mtype, imm = new mtype[31:0] setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.
```

The `mset*` instructions set the specified field of `mtype` without affecting other fields.

```
# Set msew field.
msetsew rd, imm         # rd = new mtype, msew = imm[2:0].
msetba  rd, imm         # rd = new mtype, mba = imm[0].

# Set integer type fields.
msetint rd, int4        # rd = new mtype, set mint4 = 1 to enable INT4 type.
msetint rd, int8        # rd = new mtype, set mint8 = 1 to enable INT8 type.
msetint rd, int16       # rd = new mtype, set mint16 = 1 to enable INT16 type.
msetint rd, int32       # rd = new mtype, set mint32 = 1 to enable INT32 type.
msetint rd, int64       # rd = new mtype, set mint64 = 1 to enable INT64 type.

# Set float point type fields.
msetfp  rd, e4m3        # rd = new mtype, set mfp8 = 01 to enable FP8 E4M3 type.
msetfp  rd, e5m2        # rd = new mtype, set mfp8 = 10 to enable FP8 E5M2 type.
msetfp  rd, e3m4        # rd = new mtype, set mfp8 = 11 to enable FP8 E3M4 type.
msetfp  rd, fp16        # rd = new mtype, set mfp16 = 01 to enable FP16 E5M10 type.
msetfp  rd, bf16        # rd = new mtype, set mfp16 = 10 to enable BF16 E8M7 type.
msetfp  rd, fp32        # rd = new mtype, set mfp32 = 01 to enable FP32 E8M23 type.
msetfp  rd, tf32        # rd = new mtype, set mfp32 = 10 to enable TF32 E8M10 type.
msetfp  rd, fp64        # rd = new mtype, set mfp64 = 1 to enable FP64 type.
```

The `munset*` instructions unset the specified field of `mtype` without affecting other fields.

```
munsetint rd, int4      # rd = new mtype, set mint4 = 0 to disable INT4 type.
munsetint rd, int8      # rd = new mtype, set mint8 = 0 to disable INT8 type.
munsetint rd, int16     # rd = new mtype, set mint16 = 0 to disable INT16 type.
```

```
munsetint rd, int32     # rd = new mtype, set mint32 = 0 to disable INT32 type.
munsetint rd, int64     # rd = new mtype, set mint64 = 0 to disable INT64 type.

munsetfp  rd, fp8       # rd = new mtype, set mfp8 = 00 to disable FP8 type.
munsetfp  rd, fp16      # rd = new mtype, set mfp16 = 00 to disable FP16 type.
munsetfp  rd, fp32      # rd = new mtype, set mfp32 = 00 to disable FP32 type.
munsetfp  rd, fp64      # rd = new mtype, set mfp64 = 0 to disable FP64 type.
```

The field to be set or unset is specified by `mtf` (inst[42:39]) and the value is specified by imm[25:15].

*Table 4. Field to be set or unset*

| mtf | field |
|---|---|
| 0000 | msew |
| 0001 | mint4 |
| 0010 | mint8 |
| 0011 | mint16 |
| 0100 | mint32 |
| 0101 | mint64 |
| 0110 | mfp8 |
| 0111 | mfp16 |
| 1000 | mfp32 |
| 1001 | mfp64 |
| 1010 | mba |

The `msettile{m|k|n}[i]` instructions set the mtilem/mtilek/mtilen CSRs based on their arguments, and write the new value into rd.

```
msettilemi rd, imm          # rd = new mtilem, imm = ATM
msettilem  rd, rs1          # rd = new mtilem, rs1 = ATM
msettileki rd, imm          # rd = new mtilek, imm = ATN
msettilek  rd, rs1          # rd = new mtilek, rs1 = ATN
msettileni rd, imm          # rd = new mtilen, imm = ATK
msettilen  rd, rs1          # rd = new mtilen, rs1 = ATK
```

### 4.2.1. mtype Encoding

*Table 5. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

The new `mtype` value is encoded in the immediate fields of `msettypei`, and in the `rs1` register for `msettype`. Each field can be set or unset with `msetsew`, `msetba`, `msetfp`, `msetint`, `munsetfp` and `munsetint` instructions independently.

The fields encoded by instruction directly have higher priority than the same fileds in `mtype` CSR.

### 4.2.2. ATM/ATK/ATN Encoding

There are three values, `TMMAX`, `TKMAX` and `TNMAX`, represent the maximum shapes of the matrix tiles that could be stored in matrix registers, and can be operated on with a single matrix instruction given the current SEW settings.

The values of `TMMAX`, `TKMAX` and `TNMAX` are related to MLEN, RLEN and the configuration of `mmode`.

For `A x B` mode (`mmode=00`),

- TMMAX = MLEN / RLEN
- TKMAX = min(MLEN / RLEN, RLEN / SEW)
- TNMAX = RLEN / SEW

For `A x BT` mode (`mmode=01`),

- TMMAX = MLEN / RLEN
- TKMAX = RLEN / SEW
- TNMAX = MLEN / RLEN

For `AT x B` mode (`mmode=10`),

- TMMAX = min(MLEN / RLEN, RLEN / SEW)
- TKMAX = MLEN / RLEN
- TNMAX = RLEN / SEW

For examples, with `MLEN=256`, `RLEN=64` and `mmode=00`, `TMMAX`, `TKMAX` and `TNMAX` values are shown below.

```
SEW=8,  TMMAX=4, TKMAX=4, TNMAX=8       # 4x4x8 8-bit matmul
SEW=16, TMMAX=4, TKMAX=4, TNMAX=4       # 4x4x4 16-bit matmul
SEW=32, TMMAX=4, TKMAX=2, TNMAX=2       # 4x2x2 32-bit matmul
```

The new tile shape settings are based on `ATM` / `ATK` / `ATN` values, which for `msettile{m|k|n}` is encoded in the rs1 and rd fields.

| rd | rs1 | `ATM/ATK/ATN` value | Effect on `mtilem/mtilek/mtilen` |
|---|---|---|---|
| - | !x0 | Value in `x[rs1]` | Normal tiling |
| !x0 | x0 | ~0 | Set `mtilem/mtilek/mtilen` to `TMMAX/TKMAX/TNMAX` |
| x0 | x0 | Value in `mtilem/mtilek/mtilen` | Keep existing `mtilem/mtilek/mtilen` if less than `TMMAX/TKMAX/TNMAX` |

For the `msettile{m|k|n}i` instructions, the `ATM` / `ATK` / `ATN` is encoded as a 10-bit unsigned immediate in the rs1.

### 4.2.3. Constraints on Setting mtilem/mtilek/mtilen

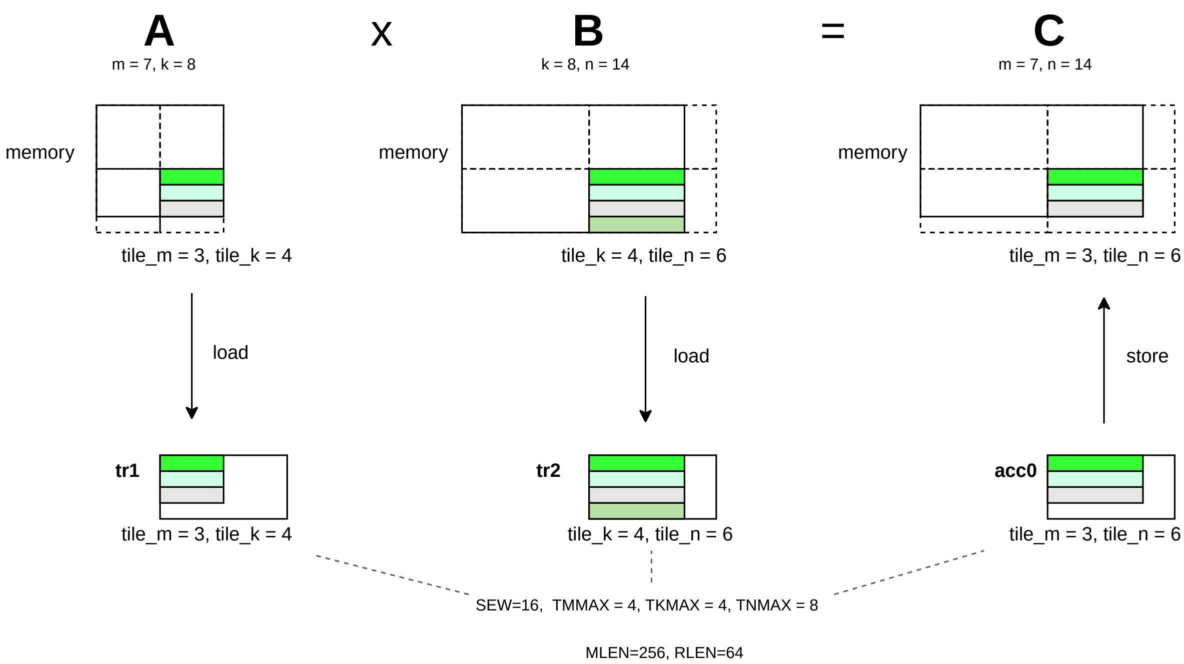
The `msettile{m|k|n}[i]` instructions first set `TMMAX/TKMAX/TNMAX` according to the mtype CSR, then set `mtilem/mtilek/mtilen` obeying the following constraints (using `mtilem` & `ATM` & `TMMAX` as an example, and the same with `mtilek` & `ATK` & `TKMAX` and `mtilen` & `ATN` & `TNMAX`):

1. `mtilem = ATM` if `ATM <= TMMAX`
2. `ceil(ATM / 2) <= mtilem <= TMMAX` if `ATM < (2 * TMMAX)`
3. `mtilem = TMMAX` if `ATM >= (2 * TMMAX)`
4. Deterministic on any given implementation for same input ATM and TMMAX values

5. These specific properties follow from the prior rules:
   a. `mtilem = 0` if `ATM = 0`
   b. `mtilem > 0` if `ATM > 0`
   c. `mtilem <= TMMAX`
   d. `mtilem <= ATM`
   e. a value read from `mtilem` when used as the ATM argument to `msettile{m|k|n}{i}` results in the same value in `mtilem`, provided the resultant TMMAX equals the value of TMMAX at the time that `mtilem` was read.

Continue to use `MLEN=256`, `RLEN=64` and `mmode=00` as a example. When SEW=16, TMMAX=4, TKMAX=4, TNMAX=8.

If A is a 7 x 8 matrix and B is a 8 x 14 matrix, we could get `mtilem/mtilek/mtilen` values as show below, in the last loop of tiling.




## 4.3. Load and Store Instructions

### 4.3.1. Load Instructions

Load a matrix tile from memory.

```
# md destination, rs1 base address, rs2 row byte stride

# For left matrix, A
```

```
# tile size = mtilem * mtilek
mlae8.m  md, (rs1), rs2         #  8-bit left tile load
mlae16.m md, (rs1), rs2         # 16-bit left tile load
mlae32.m md, (rs1), rs2         # 32-bit left tile load
mlae64.m md, (rs1), rs2         # 64-bit left tile load

# For right matrix, B
# tile size = mtilek * mtilen
mlbe8.m  md, (rs1), rs2         #  8-bit right tile load
mlbe16.m md, (rs1), rs2         # 16-bit right tile load
mlbe32.m md, (rs1), rs2         # 32-bit right tile load
mlbe64.m md, (rs1), rs2         # 64-bit right tile load

# For output matrix, C
# tile size = mtilem * mtilen
mlce8.m  md, (rs1), rs2         #  8-bit output tile load
mlce16.m md, (rs1), rs2         # 16-bit output tile load
mlce32.m md, (rs1), rs2         # 32-bit output tile load
mlce64.m md, (rs1), rs2         # 64-bit output tile load
```

Load a matrix tile from memory, where the matrix on memory is transposed.

```
# md destination, rs1 base address, rs2 row byte stride

# For left matrix, A
# tile size = mtilek * mtilem
mlate8.m  md, (rs1), rs2        #  8-bit left tile load
mlate16.m md, (rs1), rs2        # 16-bit left tile load
mlate32.m md, (rs1), rs2        # 32-bit left tile load
mlate64.m md, (rs1), rs2        # 64-bit left tile load

# For right matrix, B
# tile size = mtilen * mtilek
mlbte8.m  md, (rs1), rs2        #  8-bit right tile load
mlbte16.m md, (rs1), rs2        # 16-bit right tile load
mlbte32.m md, (rs1), rs2        # 32-bit right tile load
mlbte64.m md, (rs1), rs2        # 64-bit right tile load

# For output matrix, C
# tile size = mtilen * mtilem
mlcte8.m  md, (rs1), rs2        #  8-bit output tile load
mlcte16.m md, (rs1), rs2        # 16-bit output tile load
mlcte32.m md, (rs1), rs2        # 32-bit output tile load
mlcte64.m md, (rs1), rs2        # 64-bit output tile load
```

### 4.3.2. Store Instructions

Store a matrix tile to memory.

```
# ms3 store data, rs1 base address, rs2 row byte stride

# For left matrix, A
# tile size = mtilem * mtilek
msae8.m  ms3, (rs1), rs2       #  8-bit left tile store
msae16.m ms3, (rs1), rs2       # 16-bit left tile store
msae32.m ms3, (rs1), rs2       # 32-bit left tile store
msae64.m ms3, (rs1), rs2       # 64-bit left tile store

# For right matrix, B
# tile size = mtilek * mtilen
msbe8.m  ms3, (rs1), rs2       #  8-bit right tile store
msbe16.m ms3, (rs1), rs2       # 16-bit right tile store
msbe32.m ms3, (rs1), rs2       # 32-bit right tile store
msbe64.m ms3, (rs1), rs2       # 64-bit right tile store

# For output matrix, C
# tile size = mtilem * mtilen
msce8.m  ms3, (rs1), rs2       #  8-bit output tile store
msce16.m ms3, (rs1), rs2       # 16-bit output tile store
msce32.m ms3, (rs1), rs2       # 32-bit output tile store
msce64.m ms3, (rs1), rs2       # 64-bit output tile store
```

Save a matrix tile to memory, where the matrix on memory is transposed.

```
# ms3 store data, rs1 base address, rs2 row byte stride

# For left matrix, A
# tile size = mtilek * mtilem
msate8.m  ms3, (rs1), rs2      #  8-bit left tile store
msate16.m ms3, (rs1), rs2      # 16-bit left tile store
msate32.m ms3, (rs1), rs2      # 32-bit left tile store
msate64.m ms3, (rs1), rs2      # 64-bit left tile store

# For right matrix, B
# tile size = mtilen * mtilek
msbte8.m  ms3, (rs1), rs2      #  8-bit right tile store
msbte16.m ms3, (rs1), rs2      # 16-bit right tile store
msbte32.m ms3, (rs1), rs2      # 32-bit right tile store
msbte64.m ms3, (rs1), rs2      # 64-bit right tile store

# For output matrix, C
# tile size = mtilen * mtilem
```

```
mscte8.m  ms3, (rs1), rs2       #  8-bit output tile store
mscte16.m ms3, (rs1), rs2       # 16-bit output tile store
mscte32.m ms3, (rs1), rs2       # 32-bit output tile store
mscte64.m ms3, (rs1), rs2       # 64-bit output tile store
```

### 4.3.3. Whole Matrix Load & Store Instructions

Load a whole matrix from memory without considering the tile size.

