Integrated Matrix Extension Option A (Common-Type Variant)

José E. Moreira (IBM)

February 2024

1 Introduction

This is a strawman for Option A (one matrix tile per vector register) of the Integrated Matrix Extension (IME). As such, it is meant to guide and facilitate discussion by providing a concrete draft specification. There should be no expectation that any content in the strawman will make into the final specification, particularly in the early stages of the work. As we evolve our work in the IME Task Group, refined concepts from this strawman may be promoted to an actual draft specification.

This document focuses on the *common-type* variant of the extensions. That is, all matrix data types involved in one instruction are the same. This is distinct from a *mixed-type* variant that supports matrices of different data types in one instruction.

Notation: Matrices are represented by bold-face capital letters ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$). The element at row i, column j of matrix $\mathbf{A}$ can be denoted as either $\mathbf{A}(i,j)$ or a_{ij}. $\mathbf{A}(i,:)$ represents the i-th row of matrix $\mathbf{A}$ and $\mathbf{A}(:,j)$ its j-th column. $\mathbf{A}(i:h, j:w)$ denotes the two-dimensional section of matrix $\mathbf{A}$ consisting of the h rows beginning in row i and the w columns beginning in column j.

2 Matrix tiles

A *matrix tile* is a rectangular section of a matrix. Matrix tiles are defined by the scalar data type of their elements and their *shape* $\langle \lambda \times \kappa \rangle$, where λ is the number of rows of the tile and κ is the number of columns. The shape of the tile stored in a vector register depends on the length of the vector register (vlen) and the scalar data type.

Table 1 shows the tile shapes for all valid values of vlen and three different scalar data types: double-precision IEEE floating-point (fp64), single-precision IEEE floating-point (fp32), and the 16-bit bfloat16 format (bf16). For a given vector length vlen and scalar data type width sew, we can compute the shape of the matrix tile as follows: First, we compute the vector length in elements: $\mathsf{vlene} = \frac{\mathsf{vlen}}{\mathsf{sew}}$. Then, we compute $\kappa = 2^{\lfloor \log_2(\mathsf{vlene})/2 \rfloor}$. Finally, we compute $\lambda = \frac{\mathsf{vlene}}{\kappa}$. This procedure results in either $\lambda = \kappa$ or $\lambda = 2\kappa$. We note that a pair of vector registers can always be used to store a $\lambda \times \lambda$ section of a matrix, with the second register empty when $\kappa = \lambda$

3 Matrix instructions

All matrix instructions in the common-type variant operate on matrix tiles of a common scalar data type. Each architected vector register stores one matrix tile, and instructions take as arguments either one, two, or three vector register identifiers. Additional arguments to matrix instructions can include *index registers*, containing unsigned integer values, and/or vector masks.

	fp64		fp32		bf16	
vlen	vlene	$\lambda \times \kappa$	vlene	$\lambda \times \kappa$	vlene	$\lambda \times \kappa$
64	1	1×1	2	2×1	4	2×2
128	2	2×1	4	2×2	8	4×2
256	4	2×2	8	4×2	16	4×4
512	8	4×2	16	4×4	32	8×4
1024	16	4×4	32	8×4	64	8×8
2048	32	8×4	64	8×8	128	16×8
4096	64	8×8	128	16×8	256	16×16
8192	128	16×8	256	16×16	512	32×16
16384	256	16×16	512	32×16	1024	32×32
32768	512	32×16	1024	32×32	2048	64×32

Table 1: Matrix tile shapes for different combinations of vector length (vlen) and three scalar data types: fp64, fp32, and bf16. For each data type, we show the vector length in elements (vlene) and the shape of matrix tile in one vector register ($\kappa \times \lambda$).

3.1 Matrix computation instructions

Matrix computation instructions have the general form

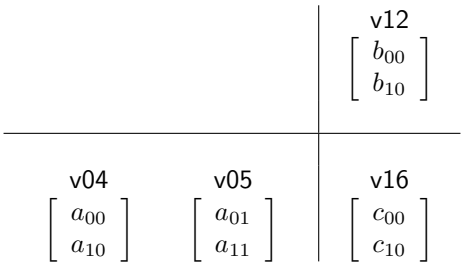
$$\mathsf{m}\{\mathsf{comp}\}\langle T, \otimes, \oplus\rangle(\mathbf{A}, \mathbf{B}, \mathbf{C}, \ldots)$$

where m identifies this as a matrix instruction and comp identifies the specific computation, usually following the nomenclature established by BLAS. T is the scalar data type of the computation, and the multiplication/addition pair $(\otimes, \oplus)$ forms a semiring on type T. (The precision of the operations may be different than the