```
mlre8.m   md, (rs1), rs2        #  8-bit whole matrix load
mlre16.m  md, (rs1), rs2        # 16-bit whole matrix load
mlre32.m  md, (rs1), rs2        # 32-bit whole matrix load
mlre64.m  md, (rs1), rs2        # 64-bit whole matrix load
```

Load a whole matrix from memory without considering the tile size, where the matrix on memory is transposed.

```
mlrte8.m  md, (rs1), rs2        #  8-bit whole matrix load
mlrte16.m md, (rs1), rs2        # 16-bit whole matrix load
mlrte32.m md, (rs1), rs2        # 32-bit whole matrix load
mlrte64.m md, (rs1), rs2        # 64-bit whole matrix load
```

Store a whole matrix to memory without considering the tile size.

```
msre8.m   ms3, (rs1), rs2       #  8-bit whole matrix store
msre16.m  ms3, (rs1), rs2       # 16-bit whole matrix store
msre32.m  ms3, (rs1), rs2       # 32-bit whole matrix store
msre64.m  ms3, (rs1), rs2       # 64-bit whole matrix store
```

Store a whole matrix to memory without considering the tile size, where the matrix on memory is transposed.

```
msrte8.m  ms3, (rs1), rs2       #  8-bit whole matrix store
msrte16.m ms3, (rs1), rs2       # 16-bit whole matrix store
msrte32.m ms3, (rs1), rs2       # 32-bit whole matrix store
msrte64.m ms3, (rs1), rs2       # 64-bit whole matrix store
```

Whole matrix load and store instructions are usually used for context saving and restoring.

# 4.4. Data Move Instructions

## 4.4.1. Data Move Instructions between Matrix Registers

Data move instructions between matrix registers are used to move elements between two tile registers, two accumulation registers, or one tile register and one accumulation register.

```
# md = ms1, md and ms1 are both tile registers.
mmve8.t.t   md, ms1
mmve16.t.t  md, ms1
mmve32.t.t  md, ms1
mmve64.t.t  md, ms1

# md = ms1, md and ms1 are both accumulation registers.
mmve8.a.a   md, ms1
mmve16.a.a  md, ms1
mmve32.a.a  md, ms1
mmve64.a.a  md, ms1

# md[i, rs2 * (RLEN / EEW) + j] = ms1[i, j]
# md is an accumulation register and ms1 is a tile register.
mmve8.a.t   md, ms1, rs2
mmve16.a.t  md, ms1, rs2
mmve32.a.t  md, ms1, rs2
mmve64.a.t  md, ms1, rs2

# md[i, j] = ms1[i, rs2 * (RLEN / EEW) + j]
# md is a tile register and ms1 is an accumulation register.
mmve8.t.a   md, ms1, rs2
mmve16.t.a  md, ms1, rs2
mmve32.t.a  md, ms1, rs2
mmve64.t.a  md, ms1, rs2

# md[i, imm * (RLEN / EEW) + j] = ms1[i, j]
# md is an accumulation register and ms1 is a tile register.
mmvie8.a.t  md, ms1, imm
mmvie16.a.t md, ms1, imm
mmvie32.a.t md, ms1, imm
mmvie64.a.t md, ms1, imm

# md[i, j] = ms1[i, imm * (RLEN / EEW) + j]
# md is a tile register and ms1 is an accumulation register.
mmvie8.t.a  md, ms1, imm
mmvie16.t.a md, ms1, imm
mmvie32.t.a md, ms1, imm
mmvie64.t.a md, ms1, imm
```

### 4.4.2. Data Move Instructions between Matrix and Integer

Data move instructions between matrix and integer are used to move single element between integer registers and tile registers. Such instructions can change a part of matrix and often used for debug.

```
# x[rd] = ms1[i, j], i = rs2[15:0], j = rs2[XLEN-1:16]
# rd is an integer register and ms1 is a tile register.
mmve8.x.t  rd, ms1, rs2
mmve16.x.t rd, ms1, rs2
mmve32.x.t rd, ms1, rs2
mmve64.x.t rd, ms1, rs2

# md[i, j] = x[rs1], i = rs2[15:0], j = rs2[XLEN-1:16]
# md is a tile register and rs1 is an integer register.
mmve8.t.x  md, rs1, rs2
mmve16.t.x md, rs1, rs2
mmve32.t.x md, rs1, rs2
mmve64.t.x md, rs1, rs2

# x[rd] = ms1[i, j], i = rs2[15:0], j = rs2[XLEN-1:16]
# rd is an integer register and ms1 is an accumulation register.
mmve8.x.a  rd, ms1, rs2
mmve16.x.a rd, ms1, rs2
mmve32.x.a rd, ms1, rs2
mmve64.x.a rd, ms1, rs2

# md[i, j] = x[rs1], i = rs2[15:0], j = rs2[XLEN-1:16]
# md is an accumulation register and rs1 is an integer register.
mmve8.a.x  md, rs1, rs2
mmve16.a.x md, rs1, rs2
mmve32.a.x md, rs1, rs2
mmve64.a.x md, rs1, rs2
```

The `mmve*.x.t/a` instruction copies a signle SEW-wide element of the matrix register to an integer register, where the element coordinates are specified by rs2. If SEW > XLEN, the least-significat XLEN bits are transferred. If SEW < XLEN, the valud is sign-extended to XLEN bits.

The `mmve*.t/a.x` instruction copies an integer register to an element of the destination matrix register, where the element coordinates are specified by rs2. If SEW < XLEN, the least-significant bits are moved and the upper (XLEN-SEW) bits are ignored. If SEW > XLEN, the valud is sign-extended to SEW bits. The other elements of the tile register are treated as out-of-bound elements, using the setting of `ba`.

### 4.4.3. Data Move Instructions between Matrix and Float-point

Float point data move instructions are similar with integer.

```
# f[rd] = ms1[i, j], i = rs2[15:0], j = rs2[XLEN-1:16]
# rd is a float-point register and ms1 is a tile register.
mfmve8.f.t  rd, ms1, rs2
mfmve16.f.t rd, ms1, rs2
mfmve32.f.t rd, ms1, rs2
mfmve64.f.t rd, ms1, rs2

# md[i, j] = f[rs1], i = rs2[15:0], j = rs2[XLEN-1:16]
# md is a tile register and rs1 is a float-point register.
mfmve8.t.f  md, rs1, rs2
mfmve16.t.f md, rs1, rs2
mfmve32.t.f md, rs1, rs2
mfmve64.t.f md, rs1, rs2

# f[rd] = ms1[i, j], i = rs2[15:0], j = rs2[XLEN-1:16]
# rd is a float-point register and ms1 is an accumulation register.
mfmve8.f.a  rd, ms1, rs2
mfmve16.f.a rd, ms1, rs2
mfmve32.f.a rd, ms1, rs2
mfmve64.f.a rd, ms1, rs2

# md[i, j] = f[rs1], i = rs2[15:0], j = rs2[XLEN-1:16]
# md is an accumulation register and rs1 is a float-point register.
mfmve8.a.f  md, rs1, rs2
mfmve16.a.f md, rs1, rs2
mfmve32.a.f md, rs1, rs2
mfmve64.a.f md, rs1, rs2
```

### 4.4.4. Data Broadcast Instructions

The first row/column and the first element of a matrix register can be broadcasted to fill the whole matrix.

```
# Broadcast the first row of a tile register to fill the whole matrix.
mbcar.m md, ms1
mbcbr.m md, ms1

# Broadcast the first row of an accumulation register to fill the whole matrix.
mbccr.m md, ms1

# Broadcast the first column of a tile register to fill the whole matrix.
mbcace8.m  md, ms1
mbcace16.m md, ms1
mbcace32.m md, ms1
mbcace64.m md, ms1
```

```
mbcbce8.m  md, ms1
mbcbce16.m md, ms1
mbcbce32.m md, ms1
mbcbce64.m md, ms1

# Broadcast the first column of an accumulation register to fill the whole matrix.
mbccce8.m  md, ms1
mbccce16.m md, ms1
mbccce32.m md, ms1
mbccce64.m md, ms1

# Broadcast the first element of a tile register to fill the whole matrix.
mbcaee8.m  md, ms1
mbcaee16.m md, ms1
mbcaee32.m md, ms1
mbcaee64.m md, ms1

mbcbee8.m  md, ms1
mbcbee16.m md, ms1
mbcbee32.m md, ms1
mbcbee64.m md, ms1

# Broadcast the first element of an accumulation register to fill the whole matrix.
mbccee8.m  md, ms1
mbccee16.m md, ms1
mbccee32.m md, ms1
mbccee64.m md, ms1
```

### 4.4.5. Matrix Transpose Instructions

Transpose instruction can only be used for square matrix. For matrix A, the sizes of two dimensions are both min(`mtilem`, `mtilek`). Matrix B and C are similar.

```
# Transpose square matrix of tile register.
mtae8.m  md, ms1
mtae16.m md, ms1
mtae32.m md, ms1
mtae64.m md, ms1

mtbe8.m  md, ms1
mtbe16.m md, ms1
mtbe32.m md, ms1
mtbe64.m md, ms1

# Transpose square matrix of accumulation register.
mtce8.m  md, ms1
mtce16.m md, ms1
```

```
mtce32.m md, ms1
mtce64.m md, ms1
```

## 4.5. Arithmetic and Logic Instructions

### 4.5.1. Matrix Multiplication Instructions

Matrix Multiplication operations take two matrix tiles from matrix **tile registers** specified by `ms1` and `ms2` respectively, and the output matrix tile is a matrix **accumulation register** specified by `md`.

```
# Unigned integer matrix multiplication and add, md = md + ms1 * ms2.
mmau.[dw].mm   md, ms1, ms2       # unsigned int64, output no-widen
mmau.[w].mm    md, ms1, ms2       # unsigned int32, output no-widen
mmau.[h].mm    md, ms1, ms2       # unsigned int16, output no-widen
mqmau.[b].mm   md, ms1, ms2       # unsigned int8, output quad-widen
momau.[hb].mm  md, ms1, ms2       # unsigned int4, output oct-widen

msmau.[dw].mm  md, ms1, ms2       # unsigned int64, output no-widen and saturated
msmau.[w].mm   md, ms1, ms2       # unsigned int32, output no-widen and saturated
msmau.[h].mm   md, ms1, ms2       # unsigned int16, output no-widen and saturated
msqmau.[b].mm  md, ms1, ms2       # unsigned int8, output quad-widen and saturated
msomau.[hb].mm md, ms1, ms2       # unsigned int4, output oct-widen and saturated

# Signed integer matrix multiplication and add, md = md + ms1 * ms2.
mma.[dw].mm    md, ms1, ms2       # signed int64, output no-widen
mma.[w].mm     md, ms1, ms2       # signed int32, output no-widen
mma.[h].mm     md, ms1, ms2       # signed int16, output no-widen
mqma.[b].mm    md, ms1, ms2       # signed int8, output quad-widen
moma.[hb].mm   md, ms1, ms2       # signed int4, output oct-widen

msma.[dw].mm   md, ms1, ms2       # signed int64, output no-widen and saturated
msma.[w].mm    md, ms1, ms2       # signed int32, output no-widen and saturated
msma.[h].mm    md, ms1, ms2       # signed int16, output no-widen and saturated
msqma.[b].mm   md, ms1, ms2       # signed int8, output quad-widen and saturated
msoma.[hb].mm  md, ms1, ms2       # signed int4, output oct-widen and saturated

# Float point matrix multiplication and add, md = md + ms1 * ms2.
mfma.[d].mm    md, ms1, ms2       # 64-bit float point
mfma.[f].mm    md, ms1, ms2       # 32-bit float point
mfma.[hf].mm   md, ms1, ms2       # 16-bit float point

mfwma.[f].mm   md, ms1, ms2       # 32-bit float point, output double-widen
mfwma.[hf].mm  md, ms1, ms2       # 16-bit float point, output double-widen
mfwma.[cf].mm  md, ms1, ms2       # 8-bit float point, output double-widen
mfqma.[cf].mm  md, ms1, ms2       # 8-bit float point, output quad-widen
```

A subset of these instructions is supported according to the implemented standard extensions (Zmi4, Zmi8, etc.).

The field `frm` from `fcsr` indicates the rounding mode of float-point matrix instructions. The encoding is shown below.

| frm | Mnemonic | Meaning |
|---|---|---|
| 000 | RNE | Round to Nearest, ties to Even |
| 001 | RTZ | Round towards Zero |
| 010 | RDN | Round Down (towards $-\infty$) |
| 011 | RUP | Round Up (towards $+\infty$) |
| 100 | RMM | Round to Nearest, ties to Max Magnitude |
| 101 | | Invalid |
| 110 | | Invalid |
| 111 | | Invalid |

### 4.5.2. Element-Wise Instructions

Matrix element-wise add/sub/multiply instructions. The input and output matrices are both accumulation registers and always with size `mtilem x mtilen`. The element-wise calculation of tile registers can be implemented by combining data move instructions (such as `mmve*.a.t` and `mmve*.t.a`).

```
# Unsigned integer matrix element-wise add.
# md[i,j] = ms1[i,j] + ms2[i,j]
maddu.[hb|b|h|w|dw].mm   md, ms1, ms2
msaddu.[hb|b|h|w|dw].mm  md, ms1, ms2  # output saturated
mwaddu.[hb|b|h|w].mm     md, ms1, ms2  # output double widen

# Signed integer matrix element-wise add.
# md[i,j] = ms1[i,j] + ms2[i,j]
madd.[hb|b|h|w|dw].mm    md, ms1, ms2
msadd.[hb|b|h|w|dw].mm   md, ms1, ms2  # output saturated
mwadd.[hb|b|h|w].mm      md, ms1, ms2  # output double widen

# Unsigned integer matrix element-wise subtract.
# md[i,j] = ms1[i,j] - ms2[i,j]
msubu.[hb|b|h|w|dw].mm   md, ms1, ms2
mssubu.[hb|b|h|w|dw].mm  md, ms1, ms2  # output saturated
mwsubu.[hb|b|h|w].mm     md, ms1, ms2  # output double widen

# Signed integer matrix element-wise subtract.
```

```
# md[i,j] = ms1[i,j] - ms2[i,j]
msub.[hb|b|h|w|dw].mm    md, ms1, ms2
mssub.[hb|b|h|w|dw].mm   md, ms1, ms2  # output saturated
mwsub.[hb|b|h|w].mm      md, ms1, ms2  # output double widen

# Integer matrix element-wise minimum.
# md[i,j] = min{ms1[i,j], ms2[i,j]}
mminu.[hb|b|h|w|dw].mm   md, ms1, ms2
mmin.[hb|b|h|w|dw].mm    md, ms1, ms2

# Integer matrix element-wise maximum.
# md[i,j] = max{ms1[i,j], ms2[i,j]}
mmaxu.[hb|b|h|w|dw].mm   md, ms1, ms2
mmax.[hb|b|h|w|dw].mm    md, ms1, ms2

# Integer matrix bit-wise logic.
mand.[hb|b|h|w|dw].mm    md, ms1, ms2
mor.[hb|b|h|w|dw].mm     md, ms1, ms2
mxor.[hb|b|h|w|dw].mm    md, ms1, ms2

# Integer matrix element-wise shift.
msll.[hb|b|h|w|dw].mm    md, ms1, ms2
msrl.[hb|b|h|w|dw].mm    md, ms1, ms2
msra.[hb|b|h|w|dw].mm    md, ms1, ms2

# Integer matrix element-wise multiply.
# md[i,j] = ms1[i,j] * ms2[i,j]
mmul.[hb|b|h|w|dw].mm    md, ms1, ms2  # signed, returning low bits of product
mmulh.[hb|b|h|w|dw].mm   md, ms1, ms2  # signed, returning high bits of product
mmulhu.[hb|b|h|w|dw].mm  md, ms1, ms2  # unsigned, returning high bits of product
mmulhsu.[hb|b|h|w|dw].mm md, ms1, ms2  # signed-unsigned, returning high bits of
product

# Saturated integer matrix element-wise multiply.
msmul.[hb|b|h|w|dw].mm   md, ms1, ms2  # signed
msmulu.[hb|b|h|w|dw].mm  md, ms1, ms2  # unsigned
msmulsu.[hb|b|h|w|dw].mm md, ms1, ms2  # signed-unsigned

# Widening integer matrix element-wise multiply.
mwmul.[hb|b|h|w].mm      md, ms1, ms2  # signed
mwmulu.[hb|b|h|w].mm     md, ms1, ms2  # unsigned
mwmulsu.[hb|b|h|w].mm    md, ms1, ms2  # signed-unsigned

# Float matrix element-wise add.
# md[i,j] = ms1[i,j] + ms2[i,j]
mfadd.[cf|hf|f|d].mm     md, ms1, ms2
mfwadd.[cf|hf|f].mm      md, ms1, ms2  # output double widen

# Float matrix element-wise subtract.
```

```
# md[i,j] = ms1[i,j] - ms2[i,j]
mfsub.[cf|hf|f|d].mm     md, ms1, ms2
mfwsub.[cf|hf|f].mm      md, ms1, ms2  # output double widen

# Float matrix element-wise minimum.
# md[i,j] = min{ms1[i,j], ms2[i,j]}
mfmin.[cf|hf|f|d].mm     md, ms1, ms2

# Float matrix element-wise maximum.
# md[i,j] = max{ms1[i,j], ms2[i,j]}
mfmax.[cf|hf|f|d].mm     md, ms1, ms2

# Float matrix element-wise multiply.
# md[i,j] = ms1[i,j] * ms2[i,j]
mfmul.[cf|hf|f|d].mm     md, ms1, ms2
mfwmul.[cf|hf|f].mm      md, ms1, ms2  # output double widen

# Float matrix element-wise divide.
# md[i,j] = ms1[i,j] / ms2[i,j]
mfdiv.[cf|hf|f|d].mm     md, ms1, ms2

# Float matrix element-wise square root.
# md[i,j] = ms1[i,j] ^ (1/2)
mfsqrt.[cf|hf|f|d].m     md, ms1
```

There is no matrix-scalar and matrix-vector version for element-wise instructions. Such operations can be replaced by a broadcast instruction and a matrix-matrix element-wise instruction.

## 4.6. Type-Convert Instructions

The input and output matrices of type-convert instructions are both accumulation registers and always with size `mtilem x mtilen`. The type convert of tile registers can be implemented by combining data move instructions (such as `mmve*.a.t` and `mmve*.t.a`).

```
# Convert float to float
mfcvt.bf.hf.m   md, ms1         # fp16 to bf16
mfcvt.hf.bf.m   md, ms1         # bf16 to fp16

mfwcvt.fw.f.m   md, ms1         # single-width float to double-width float
mfwcvt.hf.cf.m  md, ms1         # fp8 to fp16
mfwcvt.f.hf.m   md, ms1         # fp16 to fp32
mfwcvt.d.f.m    md, ms1         # fp32 to fp64

mfncvt.f.fw.m   md, ms1         # double-width float to single-width float
mfncvt.cf.hf.m  md, ms1         # fp16 to fp8
```

```
mfncvt.hf.f.m   md, ms1         # fp32 to fp16
mfncvt.f.d.m    md, ms1         # fp64 to fp32

# Convert integer to float
mfcvtu.f.x.m    md, ms1         # uint to float
mfcvtu.hf.h.m   md, ms1         # uint16 to fp16
mfcvtu.f.w.m    md, ms1         # uint32 to fp32
mfcvtu.d.dw.m   md, ms1         # uint64 to fp64

mfcvt.f.x.m     md, ms1         # int to float
mfcvt.hf.h.m    md, ms1         # int16 to fp16
mfcvt.f.w.m     md, ms1         # int32 to fp32
mfcvt.d.dw.m    md, ms1         # int64 to fp64

mfwcvtu.fw.x.m  md, ms1         # single-width uint to double-width float
mfwcvtu.fq.x.m  md, ms1         # single-width uint to quad-width float
mfwcvtu.fo.x.m  md, ms1         # single-width uint to oct-width float
mfwcvtu.hf.hb.m md, ms1         # uint4 to fp16
mfwcvtu.f.hb.m  md, ms1         # uint4 to fp32
mfwcvtu.hf.b.m  md, ms1         # uint8 to fp16
mfwcvtu.f.b.m   md, ms1         # uint8 to fp32
mfwcvtu.f.h.m   md, ms1         # uint16 to fp32
mfwcvtu.d.w.m   md, ms1         # uint32 to fp64

mfwcvt.fw.x.m   md, ms1         # single-width int to double-width float
mfwcvt.fq.x.m   md, ms1         # single-width int to quad-width float
mfwcvt.fo.x.m   md, ms1         # single-width int to oct-width float
mfwcvt.hf.hb.m  md, ms1         # int4 to fp16
mfwcvt.f.hb.m   md, ms1         # int4 to fp32
mfwcvt.hf.b.m   md, ms1         # int8 to fp16
mfwcvt.f.b.m    md, ms1         # int8 to fp32
mfwcvt.f.h.m    md, ms1         # int16 to fp32
mfwcvt.d.w.m    md, ms1         # int32 to fp64

mfncvtu.f.xw.m  md, ms1         # double-width uint to float
mfncvtu.hf.w.m  md, ms1         # uint32 to fp16
mfncvtu.f.dw.m  md, ms1         # uint64 to fp32

mfncvt.f.xw.m   md, ms1         # double-width int to float
mfncvt.hf.w.m   md, ms1         # int32 to fp16
mfncvt.f.dw.m   md, ms1         # int64 to fp32

# Convert float to integer
mfcvtu.x.f.m    md, ms1         # float to uint
mfcvtu.h.hf.m   md, ms1         # fp16 to uint16
mfcvtu.w.f.m    md, ms1         # fp32 to uint32
mfcvtu.dw.d.m   md, ms1         # fp64 to uint64

mfcvt.x.f.m     md, ms1         # float to int
```

```
mfcvt.h.hf.m    md, ms1         # fp16 to int16
mfcvt.w.f.m     md, ms1         # fp32 to int32
mfcvt.dw.d.m    md, ms1         # fp64 to int64

mfwcvtu.xw.f.m  md, ms1         # single-width float to double-width uint
mfwcvtu.w.hf.m  md, ms1         # fp16 to uint32
mfwcvtu.dw.f.m  md, ms1         # fp32 to uint64

mfwcvt.xw.f.m   md, ms1         # single-width float to double-width int
mfwcvt.w.hf.m   md, ms1         # fp16 to int32
mfwcvt.dw.f.m   md, ms1         # fp32 to int64

mfncvtu.x.fw.m  md, ms1         # double-width float to single-width uint
mfncvtu.x.fq.m  md, ms1         # quad-width float to single-width uint
mfncvtu.x.fo.m  md, ms1         # oct-width float to single-width uint
mfncvtu.hb.hf.m md, ms1         # fp16 to uint4
mfncvtu.hb.f.m  md, ms1         # fp32 to uint4
mfncvtu.b.hf.m  md, ms1         # fp16 to uint8
mfncvtu.b.f.m   md, ms1         # fp32 to uint8
mfncvtu.h.f.m   md, ms1         # fp32 to uint16
mfncvtu.w.d.m   md, ms1         # fp64 to uint32

mfncvt.x.fw.m   md, ms1         # double-width float to single-width int
mfncvt.x.fq.m   md, ms1         # quad-width float to single-width int
mfncvt.x.fo.m   md, ms1         # oct-width float to single-width int
mfncvt.hb.hf.m  md, ms1         # fp16 to int4
mfncvt.hb.f.m   md, ms1         # fp32 to int4
mfncvt.b.hf.m   md, ms1         # fp16 to int8
mfncvt.b.f.m    md, ms1         # fp32 to int8
mfncvt.h.f.m    md, ms1         # fp32 to int16
mfncvt.w.d.m    md, ms1         # fp64 to int32
```

# Chapter 5. Intrinsic Examples

## 5.1. Matrix multiplication

```
void matmul_float16(c, a, b, m, k, n) {
    msettype(e16);                              // use 16bit input matrix element
    for (i = 0; i < m; i += mtilem) {           // loop at dim m with tiling
        mtilem = msettilem(m-i);
        for (j = 0; j < n; j += mtilen) {       // loop at dim n with tiling
            mtilen = msettilen(n-j);

            out = mwsub_mm(out, out)            // clear output reg
            for (s = 0; s < k; s += mtilek) {   // loop at dim k with tiling
                mtilek = msettilek(k-s);

                tr1 = mlae16_m(&a[i][s], k*2);  // load left matrix a
                tr2 = mlbe16_m(&b[s][j], n*2);  // load right matrix b
                out = mfwma_mm(tr1, tr2);       // tiled matrix multiply,
                                                // double widen output
            }

            out = mfncvt_f_fw_m(out);           // convert widen result
            msce16_m(out, &c[i][j], n*2);       // store to matrix c
        }
    }
}
```

## 5.2. Matrix multiplication with left matrix transposed

```
void matmul_a_tr_float16(c, a, b, m, k, n) {
    msettype(e16);                              // use 16bit input matrix element
    for (i = 0; i < m; i += mtilem) {           // loop at dim m with tiling
        mtilem = msettilem(m-i);
        for (j = 0; j < n; j += mtilen) {       // loop at dim n with tiling
            mtilen = msettilen(n-j);

            out = mwsub_mm(out, out)            // clear output reg
            for (s = 0; s < k; s += mtilek) {   // loop at dim k with tiling
                mtilek = msettilek(k-s);

                tr1 = mlate16_m(&a[s][i], m*2); // load transposed left matrix a
                tr2 = mlbe16_m(&a[s][j], n*2);  // load right matrix b
                out = mfwma_mm(tr1, tr2);       // tiled matrix multiply,
                                                // double widen output
            }
```

```
            out = mfncvt_f_fw_m(out);           // convert widen result
            msce16_m(out, &c[i][j], n*2);       // store to matrix c
        }
    }
}
```

## 5.3. Matrix transpose without multiplication

```
void mattrans_float16(out, in, h, w) {
    msettype(e16);                              // use 16bit input matrix element

    for (i = 0; i < h; i += mtilem) {           // loop at dim m with tiling
        mtilem = msettilem(h-i);
        for (j = 0; j < w; j += mtilek) {       // loop at dim k with tiling
            mtilek = msettilek(w-j);

            tr_in = mlae16_m(&in[i][j], w*2);   // load input matrix
            msate16_m(tr_in, &out[j][i], h*2);  // store output matrix
        }
    }
}
```

# Chapter 6. Standard Matrix Extensions

## 6.1. Zma*b*: Matrix Mode Extension

The Zmab extension allows to use `C = A x B` mode for matrix multiplication, where the setting of `mcsr.mmode = 00` is legal.

The Zmabt extension allows to use `C = A x BT` mode for matrix multiplication, where the setting of `mcsr.mmode = 01` is legal.

The Zmatb extension allows to use `C = AT x B` mode for matrix multiplication, where the setting of `mcsr.mmode = 10` is legal.

## 6.2. Zmi4: Matrix 4-bit Integer Extension

The Zmi4 extension allows to use 4-bit integer as the data type of input matrix elements.

The Zmi4 extension adds a bit `mtype[3]` in `mtype` register.

*Table 6. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mint4` field, write 1 to enable 4-bit integer where a 8-bit integer will be treated as a pair of 4-bit integers (the size of a row must be even). 0 will be returned and `mtype.mill` will be set if 4-bit integer is not supported.

The `mint4` field can be set with other fields by `msettype[i]` or set independently by `msetint` or `munsetint`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetint   rd, int4      # rd = new mtype, set mint4 = 1 to enable INT4 type.
munsetint rd, int4      # rd = new mtype, set mint4 = 0 to disable INT4 type.
```

As int4 must be in pair, the e8 load/store and data move instructions are reused for int4 data.

The element-wise and type-convert instructions with suffix .hb are added for int4 format.

Four octuple-widen instructions are added to support int4 matrix multiplication. So the output type is always 32-bit integer.

```
momau.[hb].mm  md, ms1, ms2     # unsigned int4, output oct-widen
msomau.[hb].mm md, ms1, ms2     # unsigned int4, output oct-widen and saturated

moma.[hb].mm   md, ms1, ms2     # signed int4, output oct-widen
msoma.[hb].mm  md, ms1, ms2     # signed int4, output oct-widen and saturated
```

## 6.3. Zmi8: Matrix 8-bit Integer Extension

The Zmi8 extension allows to use 8-bit integer as the data type of input matrix elements.

The Zmi8 extension adds a bit `mtype[4]` in `mtype` register.

*Table 7. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |

| Bits | Name | Description |
|---|---|---|
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mint8` field, write 1 to enable 8-bit integer. 0 will be returned and `mtype.mill` will be set if 8-bit integer is not supported.

The `mint8` field can be set with other fields by `msettype[i]` or set independently by `msetint` or `munsetint`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetint   rd, int8      # rd = new mtype, set mint8 = 1 to enable INT8 type.
munsetint rd, int8      # rd = new mtype, set mint8 = 0 to disable INT8 type.
```

The e8 load/store and data move instructions are used for int8 data.

The element-wise and type-convert instructions with .b suffix are added for int8 format.

Four quadruple-widen instructions are added to support int8 matrix multiplication. So the output type is always 32-bit integer.

```
mqmau.[b].mm  md, ms1, ms2      # unsigned int8, output quad-widen
msqmau.[b].mm md, ms1, ms2      # unsigned int8, output quad-widen and saturated

mqma.[b].mm   md, ms1, ms2      # signed int8, output quad-widen
msqma.[b].mm  md, ms1, ms2      # signed int8, output quad-widen and saturated
```

## 6.4. Zmi16: Matrix 16-bit Integer Extension

The Zmi16 extension allows to use 16-bit integer as the data type of input matrix elements.

The Zmi16 extension adds a bit `mtype[5]` in `mtype` register.

*Table 8. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mint16` field, write 1 to enable 16-bit integer. 0 will be returned and `mtype.mill` will be set if 16-bit integer is not supported.

The `mint16` field can be set with other fields by `msettype[i]` or set independently by `msetint` or `munsetint`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetint   rd, int16     # rd = new mtype, set mint16 = 1 to enable INT16 type.
munsetint rd, int16     # rd = new mtype, set mint16 = 0 to disable INT16 type.
```

The e16 load/store and data move instructions are used for int16 data.

The element-wise and type-convert instructions with .h suffix are added for int16 format.

Four no-widen instructions are added to support int16 matrix multiplication. So the output type is always 16-bit integer.

```
mmau.[h].mm  md, ms1, ms2      # unsigned int16, output no-widen
msmau.[h].mm md, ms1, ms2      # unsigned int16, output no-widen and saturated

mma.[h].mm   md, ms1, ms2      # signed int16, output no-widen
```

```
msma.[h].mm  md, ms1, ms2      # signed int16, output no-widen and saturated
```

## 6.5. Zmi32: Matrix 32-bit Integer Extension

The Zmi32 extension allows to use 32-bit integer as the data type of input matrix elements.

The Zmi32 extension adds a bit `mtype[6]` in `mtype` register.

*Table 9. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mint32` field, write 1 to enable 32-bit integer. 0 will be returned and `mtype.mill` will be set if 32-bit integer is not supported.

The `mint32` field can be set with other fields by `msettype[i]` or set independently by `msetint` or `munsetint`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetint   rd, int32     # rd = new mtype, set mint32 = 1 to enable INT32 type.
munsetint rd, int32     # rd = new mtype, set mint32 = 0 to disable INT32 type.
```

The e32 load/store and data move instructions are used for int32 data.

The element-wise and type-convert instructions with .w suffix are added for int32 format.

Four no-widen instructions are added to support int32 matrix multiplication. So the output type is always 32-bit integer.

```
mmau.[w].mm  md, ms1, ms2      # unsigned int32, output no-widen
msmau.[w].mm md, ms1, ms2      # unsigned int32, output no-widen and saturated

mma.[w].mm   md, ms1, ms2      # signed int32, output no-widen
msma.[w].mm  md, ms1, ms2      # signed int32, output no-widen and saturated
```

## 6.6. Zmi64: Matrix 64-bit Integer Extension

The Zmi64 extension allows to use 64-bit integer as the data type of input matrix elements.

The Zmi64 extension adds a bit `mtype[7]` in `mtype` register.

*Table 10. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mint64` field, write 1 to enable 64-bit integer. 0 will be returned and `mtype.mill` will be set if 64-bit integer is not supported.

The `mint64` field can be set with other fields by `msettype[i]` or set independently by `msetint` or `munsetint`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetint   rd, int64     # rd = new mtype, set mint64 = 1 to enable INT64 type.
munsetint rd, int64     # rd = new mtype, set mint64 = 0 to disable INT64 type.
```

The e64 load/store and data move instructions are used for int64 data.

The element-wise and type-convert instructions with .dw suffix are added for int64 format.

Four no-widen instructions are added to support int64 matrix multiplication. So the output type is always 64-bit integer.

```
mmau.[dw].mm  md, ms1, ms2      # unsigned int64, output no-widen
msmau.[dw].mm md, ms1, ms2      # unsigned int64, output no-widen and saturated

mma.[dw].mm   md, ms1, ms2      # signed int64, output no-widen
msma.[dw].mm  md, ms1, ms2      # signed int64, output no-widen and saturated
```

## 6.7. Zmf8e4m3: Matrix 8-bit E4M3 Float Point Extension

The Zmf8e4m3 extension allows to use 8-bit float point format with 4-bit exponent and 3-bit mantissa as the data type of input matrix elements.

The Zmf8e4m3 extension uses a 2-bit `mfp8` field, `mtype[9:8]`, in `mtype` register.

*Table 11. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |

| Bits | Name | Description |
|---|---|---|
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp8` field, write 01 to enable 8-bit E4M3 float point. 0 will be returned and `mtype.mill` will be set if E4M3 is not supported.

The `mfp8` field can be set with other fields by `msettype[i]` or set independently by `msetfp` or `munsetfp`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetfp    rd, fp8       # rd = new mtype, set mfp8 = 01 to enable E4M3 type.
msetfp    rd, e4m3      # rd = new mtype, set mfp8 = 01 to enable E4M3 type.
munsetfp  rd, fp8       # rd = new mtype, set mfp8 = 00 to disable FP8 type.
```

The e8 load/store and data move instructions are used for E4M3 data.

The element-wise and type-convert instructions with .cf suffix are added for E4M3 format.

A double-widen instruction and a quadruple-widen instruction are added to support E4M3 matrix multiplication. So the output type is 16-bit or 32-bit float point.

```
mfwma.[cf].mm md, ms1, ms2      # 8-bit float point, output double-widen
mfqma.[cf].mm md, ms1, ms2      # 8-bit float point, output quad-widen
```

## 6.8. Zmf8e5m2: Matrix 8-bit E5M2 Float Point Extension

The Zmf8e5m2 extension allows to use 8-bit float point format with 5-bit exponent and 2-bit mantissa as the data type of input matrix elements.

The Zmf8e5m2 extension uses a 2-bit `mfp8` field, `mtype[9:8]`, in `mtype` register.

*Table 12. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |

| Bits | Name | Description |
|---|---|---|
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp8` field, write 10 to enable 8-bit E5M2 float point. 0 will be returned and `mtype.mill` will be set if E5M2 is not supported.

The `mfp8` field can be set with other fields by `msettype[i]` or set independently by `msetfp` or `munsetfp`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetfp    rd, e5m2      # rd = new mtype, set mfp8 = 10 to enable E5M2 type.
munsetfp  rd, fp8       # rd = new mtype, set mfp8 = 00 to disable FP8 type.
```

The e8 load/store and data move instructions are used for E5M2 data.

The element-wise and type-convert instructions with .cf suffix are added for E5M2 format.

A double-widen instruction and a quadruple-widen instruction are added to support E5M2 matrix multiplication. So the output type is 16-bit or 32-bit float point.

```
mfwma.[cf].mm md, ms1, ms2      # 8-bit float point, output double-widen
mfqma.[cf].mm md, ms1, ms2      # 8-bit float point, output quad-widen
```

## 6.9. Zmf8e3m4: Matrix 8-bit E3M4 Float Point Extension

The Zmf8e3m4 extension allows to use 8-bit float point format with 3-bit exponent and 4-bit mantissa as the data type of input matrix elements.

The Zmf8e3m4 extension uses a 2-bit `mfp8` field, `mtype[9:8]`, in `mtype` register.

*Table 13. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp8` field, write 11 to enable 8-bit E3M4 float point. 0 will be returned and `mtype.mill` will be set if E3M4 is not supported.

The `mfp8` field can be set with other fields by `msettype[i]` or set independently by `msetfp` or `munsetfp`.

```
msettypei rd, imm       # rd = new mtype, imm = new mtype setting.
msettype  rd, rs1       # rd = new mtype, rs1 = new mtype value.

msetfp    rd, e3m4      # rd = new mtype, set mfp8 = 11 to enable E3M4 type.
munsetfp  rd, fp8       # rd = new mtype, set mfp8 = 00 to disable FP8 type.
```

The e8 load/store and data move instructions are used for E3M4 data.

The element-wise and type-convert instructions with .cf suffix are added for E3M4 format.

A double-widen instruction and a quadruple-widen instruction are added to support E3M4 matrix multiplication. So the output type is 16-bit or 32-bit float point.

```
mfwma.[cf].mm md, ms1, ms2      # 8-bit float point, output double-widen
mfqma.[cf].mm md, ms1, ms2      # 8-bit float point, output quad-widen
```

## 6.10. Zmf16e5m10: Matrix 16-bit Half-precision Float-point (FP16) Extension

The Zmf16e5m10 extension allows to use FP16 format with 5-bit exponent and 10-bit mantissa as the data type of input matrix elements.

The Zmf16e5m10 extension uses a 2-bit `mfp16` field, `mtype[11:10]`, in `mtype` register.

*Table 14. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp16` field, write 01 to enable 16-bit E5M10 float point (FP16). 0 will be returned and `mtype.mill` will be set if FP16 is not supported.

The `mfp16` field can be set with other fields by `msettype[hi]` or set independently by `msetfp` or `munsetfp`.

```
msettypehi rd, imm      # rd = new mtype, imm = new mtype setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.

msetfp     rd, fp16     # rd = new mtype, set mfp16 = 01 to enable FP16 type.
munsetfp   rd, fp16     # rd = new mtype, set mfp16 = 00 to disable FP16 type.
```

The e16 load/store and data move instructions are used for FP16 data.

The element-wise and type-convert instructions with .hf suffix are added for FP16 format.

A no-widen instruction and a double-widen instruction are added to support FP16 matrix multiplication. So the output type is 16-bit or 32-bit float point.

```
mfma.[hf].mm  md, ms1, ms2      # 16-bit float point, output no-widen
mfwma.[hf].mm md, ms1, ms2      # 16-bit float point, output double-widen
```

## 6.11. Zmf16e8m7: Matrix 16-bit BFloat (BF16) Extension

The Zmf16e8m7 extension allows to use BF16 format with 8-bit exponent and 7-bit mantissa as the data type of input matrix elements.

The Zmf16e8m7 extension uses a 2-bit `mfp16` field, `mtype[11:10]`, in `mtype` register.

*Table 15. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |

| Bits | Name | Description |
|---|---|---|
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp16` field, write 10 to enable 16-bit E8M7 float point (BF16). 0 will be returned and `mtype.mill` will be set if BF16 is not supported.

The `mfp16` field can be set with other fields by `msettype[hi]` or set independently by `msetfp` or `munsetfp`.

```
msettypehi rd, imm      # rd = new mtype, imm = new mtype setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.

msetfp     rd, bf16     # rd = new mtype, set mfp16 = 10 to enable BF16 type.
munsetfp   rd, fp16     # rd = new mtype, set mfp16 = 00 to disable BF16 type.
```

The e16 load/store and data move instructions are used for BF16 data.

The element-wise and type-convert instructions with .hf suffix are reused for BF16 format.

A no-widen instruction and a double-widen instruction are added to support BF16 matrix multiplication. So the output type is 16-bit or 32-bit float point.

```
mfma.[hf].mm  md, ms1, ms2      # 16-bit float point, output no-widen
mfwma.[hf].mm md, ms1, ms2      # 16-bit float point, output double-widen
```

## 6.12. Zmf32e8m23: Matrix 32-bit Float-point Extension

The Zmf32e8m23 extension allows to use standard FP32 format with 8-bit exponent and 23-bit mantissa as the data type of input matrix elements.

The Zmf32e8m23 extension uses a 2-bit `mfp32` field, `mtype[13:12]`, in `mtype` register.

*Table 16. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |

| Bits | Name | Description |
|---|---|---|
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp32` field, write 01 to enable 32-bit E8M23 float point (FP32). 0 will be returned and `mtype.mill` will be set if FP32 is not supported.

The `mfp32` field can be set with other fields by `msettype[hi]` or set independently by `msetfp` or `munsetfp`.

```
msettypehi rd, imm      # rd = new mtype, imm = new mtype setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.

msetfp     rd, fp32     # rd = new mtype, set mfp32 = 01 to enable FP32 type.
munsetfp   rd, fp32     # rd = new mtype, set mfp32 = 00 to disable FP32 type.
```

The e32 load/store and data move instructions are used for FP32 data.

The element-wise and type-convert instructions with .f suffix are added for FP32 format.

A no-widen instructionis added to support FP32 matrix multiplication. So the output type is 32-bit float point.

```
mfma.[f].mm  md, ms1, ms2       # 32-bit float point, output no-widen
```

## 6.13. Zmf19e8m10: Matrix 19-bit TensorFloat32 (TF32) Extension

The Zmf19e8m10 extension allows to use TF32 format with 8-bit exponent and 10-bit mantissa as the data type of input matrix elements.

The Zmf19e8m10 extension uses a 2-bit `mfp32` field, `mtype[13:12]`, in `mtype` register.

*Table 17. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp32` field, write 10 to enable 19-bit E8M10 float point (TF32). 0 will be returned and `mtype.mill` will be set if TF32 is not supported.

The `mfp32` field can be set with other fields by `msettype[hi]` or set independently by `msetfp` or `munsetfp`.

```
msettypehi rd, imm      # rd = new mtype, imm = new mtype setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.

msetfp     rd, tf32     # rd = new mtype, set mfp32 = 10 to enable TF32 type.
munsetfp   rd, fp32     # rd = new mtype, set mfp32 = 00 to disable FP32 type.
```

TF32 implementions are designed to achieve better performance on matrix multiplications and convolutions by rounding input Float32 data to have 10 bits of mantissa, and accumulating results with FP32 precision, maintaining FP32 dynamic range.

So when Zmtf32 is used, Float32 is still used as the input and output data type for matrix multiplication.

The e32 load/store and data move instructions are used for TF32 data.

The element-wise and type-convert instructions are not supported for TF32 format.

A no-widen instruction is added to support TF32 matrix multiplication. So the output type is always 32-bit float point (FP32).

```
mfma.[f].mm  md, ms1, ms2       # 19-bit float point, output no-widen
```

There is no double-widen version for TF32 matrix multiplication (a double-widen version for standard FP32 is supported by Zmf64e11m52 extension).

## 6.14. Zmf64e11m52: Matrix 64-bit Float-point Extension

The Zmf64e11m52 extension allows to use standard FP64 format with 11-bit exponent and 52-bit mantissa as the data type of input matrix elements.

The Zmf64e11m52 extension uses a 1-bit `mfp64` field, `mtype[14]`, in `mtype` register.

*Table 18. `mtype` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN-1 | mill | Illegal value if set. |
| XLEN-2:16 | 0 | Reserved if non-zero. |
| 15 | mba | Matrix out of bound agnostic. |
| 14 | mfp64 | 64-bit float point enabling. |
| 13:12 | mfp32[1:0] | 32-bit float point enabling. |
| 11:10 | mfp16[1:0] | 16-bit float point enabling. |
| 9:8 | mfp8[1:0] | 8-bit float point enabling. |
| 7 | mint64 | 64-bit integer enabling. |
| 6 | mint32 | 32-bit integer enabling. |
| 5 | mint16 | 16-bit integer enabling. |
| 4 | mint8 | 8-bit integer enabling. |
| 3 | mint4 | 4-bit integer enabling. |
| 2:0 | msew[2:0] | Selected element width (SEW) setting. |

For `mfp64` field, write 1 to enable 64-bit E11M52 float point (FP64). 0 will be returned and `mtype.mill` will be set if FP64 is not supported.

The `mfp64` field can be set with other fields by `msettype[hi]` or set independently by `msetfp` or `munsetfp`.

```
msettypehi rd, imm      # rd = new mtype, imm = new mtype setting.
msettype   rd, rs1      # rd = new mtype, rs1 = new mtype value.

msetfp     rd, fp64     # rd = new mtype, set mfp64 = 1 to enable FP64 type.
munsetfp   rd, fp64     # rd = new mtype, set mfp64 = 0 to disable FP64 type.
```

The e64 load/store and data move instructions are used for FP64 data.

The element-wise and type-convert instructions with .d suffix are added for FP64 format.

A no-widen instruction is added to support FP64 matrix multiplication. And a double-widen instruction is added to support FP32 widening matrix multiplication. The output type is always 64-bit float point (FP64).

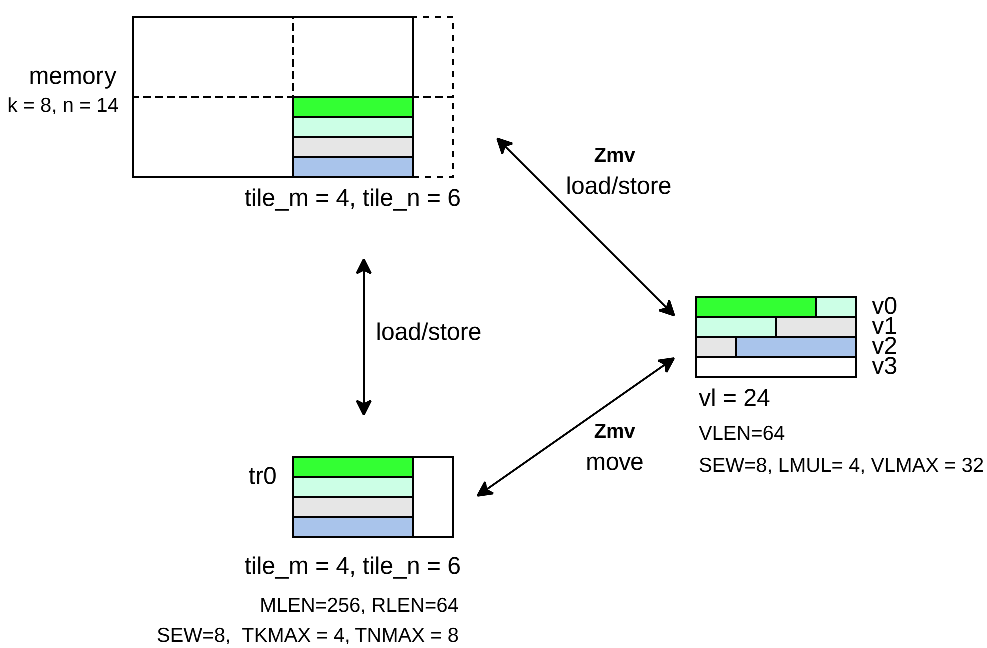
```
mfma.[d].mm  md, ms1, ms2       # 64-bit float point, output no-widen
mfwma.[f].mm md, ms1, ms2       # 32-bit float point, output double-widen
```

## 6.15. Zmv: Matrix for Vector operations

The Zmv extension is defined to provide matrix support with the RISC-V Vector "V" extension.

The Zmv extension allows to load matrix tile slices into vector registers, and move data between slices of a matrix register and vector registers.

The data layout examples of registers and memory in Zmv are shown below.



### 6.15.1. Load Instructions

```
# vd destination, rs1 base address, rs2 row byte stride
# lmul / (eew/sew) rows or columns

# for left matrix, a
mlae8.v    vd, (rs1), rs2 #  8-bit tile slices load to vregs
mlae16.v   vd, (rs1), rs2 # 16-bit tile slices load to vregs
mlae32.v   vd, (rs1), rs2 # 32-bit tile slices load to vregs
mlae64.v   vd, (rs1), rs2 # 64-bit tile slices load to vregs

# for right matrix, b
mlbe8.v    vd, (rs1), rs2 #  8-bit tile slices load to vregs
mlbe16.v   vd, (rs1), rs2 # 16-bit tile slices load to vregs
mlbe32.v   vd, (rs1), rs2 # 32-bit tile slices load to vregs
mlbe64.v   vd, (rs1), rs2 # 64-bit tile slices load to vregs

# for output matrix, c
mlce8.v    vd, (rs1), rs2 #  8-bit tile slices load to vregs
mlce16.v   vd, (rs1), rs2 # 16-bit tile slices load to vregs
mlce32.v   vd, (rs1), rs2 # 32-bit tile slices load to vregs
mlce64.v   vd, (rs1), rs2 # 64-bit tile slices load to vregs
```

### 6.15.2. Store Instructions

```
# vs3 store data, rs1 base address, rs2 row byte stride
# lmul / (eew/sew) rows or columns

# for left matrix, a
msae8.v    vs3, (rs1), rs2 #  8-bit tile slices store from vregs
msae16.v   vs3, (rs1), rs2 # 16-bit tile slices store from vregs
msae32.v   vs3, (rs1), rs2 # 32-bit tile slices store from vregs
msae64.v   vs3, (rs1), rs2 # 64-bit tile slices store from vregs

# for right matrix, b
msbe8.v    vs3, (rs1), rs2 #  8-bit tile slices store from vregs
msbe16.v   vs3, (rs1), rs2 # 16-bit tile slices store from vregs
msbe32.v   vs3, (rs1), rs2 # 32-bit tile slices store from vregs
msbe64.v   vs3, (rs1), rs2 # 64-bit tile slices store from vregs

# for output matrix, c
msce8.v    vs3, (rs1), rs2 #  8-bit tile slices store from vregs
msce16.v   vs3, (rs1), rs2 # 16-bit tile slices store from vregs
msce32.v   vs3, (rs1), rs2 # 32-bit tile slices store from vregs
msce64.v   vs3, (rs1), rs2 # 64-bit tile slices store from vregs
```

### 6.15.3. Data Move Instructions

For data moving between vector and matrix, the vsew of vector must equal to msew of matrix.

The number of elements moved is min(VLEN/SEW * VLMUL, matrix_size).

- For matrix A, matrix_size = mtilem * mtilek.
- For matrix B, matrix_size = mtilek * mtilen.
- For matrix C, matrix_size = mtilem * mtilen.

```
# Data move between matrix register rows and vector registers.

# vd[(i - rs2) * mtilek + j] = md[i, j], i = rs2 .. rs2 + mtilem - 1
mmvare8.v.m   vd, ms1, rs2
mmvare16.v.m  vd, ms1, rs2
mmvare32.v.m  vd, ms1, rs2
mmvare64.v.m  vd, ms1, rs2

# vd[(i - rs2) * mtilen + j] = md[i, j], i = rs2 .. rs2 + mtilek - 1
mmvbre8.v.m   vd, ms1, rs2
mmvbre16.v.m  vd, ms1, rs2
mmvbre32.v.m  vd, ms1, rs2
mmvbre64.v.m  vd, ms1, rs2

# vd[(i - rs2) * mtilen + j] = md[i, j], i = rs2 .. rs2 + mtilem - 1
mmvcre8.v.m   vd, ms1, rs2
mmvcre16.v.m  vd, ms1, rs2
mmvcre32.v.m  vd, ms1, rs2
mmvcre64.v.m  vd, ms1, rs2

# md[i, j] = vd[(i - rs2) * mtilek + j], i = rs2 .. rs2 + mtilem - 1
mmvare8.m.v   md, vs1, rs2
mmvare16.m.v  md, vs1, rs2
mmvare32.m.v  md, vs1, rs2
mmvare64.m.v  md, vs1, rs2

# md[i, j] = vd[(i - rs2) * mtilen + j], i = rs2 .. rs2 + mtilek - 1
mmvbre8.m.v   md, vs1, rs2
mmvbre16.m.v  md, vs1, rs2
mmvbre32.m.v  md, vs1, rs2
mmvbre64.m.v  md, vs1, rs2

# md[i, j] = vd[(i - rs2) * mtilen + j], i = rs2 .. rs2 + mtilem - 1
mmvcre8.m.v   md, vs1, rs2
mmvcre16.m.v  md, vs1, rs2
mmvcre32.m.v  md, vs1, rs2
mmvcre64.m.v  md, vs1, rs2
```

```
# Data move between matrix register columns and vector registers.

# vd[(j - rs2) * mtilem + i] = md[i, j], j = rs2 .. rs2 + mtilek - 1
mmvace8.v.m   vd, ms1, rs2
mmvace16.v.m  vd, ms1, rs2
mmvace32.v.m  vd, ms1, rs2
mmvace64.v.m  vd, ms1, rs2

# vd[(j - rs2) * mtilek + i] = md[i, j], j = rs2 .. rs2 + mtilen - 1
mmvbce8.v.m   vd, ms1, rs2
mmvbce16.v.m  vd, ms1, rs2
mmvbce32.v.m  vd, ms1, rs2
mmvbce64.v.m  vd, ms1, rs2

# vd[(j - rs2) * mtilem + i] = md[i, j], j = rs2 .. rs2 + mtilen - 1
mmvcce8.v.m   vd, ms1, rs2
mmvcce16.v.m  vd, ms1, rs2
mmvcce32.v.m  vd, ms1, rs2
mmvcce64.v.m  vd, ms1, rs2

# md[i, j] = vd[(j - rs2) * mtilem + i], j = rs2 .. rs2 + mtilek - 1
mmvace8.m.v   md, vs1, rs2
mmvace16.m.v  md, vs1, rs2
mmvace32.m.v  md, vs1, rs2
mmvace64.m.v  md, vs1, rs2

# md[i, j] = vd[(j - rs2) * mtilek + i], j = rs2 .. rs2 + mtilen - 1
mmvbce8.m.v   md, vs1, rs2
mmvbce16.m.v  md, vs1, rs2
mmvbce32.m.v  md, vs1, rs2
mmvbce64.m.v  md, vs1, rs2

# md[i, j] = vd[(j - rs2) * mtilem + i], j = rs2 .. rs2 + mtilen - 1
mmvcce8.m.v   md, vs1, rs2
mmvcce16.m.v  md, vs1, rs2
mmvcce32.m.v  md, vs1, rs2
mmvcce64.m.v  md, vs1, rs2
```

### 6.15.4. Intrinsic Example: Matrix multiplication fused with element-wise vector operation

```
void fused_matmul_relu_float16(c, a, b, m, k, n) {
    msettype(e16);                             // use 16bit input matrix element
    for (i = 0; i < m; i += tile_m) {          // loop at dim m with tiling
        tile_m = msettilem(m-i);
        for (j = 0; j < n; j += tile_n) {      // loop at dim n with tiling
```

```
            tile_n = msettilen(n-j);

            out = mwsub_mm(out, out)            // clear acc reg
            for (s = 0; s < k; s += tile_k) {   // loop at dim k with tiling
                tile_k = msettilek(k-s);

                tr1 = mlae16_m(&a[i][s]);       // load left matrix a
                tr2 = mlbe16_m(&b[s][j]);       // load right matrix b
                out = mfwma_mm(tr1, tr2);       // tiled matrix multiply,
                                                // double widen output
            }

            out = mfncvt_f_fw_m(out, m2);       // convert widen result to single

            for (s = 0; s < tile_m; s += rows) {
                // max rows could move into 8 vregs
                rows = min(tile_m - s, 8*vlenb/rlenb);
                vsetvl(tile_n*rows, e16, m8);

                v1 = mmvcr_v_m(out, s);         // move out rows to vreg
                v1 = vfmax_vf(0.f, v1);         // vfmax.vf for relu

                msce16_v(v1, &c[i+s][j], n);    // store output tile slices
            }
        }
    }
}
```

## 6.16. Zmi2c: Im2col Extension

Im2col stands for Image to Column, and is an implementation technique of computing Convolution operation (in Machine Learning) using GEMM operations.

The Zmi2c extension allows to perform im2col operation on-the-fly, by a set of new load instructions.

The **Load Unfold** instructions allows to load and extract sliding local blocks from memory into the matrix tile registers.

### 6.16.1. CSRs

The matrix extension adds 7 unprivileged CSRs (moutsh, minsh, mpad, mstdi, minsk, moutsk, mpadval) to the base scalar RISC-V ISA.

*Table 19. New matrix CSRs*

| Address | Privilege | Name | Description |
|---|---|---|---|
| 0xC47 | URO | moutsh | Fold/unfold output shape. |
| 0xC48 | URO | minsh | Fold/unfold input shape. |
| 0xC49 | URO | mpad | Fold/unfold padding parameters. |
| 0xC4A | URO | mstdi | Fold/unfold sliding strides and dilations. |
| 0xC4B | URO | minsk | Fold/unfold sliding kernel position of input. |
| 0xC4C | URO | moutsk | Fold/unfold sliding kernel position of output. |
| 0xC4D | URO | mpadval | Fold/unfold padding value, default to zero. |

*Table 20. `minsh` `moutsh` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN:32 | 0 | Reserved |
| 31:16 | shape[1] | shape of dim 1, height |
| 15:0 | shape[0] | shape of dim 0, width |

*Table 21. `mpad` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN:32 | 0 | Reserved |
| 31:24 | mpad_top | Padding added to up side of input |
| 23:16 | mpad_bottom | Padding added to bottom side of input |
| 15:8 | mpad_left | Padding added to left side of input |
| 7:0 | mpad_right | Padding added to left side of input |

*Table 22. `mstdi` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN:32 | 0 | Reserved |
| 31:24 | mdil_h | Height spacing of the kernel elements |
| 23:16 | mdil_w | Weight spacing of the kernel elements |
| 15:8 | mstr_h | Height stride of the convolution |
| 7:0 | mstr_w | Weight stride of the convolution |

*Table 23. `minsk` `moutsk` register layout*

| Bits | Name | Description |
|---|---|---|
| XLEN:32 | 0 | Reserved |
| 31:16 | msk[1] | Sliding kernel position of dim 1, height |
| 15:0 | msk[0] | Sliding kernel position of dim 0, width |

### 6.16.2. Configuration Instructions

```
msetoutsh  rd, rs1, rs2 # set output shape(rs1), strides and dilations(rs2)
msetinsh   rd, rs1, rs2 # set input shape(rs1) and padding(rs2)
msetsk     rd, rs1, rs2 # set fold/unfold sliding positions, insk(rs1), outsk(rs2)
msetpadval rd, rs1      # set fold/unfold padding value
```

### 6.16.3. Load Unfold Instructions

The **Load Unfold** instructions allows to load and extract sliding local blocks from memory into the matrix tile registers. Similar to PyTorch, for the case of two input spatial dimensions this operation is sometimes called `im2col`.

```
# md destination, rs1 base address, rs2 row byte stride

# for left matrix, a
mlufae8.m    md, (rs1), rs2
mlufae16.m   md, (rs1), rs2
mlufae32.m   md, (rs1), rs2
mlufae64.m   md, (rs1), rs2

# for left matrix, b
mlufbe8.m    md, (rs1), rs2
mlufbe16.m   md, (rs1), rs2
mlufbe32.m   md, (rs1), rs2
mlufbe64.m   md, (rs1), rs2

# for left matrix, c
mlufce8.m    md, (rs1), rs2
mlufce16.m   md, (rs1), rs2
mlufce32.m   md, (rs1), rs2
mlufce64.m   md, (rs1), rs2
```

### 6.16.4. Intrinsic Example: Conv2D

```
void conv2d_float16(c, a, b, outh, outw, outc, inh, inw, inc,
        kh, kw, pt, pb, pl, pr, sw, dh, dw) {
```

```
    m = outh * outw;
    k = kh * kw * inc;
    n = outc;

    msettype(e16);      // use 16bit input matrix element

    // set in/out shape, sliding strides and dilations, and padding
    msetoutsh(outh << 16 | outw, dh << 24 | dw << 16 | sh << 8 | sw);
    msetinsh(inh << 16 | inw, pt << 24 | pb << 16 | pl << 8 | pr);

    for (i = 0; i < m; i += tile_m) {              // loop at dim m with tiling
        tile_m = msettilem(m-i);

        outh_pos = i / outw;
        outw_pos = i - outh_pos * outw;

        for (j = 0; j < n; j += tile_n) {          // loop at dim n with tiling
            tile_n = msettilen(n-j);

            out = mwsub_mm(out, out)               // clear output reg
            for (skh = 0; skh < kh; skh++) {       // loop for kernel height
                inh_pos = outh_pos * sh - pt + skh * dh;
                for (skw = 0; skw < kw; skw++) {    // loop for kernel width
                    inw_pos = outw_pos * sw - pl + skw * dw;

                    // set sliding position
                    msetsk(inh_pos << 16 | inw_pos, skw * dw << 16 | outw_pos)

                    // loop for kernel channels
                    for (skc = 0; skc < inc; skc += tile_k) {
                        tile_k = msettilek(inc-skc);

                        tr1 = mlufae16_m(&a[inh_pos][inw_pos][skc]);
                                                   // load and unfold input blocks
                        tr2 = mlbe16_m(&b[s][j]);   // load right matrix b
                        out = mfwma_mm(tr1, tr2);   // tiled matrix multiply,
                                                   // double widen output
                    }
                }
            }

            out = mfncvt_f_fw_m(out, m2);          // convert widen result
            msce16_m(out, &c[i][j], n*2);          // store to matrix c
        }
    }
}
```

### 6.16.5. Intrinsic Example: Conv3D

```
void conv3d_float16(c, a, b, outh, outw, outc, ind, inh, inw, inc,
        kd, kh, kw, pt, pb, pl, pr, sw, dh, dw) {
    m = outh * outw;
    k = kd * kh * kw * inc;
    n = outc;

    msettype(e16);      // use 16bit input matrix element

    // set in/out shape, sliding strides and dilations, and padding
    msetoutsh(outh << 16 | outw, dh << 24 | dw << 16 | sh << 8 | sw);
    msetinsh(inh << 16 | inw, pt << 24 | pb << 16 | pl << 8 | pr);

    for (i = 0; i < m; i += tile_m) {               // loop at dim m with tiling
        tile_m = msettilem(m-i);

        outh_pos = i / outw;
        outw_pos = i - outh_pos * outw;

        for (j = 0; j < n; j += tile_n) {           // loop at dim n with tiling
            tile_n = msettilen(n-j);

            out = mwsub_mm(out, out)                // clear output reg
            for (skd = 0; skd < kd; skd++) {        // loop for kernel *depth*
                for (skh = 0; skh < kh; skh++) {    // loop for kernel height
                    inh_pos = outh_pos * sh - pt + skh * dh;
                    for (skw = 0; skw < kw; skw++) {    // loop for kernel width
                        inw_pos = outw_pos * sw - pl + skw * dw;

                        msetsk(inh_pos << 16 | inw_pos, skw * dw << 16 | outw_pos)
                                                        // set sliding position

                        for (skc = 0; skc < inc; skc += tile_k) {
                            tile_k = msettilek(inc-skc);

                            tr1 = mlufae16_m(&a[skd][inh_pos][inw_pos][skc]);
                                                        // load and unfold blocks
                            tr2 = mlbe16_m(&b[s][j]);   // load right matrix b
                            out = mfwma_mm(tr1, tr2);   // tiled matrix multiply,
                                                        // double widen output
                        }
                    }
                }
            }

            out = mfncvt_f_fw_m(out, m2);   // convert widen result
            msce16_m(out, &c[i][j], n*2);   // store to matrix c
```

```
        }
    }
}
```

### 6.16.6. Intrinsic Example: MaxPool2D

```
void maxpool2d_float16(out, in, outh, outw, outc, inh, inw, inc,
        kh, kw, pt, pb, pl, pr, sw, dh, dw) {
    m = outh * outw;
    n = outc;

    msettype(e16);      // use 16bit input matrix element

    // set in/out shape, sliding strides and dilations, and padding
    msetoutsh(outh << 16 | outw, dh << 24 | dw << 16 | sh << 8 | sw);
    msetinsh(inh << 16 | inw, pt << 24 | pb << 16 | pl << 8 | pr);

    for (i = 0; i < m; i += tile_m) {           // loop at dim m with tiling
        tile_m = msettilem(m-i);

        outh_pos = i / outw;
        outw_pos = i - outh_pos * outw;

        for (j = 0; j < n; j += tile_n) {       // loop at dim n with tiling
            tile_n = msettilen(n-j);

            m_out = mfmv_s_f(tr_out, -inf)     // move -inf to output reg
            m_out = mbcce_m (tr_out)           // fill -inf to output reg
            for (skh = 0; skh < kh; skh++) {    // loop for kernel height
                inh_pos = outh_pos * sh - pt + skh * dh;
                for (skw = 0; skw < kw; skw++) {        // loop for kernel width
                    inw_pos = outw_pos * sw - pl + skw * dw;

                    msetsk(inh_pos << 16 | inw_pos, skw * dw << 16 | outw_pos)
                                                    // set sliding position

                    // load and unfold matrix blocks
                    m_in = mlufce16_m(&in[inh_pos][inw_pos][j]);
                    m_out = mfmax_mm(m_out, m_in);
                }
            }

            msce16_m(tr_out, &out[i][j], n*2);  // store to matrix c
        }
    }
}
```

### 6.16.7. Intrinsic Example: AvgPool2D

```
void avgpool2d_float16(out, in, outh, outw, outc, inh, inw, inc,
        kh, kw, pt, pb, pl, pr, sw, dh, dw) {
    m = outh * outw;
    n = outc;

    msettype(e16);      // use 16bit input matrix element

    // set in/out shape, sliding strides and dilations, and padding
    msetoutsh(outh << 16 | outw, dh << 24 | dw << 16 | sh << 8 | sw);
    msetinsh(inh << 16 | inw, pt << 24 | pb << 16 | pl << 8 | pr);

    // set divider
    m_div = mfmv_s_f(m_div, kh*kw)
    m_div = mbcce_m (m_div)

    for (i = 0; i < m; i += tile_m) {   // loop at dim m with tiling
        tile_m = msettilem(m-i);

        outh_pos = i / outw;
        outw_pos = i - outh_pos * outw;

        for (j = 0; j < n; j += tile_n) {   // loop at dim n with tiling
            tile_n = msettilen(n-j);

            m_out = mwsub_mm(m_out, m_out)          // clear output reg
            for (skh = 0; skh < kh; skh++) {        // loop for kernel height
                inh_pos = outh_pos * sh - pt + skh * dh;
                for (skw = 0; skw < kw; skw++) {    // loop for kernel width
                    inw_pos = outw_pos * sw - pl + skw * dw;

                    msetsk(inh_pos << 16 | inw_pos, skw * dw << 16 | outw_pos)
                                                    // set sliding position

                    // load and unfold matrix blocks
                    m_in = mlufce16_m(&in[inh_pos][inw_pos][j]);
                    m_out = mfadd_mm(m_out, m_in);
                }
            }

            m_out = mfdiv_mm(m_out, m_div);
            msce16_m(m_out, &out[i][j], n*2);      // store to matrix c
        }
    }
}
```

## 6.17. Zmc2i: Col2im Extension

The Zmc2i extension allows to perform Column to Image operation on-the-fly, by a set of new store instructions.

### 6.17.1. CSRs

The Zmc2i extension reuses 7 unprivileged CSRs (moutsh, minsh, mpad, mstdi, minsk, moutsk, mpadval) of Zmi2c.

### 6.17.2. Configuration Instructions

The Zmc2i extension reuses all configuration instructions of Zmi2c.

### 6.17.3. Store Fold Instructions

The **Store Fold** instructions allows to store and combine an array of sliding local blocks from the matrix tile regstiers into memory. Similar to PyTorch, for the case of two output spatial dimensions this operation is sometimes called `col2im`.

```
# ms3 destination, rs1 base address, rs2 row byte stride

# for left matrix, a
msfdae8.m    ms3, (rs1), rs2
msfdae16.m   ms3, (rs1), rs2
msfdae32.m   ms3, (rs1), rs2
msfdae64.m   ms3, (rs1), rs2

# for left matrix, b
msfdbe8.m    ms3, (rs1), rs2
msfdbe16.m   ms3, (rs1), rs2
msfdbe32.m   ms3, (rs1), rs2
msfdbe64.m   ms3, (rs1), rs2

# for left matrix, c
msfdce8.m    ms3, (rs1), rs2
msfdce16.m   ms3, (rs1), rs2
msfdce32.m   ms3, (rs1), rs2
msfdce64.m   ms3, (rs1), rs2
```

## 6.18. Zmsp*: Matrix Sparsity Extension

The Zmspa extension allows to perform 2:4 sparsing for left matrix.

The Zmspb extension allows to perform 2:4 sparsing for right matrix.

The Zmsp* extension adds one unprivileged CSR, two configuration instructions, and two groups of matrix multiplication instructions, both for left matrix and right matrix.

*Table 24. Sparsity CSRs*

| Address | Privilege | Name | Description |
|---|---|---|---|
| 0xC4F | URO | mdsp | The direction of sparsity (0 for row and 1 for column). |

### 6.18.1. Configuration Instructions

The Zmsp* extension adds two configuration instruction to configure the source index register and sparsity direction.

```
# Set sparsity direction.
msetdspi  rd, imm   # rd = new mdsp, imm = direction
msetdsp   rd, rs1   # rd = new mdsp, rs1 = direction
```

An implementation may support one of sparsity directions or both two directions. The `msetdsp[i]` always returns the supported direction.

### 6.18.2. Matrix Multiplication Instructions

The Zmspa extension adds a group of matrix multiplication instructions for left matrix sparsity.

```
# Unigned integer sparsing matrix multiplication and add, md = md + ms1 * ms2.
mmau.spa.[dw].mm   md, ms1, ms2     # left matrix is sparsing
mmau.spa.[w].mm    md, ms1, ms2     # left matrix is sparsing
mmau.spa.[h].mm    md, ms1, ms2     # left matrix is sparsing
mqmau.spa.[b].mm   md, ms1, ms2     # left matrix is sparsing
momau.spa.[hb].mm  md, ms1, ms2     # left matrix is sparsing

msmau.spa.[dw].mm  md, ms1, ms2     # left matrix is sparsing
msmau.spa.[w].mm   md, ms1, ms2     # left matrix is sparsing
msmau.spa.[h].mm   md, ms1, ms2     # left matrix is sparsing
msqmau.spa.[b].mm  md, ms1, ms2     # left matrix is sparsing
msomau.spa.[hb].mm md, ms1, ms2     # left matrix is sparsing

# Signed integer sparsing matrix multiplication and add, md = md + ms1 * ms2.
mma.spa.[dw].mm    md, ms1, ms2     # left matrix is sparsing
mma.spa.[w].mm     md, ms1, ms2     # left matrix is sparsing
mma.spa.[h].mm     md, ms1, ms2     # left matrix is sparsing
mqma.spa.[b].mm    md, ms1, ms2     # left matrix is sparsing
moma.spa.[hb].mm   md, ms1, ms2     # left matrix is sparsing

msma.spa.[dw].mm   md, ms1, ms2     # left matrix is sparsing
msma.spa.[w].mm    md, ms1, ms2     # left matrix is sparsing
```

```
msma.spa.[h].mm    md, ms1, ms2     # left matrix is sparsing
msqma.spa.[b].mm   md, ms1, ms2     # left matrix is sparsing
msoma.spa.[hb].mm  md, ms1, ms2     # left matrix is sparsing

# Float point sparsing matrix multiplication and add, md = md + ms1 * ms2.
mfma.spa.[d].mm    md, ms1, ms2     # left matrix is sparsing
mfma.spa.[f].mm    md, ms1, ms2     # left matrix is sparsing
mfma.spa.[hf].mm   md, ms1, ms2     # left matrix is sparsing

mfwma.spa.[f].mm   md, ms1, ms2     # left matrix is sparsing
mfwma.spa.[hf].mm  md, ms1, ms2     # left matrix is sparsing
mfwma.spa.[cf].mm  md, ms1, ms2     # left matrix is sparsing
mfqma.spa.[cf].mm  md, ms1, ms2     # left matrix is sparsing
```

The Zmspb extension adds a group of matrix multiplication instructions for right matrix sparsity.

```
# Unigned integer sparsing matrix multiplication and add, md = md + ms1 * ms2.
mmau.spb.[dw].mm   md, ms1, ms2     # right matrix is sparsing
mmau.spb.[w].mm    md, ms1, ms2     # right matrix is sparsing
mmau.spb.[h].mm    md, ms1, ms2     # right matrix is sparsing
mqmau.spb.[b].mm   md, ms1, ms2     # right matrix is sparsing
momau.spb.[hb].mm  md, ms1, ms2     # right matrix is sparsing

msmau.spb.[dw].mm  md, ms1, ms2     # right matrix is sparsing
msmau.spb.[w].mm   md, ms1, ms2     # right matrix is sparsing
msmau.spb.[h].mm   md, ms1, ms2     # right matrix is sparsing
msqmau.spb.[b].mm  md, ms1, ms2     # right matrix is sparsing
msomau.spb.[hb].mm md, ms1, ms2     # right matrix is sparsing

# Signed integer sparsing matrix multiplication and add, md = md + ms1 * ms2.
mma.spb.[dw].mm    md, ms1, ms2     # right matrix is sparsing
mma.spb.[w].mm     md, ms1, ms2     # right matrix is sparsing
mma.spb.[h].mm     md, ms1, ms2     # right matrix is sparsing
mqma.spb.[b].mm    md, ms1, ms2     # right matrix is sparsing
moma.spb.[hb].mm   md, ms1, ms2     # right matrix is sparsing

msma.spb.[dw].mm   md, ms1, ms2     # right matrix is sparsing
msma.spb.[w].mm    md, ms1, ms2     # right matrix is sparsing
msma.spb.[h].mm    md, ms1, ms2     # right matrix is sparsing
msqma.spb.[b].mm   md, ms1, ms2     # right matrix is sparsing
msoma.spb.[hb].mm  md, ms1, ms2     # right matrix is sparsing

# Float point sparsing matrix multiplication and add, md = md + ms1 * ms2.
mfma.spb.[d].mm    md, ms1, ms2     # right matrix is sparsing
mfma.spb.[f].mm    md, ms1, ms2     # right matrix is sparsing
mfma.spb.[hf].mm   md, ms1, ms2     # right matrix is sparsing

mfwma.spb.[f].mm   md, ms1, ms2     # right matrix is sparsing
```

```
mfwma.spb.[hf].mm  md, ms1, ms2     # right matrix is sparsing
mfwma.spb.[cf].mm  md, ms1, ms2     # right matrix is sparsing
mfqma.spb.[cf].mm  md, ms1, ms2     # right matrix is sparsing
```

# Chapter 7. Matrix Instruction Listing

*Table 25. Configuration Instructions*

| **Format** | 63 43 | | | | 42 39 | 38 32 |
|---|---|---|---|---|---|---|
| | **imm[31:11]** | | | | **funct4** | **opcode** |
| | 31 26 | 25 20 | 19 15 | 14 12 | 11 7 | 6 0 |
| | **funct6** | **imm[10:5]** | **rs1** | **funct3** | **rd** | **suffix** |
| msettype | 0…00 | | | | 0000 | xxyyy11 |
| | 000000 | 000000 | rs1 | 000 | rd | 0111111 |
| msettypei | mtypei[31:11] | | | | 0000 | xxyyy11 |
| | 000001 | mtypei[10:0] | | 000 | rd | 0111111 |
| msetsew | 0…00 | | | | 0000 | xxyyy11 |
| | 000011 | setval | | 000 | rd | 0111111 |
| msetint | 0…00 | | | | mtf | xxyyy11 |
| | 000011 | 1 | | 000 | rd | 0111111 |
| munsetint | 0…00 | | | | mtf | xxyyy11 |
| | 000011 | 0 | | 000 | rd | 0111111 |
| msetfp | 0…00 | | | | mtf | xxyyy11 |
| | 000011 | setval | | 000 | rd | 0111111 |
| munsetfp | 0…00 | | | | mtf | xxyyy11 |
| | 000011 | 0 | | 000 | rd | 0111111 |
| msetba | 0…00 | | | | 1010 | xxyyy11 |
| | 000011 | setval | | 000 | rd | 0111111 |
| msettilem | 0…00 | | | | 0000 | xxyyy11 |
| | 000100 | 000000 | rs1 | 000 | rd | 0111111 |
| msettilemi | mtypei[31:11] | | | | 0000 | xxyyy11 |
| | 000101 | mtypei[10:0] | | 000 | rd | 0111111 |
| msettilen | 0…00 | | | | 0000 | xxyyy11 |
| | 001000 | 000000 | rs1 | 000 | rd | 0111111 |

| | | | | | | |
|---|---|---|---|---|---|---|
| msettileni | mtypei[31:11] | | | | 0000 | xxyyy11 |
| | 001001 | mtypei[10:0] | | 000 | rd | 0111111 |
| msettilen | 0…00 | | | | 0000 | xxyyy11 |
| | 001100 | 000000 | rs1 | 000 | rd | 0111111 |
| msettileni | mtypei[31:11] | | | | 0000 | xxyyy11 |
| | 001101 | mtypei[10:0] | | 000 | rd | 0111111 |
| msetoutsh | 0…00 | | | | 0000 | xxyyy11 |
| | 010000 | 000000 | rs1 | 000 | rd | 0111111 |
| msetinsh | 0…00 | | | | 0000 | xxyyy11 |
| | 010100 | 000000 | rs1 | 000 | rd | 0111111 |
| msetsk | 0…00 | | | | 0000 | xxyyy11 |
| | 011000 | 000000 | rs1 | 000 | rd | 0111111 |
| msetpadval | 0…00 | | | | 0000 | xxyyy11 |
| | 011100 | 000000 | rs1 | 000 | rd | 0111111 |

*Table 26. Load/Store Instructions*

| **Format** | 63 44 | | 49 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|
| | **resv** | | **mt** | **ba** | **eew** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
| | **funct6** | **ls** | **rs2** | **rs1** | **funct3** | **ms3/md** | **suffix** |
| mlae*.m | 0.000 | | 01 | ba | eew | 00000 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlbe*.m | 0.000 | | 10 | ba | eew | 00000 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlce*.m | 0.000 | | 00 | ba | eew | 00000 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlre*.m | 0.000 | | 11 | ba | eew | 00000 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlate*.m | 0.000 | | 01 | ba | eew | 00000 | xxyyy11 |
| | 00001 | 0 | rs2 | rs1 | 001 | md | 0111111 |

<table>
<tr><td rowspan="2">mlbte*.m</td><td colspan="2">0.000</td><td>10</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
<tr><td rowspan="2">mlcte*.m</td><td colspan="2">0.000</td><td>00</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
<tr><td rowspan="2">mlrte*.m</td><td colspan="2">0.000</td><td>11</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
<tr><td rowspan="2">msae*.m</td><td colspan="2">0.000</td><td>01</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msbe*.m</td><td colspan="2">0.000</td><td>10</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msce*.m</td><td colspan="2">0.000</td><td>00</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msre*.m</td><td colspan="2">0.000</td><td>11</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msate*.m</td><td colspan="2">0.000</td><td>01</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msbte*.m</td><td colspan="2">0.000</td><td>10</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">mscte*.m</td><td colspan="2">0.000</td><td>00</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">msrte*.m</td><td colspan="2">0.000</td><td>11</td><td>ba</td><td>eew</td><td>00000</td><td>xxyyy11</td></tr>
<tr><td>00001</td><td>1</td><td>rs2</td><td>rs1</td><td>001</td><td>ms3</td><td>0111111</td></tr>
<tr><td rowspan="2">mlae*.v</td><td colspan="2">0.000</td><td>01</td><td>ba</td><td>eew</td><td>00001</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
<tr><td rowspan="2">mlbe*.v</td><td colspan="2">0.000</td><td>10</td><td>ba</td><td>eew</td><td>00001</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
<tr><td rowspan="2">mlce*.v</td><td colspan="2">0.000</td><td>00</td><td>ba</td><td>eew</td><td>00001</td><td>xxyyy11</td></tr>
<tr><td>00000</td><td>0</td><td>rs2</td><td>rs1</td><td>001</td><td>md</td><td>0111111</td></tr>
</table>

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| msae*.v | 0.000 | | 01 | ba | eew | 00001 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |
| msbe*.v | 0.000 | | 10 | ba | eew | 00001 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |
| msce*.v | 0.000 | | 00 | ba | eew | 00001 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |
| mlufae*.m | 0.000 | | 01 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlufbe*.m | 0.000 | | 10 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| mlufce*.m | 0.000 | | 00 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 0 | rs2 | rs1 | 001 | md | 0111111 |
| msfdae*.m | 0.000 | | 01 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |
| msfdbe*.m | 0.000 | | 10 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |
| msfdce*.m | 0.000 | | 00 | ba | eew | 00010 | xxyyy11 |
| | 00000 | 1 | rs2 | rs1 | 001 | ms3 | 0111111 |

*Table 27. Data Move Instructions*

| **Format** | 63 59 | 58 57 | 56 52 | 51 50 | 49 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|---|
| | **mks** | **mkm** | **resv** | **rc** | **mt** | **ba** | **eew** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 20 | 19 15 | | | 14 12 | 11 7 | 6 0 |
| | **funct6** | **di** | **rs2** | **rs1/ms1** | | | **funct3** | **rd/md** | **suffix** |
| mmve*.t.t | mks | mkm | 00000 | 00 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000000 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mmve*.a.a | mks | mkm | 00000 | 00 | 00 | ba | eew | 00001 | xxyyy11 |
| | 000000 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mmve*.a.t | mks | mkm | 00000 | 00 | 00 | ba | eew | 00010 | xxyyy11 |
| | 000000 | 0 | rs2 | ms1 | | | 010 | md | 0111111 |
| mmve*.t.a | mks | mkm | 00000 | 00 | 00 | ba | eew | 00010 | xxyyy11 |
| | 000000 | 1 | rs2 | ms1 | | | 010 | md | 0111111 |
| mmvie*.a.t | mks | mkm | 00000 | 00 | 00 | ba | eew | 00011 | xxyyy11 |
| | 000000 | 0 | imm | ms1 | | | 010 | md | 0111111 |
| mmvie*.t.a | mks | mkm | 00000 | 00 | 00 | ba | eew | 00011 | xxyyy11 |
| | 000000 | 1 | imm | ms1 | | | 010 | md | 0111111 |
| mmve*.x.t | mks | mkm | 00000 | 00 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000001 | 0 | rs2 | ms1 | | | 010 | rd | 0111111 |
| mmve*.t.x | mks | mkm | 00000 | 00 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000001 | 1 | rs2 | rs1 | | | 010 | md | 0111111 |
| mmve*.x.a | mks | mkm | 00000 | 00 | 00 | ba | eew | 00001 | xxyyy11 |
| | 000001 | 0 | rs2 | ms1 | | | 010 | rd | 0111111 |
| mmve*.a.x | mks | mkm | 00000 | 00 | 00 | ba | eew | 00001 | xxyyy11 |
| | 000001 | 1 | rs2 | rs1 | | | 010 | md | 0111111 |
| mfmve*.x.t | mks | mkm | 00000 | 00 | 00 | ba | eew | 00010 | xxyyy11 |
| | 000001 | 0 | rs2 | ms1 | | | 010 | rd | 0111111 |
| mfmve*.t.x | mks | mkm | 00000 | 00 | 00 | ba | eew | 00010 | xxyyy11 |
| | 000001 | 1 | rs2 | rs1 | | | 010 | md | 0111111 |
| mfmve*.x.a | mks | mkm | 00000 | 00 | 00 | ba | eew | 00011 | xxyyy11 |
| | 000001 | 0 | rs2 | ms1 | | | 010 | rd | 0111111 |
| mfmve*.a.x | mks | mkm | 00000 | 00 | 00 | ba | eew | 00011 | xxyyy11 |
| | 000001 | 1 | rs2 | rs1 | | | 010 | md | 0111111 |
| mbcar.m | mks | mkm | 00000 | 01 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbcbr.m | mks | mkm | 00000 | 01 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mbccr.m | mks | mkm | 00000 | 01 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbcace*.m | mks | mkm | 00000 | 10 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbcbce*.m | mks | mkm | 00000 | 10 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbccce*.m | mks | mkm | 00000 | 10 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbcaee*.m | mks | mkm | 00000 | 00 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbcbee*.m | mks | mkm | 00000 | 00 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mbccee*.m | mks | mkm | 00000 | 00 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000010 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mtae*.m | mks | mkm | 00000 | 00 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000011 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mtbe*.m | mks | mkm | 00000 | 00 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000011 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mtce*.m | mks | mkm | 00000 | 00 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000011 | 0 | 00000 | ms1 | | | 010 | md | 0111111 |
| mmvare*.v.m | mks | mkm | 00000 | 01 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvbre*.v.m | mks | mkm | 00000 | 01 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvcre*.v.m | mks | mkm | 00000 | 01 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvare*.m.v | mks | mkm | 00000 | 01 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mmvbre*.m.v | mks | mkm | 00000 | 01 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |
| mmvcre*.m.v | mks | mkm | 00000 | 01 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |
| mmvace*.v.m | mks | mkm | 00000 | 10 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvbce*.v.m | mks | mkm | 00000 | 10 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvcce*.v.m | mks | mkm | 00000 | 10 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 0 | rs2 | ms1 | | | 010 | vd | 0111111 |
| mmvace*.m.v | mks | mkm | 00000 | 10 | 01 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |
| mmvbce*.m.v | mks | mkm | 00000 | 10 | 10 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |
| mmvcce*.m.v | mks | mkm | 00000 | 10 | 00 | ba | eew | 00000 | xxyyy11 |
| | 000100 | 1 | rs2 | vs1 | | | 010 | md | 0111111 |

*Table 28. Matrix Multiplication Instructions*

| **Format** | 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|---|
| | **sps** | **spm** | **typ2** | **typ1** | **typd** | **ba** | **frm** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 20 | | 19 15 | | 14 12 | 11 7 | 6 0 |
| | **funct6** | **fp** | **ms2** | | **ms1** | | **funct3** | **md** | **suffix** |
| mmau.mm | 00000 | 00 | 100 | 100 | 000 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.h.mm | 00000 | 00 | 001 | 001 | 001 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.w.mm | 00000 | 00 | 010 | 010 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.dw.mm | 00000 | 00 | 011 | 011 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| msmau.mm | 00000 | 00 | 100 | 100 | 000 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.h.mm | 00000 | 00 | 001 | 001 | 001 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.w.mm | 00000 | 00 | 010 | 010 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.dw.mm | 00000 | 00 | 011 | 011 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.mm | 00000 | 00 | 100 | 100 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.b.mm | 00000 | 00 | 000 | 000 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.mm | 00000 | 00 | 100 | 100 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.hb.mm | 00000 | 00 | 111 | 111 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.mm | 00000 | 00 | 100 | 100 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.b.mm | 00000 | 00 | 000 | 000 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.mm | 00000 | 00 | 100 | 100 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.hb.mm | 00000 | 00 | 111 | 111 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.mm | 00000 | 00 | 100 | 100 | 000 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.h.mm | 00000 | 00 | 001 | 001 | 001 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mma.w.mm | 00000 | 00 | 010 | 010 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.dw.mm | 00000 | 00 | 011 | 011 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.mm | 00000 | 00 | 100 | 100 | 000 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.h.mm | 00000 | 00 | 001 | 001 | 001 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.w.mm | 00000 | 00 | 010 | 010 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.dw.mm | 00000 | 00 | 011 | 011 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.mm | 00000 | 00 | 100 | 100 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.b.mm | 00000 | 00 | 000 | 000 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.mm | 00000 | 00 | 100 | 100 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.hb.mm | 00000 | 00 | 111 | 111 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.mm | 00000 | 00 | 100 | 100 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.b.mm | 00000 | 00 | 000 | 000 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msoma.mm | 00000 | 00 | 100 | 100 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msoma.hb.mm | 00000 | 00 | 111 | 111 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfma.mm | 00000 | 00 | 100 | 100 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.hf.mm | 00000 | 00 | 001 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.f.mm | 00000 | 00 | 010 | 010 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.d.mm | 00000 | 00 | 011 | 011 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.mm | 00000 | 00 | 100 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.cf.mm | 00000 | 00 | 000 | 000 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.hf.mm | 00000 | 00 | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.f.mm | 00000 | 00 | 010 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.mm | 00000 | 00 | 100 | 100 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.cf.mm | 00000 | 00 | 000 | 000 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |

*Table 29. Sparsing Matrix Multiplication Instructions*

| **Format** | 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|---|
| | **sps** | **spm** | **typ2** | **typ1** | **typd** | **ba** | **frm** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 20 | | 19 15 | | 14 12 | 11 7 | 6 0 |
| | **funct6** | **fp** | **ms2** | | **ms1** | | **funct3** | **md** | **suffix** |
| mmau.spa.mm | sps | 01 | 100 | 100 | 000 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.spa.h.mm | sps | 01 | 001 | 001 | 001 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mmau.spa.w.mm | sps | 01 | 010 | 010 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.spa.dw.mm | sps | 01 | 011 | 011 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spa.mm | sps | 01 | 100 | 100 | 000 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spa.h.mm | sps | 01 | 001 | 001 | 001 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spa.w.mm | sps | 01 | 010 | 010 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spa.dw.mm | sps | 01 | 011 | 011 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.spa.mm | sps | 01 | 100 | 100 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.spa.b.mm | sps | 01 | 000 | 000 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.spa.mm | sps | 01 | 100 | 100 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.spa.hb.mm | sps | 01 | 111 | 111 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.spa.mm | sps | 01 | 100 | 100 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.spa.b.mm | sps | 01 | 000 | 000 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.spa.mm | sps | 01 | 100 | 100 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.spa.hb.mm | sps | 01 | 111 | 111 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mma.spa.mm | sps | 01 | 100 | 100 | 000 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spa.h.mm | sps | 01 | 001 | 001 | 001 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spa.w.mm | sps | 01 | 010 | 010 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spa.dw.mm | sps | 01 | 011 | 011 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spa.mm | sps | 01 | 100 | 100 | 000 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spa.h.mm | sps | 01 | 001 | 001 | 001 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spa.w.mm | sps | 01 | 010 | 010 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spa.dw.mm | sps | 01 | 011 | 011 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.spa.mm | sps | 01 | 100 | 100 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.spa.b.mm | sps | 01 | 000 | 000 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.spa.mm | sps | 01 | 100 | 100 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.spa.hb.mm | sps | 01 | 111 | 111 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.spa.mm | sps | 01 | 100 | 100 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.spa.b.mm | sps | 01 | 000 | 000 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| msoma.spa.mm | sps | 01 | 100 | 100 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msoma.spa.hb.mm | sps | 01 | 111 | 111 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spa.mm | sps | 01 | 100 | 100 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spa.hf.mm | sps | 01 | 001 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spa.f.mm | sps | 01 | 010 | 010 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spa.d.mm | sps | 01 | 011 | 011 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spa.mm | sps | 01 | 100 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spa.cf.mm | sps | 01 | 000 | 000 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spa.hf.mm | sps | 01 | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spa.f.mm | sps | 00 | 010 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.spa.mm | sps | 01 | 100 | 100 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.spa.cf.mm | sps | 01 | 000 | 000 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.spb.mm | sps | 10 | 100 | 100 | 000 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.spb.h.mm | sps | 10 | 001 | 001 | 001 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mmau.spb.w.mm | sps | 10 | 010 | 010 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mmau.spb.dw.mm | sps | 10 | 011 | 011 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spb.mm | sps | 10 | 100 | 100 | 000 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spb.h.mm | sps | 10 | 001 | 001 | 001 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spb.w.mm | sps | 10 | 010 | 010 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msmau.spb.dw.mm | sps | 10 | 011 | 011 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.spb.mm | sps | 10 | 100 | 100 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqmau.spb.b.mm | sps | 10 | 000 | 000 | 010 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.spb.mm | sps | 10 | 100 | 100 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| momau.spb.hb.mm | sps | 10 | 111 | 111 | 011 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.spb.mm | sps | 10 | 100 | 100 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqmau.spb.b.mm | sps | 10 | 000 | 000 | 010 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.spb.mm | sps | 10 | 100 | 100 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msomau.spb.hb.mm | sps | 10 | 111 | 111 | 011 | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mma.spb.mm | sps | 10 | 100 | 100 | 000 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spb.h.mm | sps | 10 | 001 | 001 | 001 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spb.w.mm | sps | 10 | 010 | 010 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mma.spb.dw.mm | sps | 10 | 011 | 011 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spb.mm | sps | 10 | 100 | 100 | 000 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spb.h.mm | sps | 10 | 001 | 001 | 001 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spb.w.mm | sps | 10 | 010 | 010 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msma.spb.dw.mm | sps | 10 | 011 | 011 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.spb.mm | sps | 10 | 100 | 100 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mqma.spb.b.mm | sps | 10 | 000 | 000 | 010 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.spb.mm | sps | 10 | 100 | 100 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| moma.spb.hb.mm | sps | 10 | 111 | 111 | 011 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.spb.mm | sps | 10 | 100 | 100 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msqma.spb.b.mm | sps | 10 | 000 | 000 | 010 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| msoma.spb.mm | sps | 10 | 100 | 100 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| msoma.spb.hb.mm | sps | 10 | 111 | 111 | 011 | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spb.mm | sps | 10 | 100 | 100 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spb.hf.mm | sps | 10 | 001 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spb.f.mm | sps | 10 | 010 | 010 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfma.spb.d.mm | sps | 10 | 011 | 011 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spb.mm | sps | 10 | 100 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spb.cf.mm | sps | 10 | 000 | 000 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spb.hf.mm | sps | 10 | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfwma.spb.f.mm | sps | 00 | 010 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.spb.mm | sps | 10 | 100 | 100 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |
| mfqma.spb.cf.mm | sps | 10 | 000 | 000 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 100 | md | 0111111 |

*Table 30. Element-wise Arithmetic & Logic Instructions*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Format** | 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
| | **mks** | **mkm** | **typ2** | **typ1** | **typd** | **ba** | **frm** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 20 | | 19 15 | | 14 12 | 11 7 | 6 0 |
| | **funct6** | **fp** | **ms2** | | **ms1** | | **funct3** | **md** | **suffix** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| maddu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msaddu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwaddu.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00000 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| madd.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msadd.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwadd.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00001 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msubu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00010 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mssubu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10010 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwsubu.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00010 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msub.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00011 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mssub.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10011 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwsub.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00011 | xxyyy11 |
| | 000000 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mminu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00000 | xxyyy11 |
| | 000001 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmin.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00001 | xxyyy11 |
| | 000001 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mmaxu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00010 | xxyyy11 |
| | 000001 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmax.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00011 | xxyyy11 |
| | 000001 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mand.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00000 | xxyyy11 |
| | 000010 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mor.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00001 | xxyyy11 |
| | 000010 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mxor.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00010 | xxyyy11 |
| | 000010 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msll.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00000 | xxyyy11 |
| | 000011 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msrl.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00001 | xxyyy11 |
| | 000011 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msra.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00010 | xxyyy11 |
| | 000011 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmul.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00000 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmulh.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00001 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmulhu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00010 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mmulhsu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 00011 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msmulu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10000 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| msmul.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10001 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| msmulsu.*.mm | mks | mkm | eew | eew | eew | ba | 000 | 10011 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwmulu.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00000 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwmul.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00001 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mwmulsu.*.mm | mks | mkm | eew | eew | +1 | ba | 000 | 00011 | xxyyy11 |
| | 000100 | 0 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfadd.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfwadd.*.mm | mks | mkm | eew | eew | +1 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfsub.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfwsub.*.mm | mks | mkm | eew | eew | +1 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfmin.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00000 | xxyyy11 |
| | 000001 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfmax.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00010 | xxyyy11 |
| | 000001 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfmul.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00000 | xxyyy11 |
| | 000100 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfwmul.*.mm | mks | mkm | eew | eew | +1 | ba | frm | 00000 | xxyyy11 |
| | 000100 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfdiv.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00000 | xxyyy11 |
| | 000101 | 1 | ms2 | | ms1 | | 101 | md | 0111111 |
| mfsqrt.*.mm | mks | mkm | eew | eew | eew | ba | frm | 00000 | xxyyy11 |
| | 000110 | 1 | 00000 | | ms1 | | 101 | md | 0111111 |

*Table 31. Type Convert Instructions*

| Format | 63 59 | 58 57 | 56 54 | 53 51 | 50 48 | 47 | 46 44 | 43 39 | 38 32 |
|---|---|---|---|---|---|---|---|---|---|
| | **mks** | **mkm** | **enw** | **typ1** | **typd** | **ba** | **frm** | **funct5** | **opcode** |
| | 31 26 | 25 | 24 | 23 20 | 19 15 | | 14 12 | 11 7 | 6 0 |
| | **funct6** | **fp** | **fs** | **f4** | **ms1** | | **funct3** | **md** | **suffix** |
| mfcvt.bf.hf.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.hf.bf.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.fw.f.m | mks | mkm | 001 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.hf.cf.m | mks | mkm | 001 | 000 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.f.hf.m | mks | mkm | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.d.f.m | mks | mkm | 001 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.f.fw.m | mks | mkm | 111 | 100 | 111 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.cf.hf.m | mks | mkm | 111 | 001 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.hf.f.m | mks | mkm | 111 | 010 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.f.d.m | mks | mkm | 111 | 011 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.f.x.m | mks | mkm | 000 | 100 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.hf.h.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfcvtu.f.w.m | mks | mkm | 000 | 010 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.d.dw.m | mks | mkm | 000 | 011 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.f.x.m | mks | mkm | 000 | 100 | 000 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.hf.h.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.f.w.m | mks | mkm | 000 | 010 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.d.dw.m | mks | mkm | 000 | 011 | 011 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.fw.x.m | mks | mkm | 001 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.fq.x.m | mks | mkm | 010 | 100 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.fo.x.m | mks | mkm | 011 | 100 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.hf.hb.m | mks | mkm | 010 | 111 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.f.hb.m | mks | mkm | 011 | 111 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.hf.b.m | mks | mkm | 001 | 000 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.f.b.m | mks | mkm | 010 | 000 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.f.h.m | mks | mkm | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfwcvtu.d.w.m | mks | mkm | 001 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.fw.x.m | mks | mkm | 001 | 100 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.fq.x.m | mks | mkm | 010 | 100 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.fo.x.m | mks | mkm | 011 | 100 | 011 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.hf.hb.m | mks | mkm | 010 | 111 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.f.hb.m | mks | mkm | 011 | 111 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.hf.b.m | mks | mkm | 001 | 000 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.f.b.m | mks | mkm | 010 | 000 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.f.h.m | mks | mkm | 001 | 001 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.d.w.m | mks | mkm | 001 | 010 | 011 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.f.xw.m | mks | mkm | 111 | 100 | 111 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.hf.w.m | mks | mkm | 111 | 010 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.f.dw.m | mks | mkm | 111 | 011 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.f.xw.m | mks | mkm | 111 | 100 | 111 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfncvt.hf.w.m | mks | mkm | 111 | 010 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.f.dw.m | mks | mkm | 111 | 011 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 1 | 0 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.x.f.m | mks | mkm | 000 | 100 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.h.hf.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.w.f.m | mks | mkm | 000 | 010 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvtu.dw.d.m | mks | mkm | 000 | 011 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.x.f.m | mks | mkm | 000 | 100 | 000 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.h.hf.m | mks | mkm | 000 | 001 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.w.f.m | mks | mkm | 000 | 010 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfcvt.dw.d.m | mks | mkm | 000 | 011 | 011 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.xw.f.m | mks | mkm | 001 | 100 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.w.hf.m | mks | mkm | 001 | 001 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvtu.dw.f.m | mks | mkm | 001 | 010 | 011 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.xw.f.m | mks | mkm | 001 | 100 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfwcvt.w.hf.m | mks | mkm | 001 | 001 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfwcvt.dw.f.m | mks | mkm | 001 | 010 | 011 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.x.fw.m | mks | mkm | 111 | 100 | 111 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.x.fq.m | mks | mkm | 110 | 100 | 110 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.x.fo.m | mks | mkm | 101 | 100 | 101 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.hb.hf.m | mks | mkm | 110 | 001 | 111 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.hb.f.m | mks | mkm | 101 | 010 | 111 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.b.hf.m | mks | mkm | 111 | 001 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.b.f.m | mks | mkm | 110 | 010 | 000 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.h.f.m | mks | mkm | 111 | 010 | 001 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvtu.w.d.m | mks | mkm | 111 | 011 | 010 | ba | frm | 00000 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.x.fw.m | mks | mkm | 111 | 100 | 111 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.x.fq.m | mks | mkm | 110 | 100 | 110 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.x.fo.m | mks | mkm | 101 | 100 | 101 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mfncvt.hb.hf.m | mks | mkm | 110 | 001 | 111 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.hb.f.m | mks | mkm | 101 | 010 | 111 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.b.hf.m | mks | mkm | 111 | 001 | 000 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.b.f.m | mks | mkm | 110 | 010 | 000 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.h.f.m | mks | mkm | 111 | 010 | 001 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |
| mfncvt.w.d.m | mks | mkm | 111 | 011 | 010 | ba | frm | 00001 | xxyyy11 |
| | 000000 | 0 | 1 | 0000 | ms1 | | 111 | md | 0111111 |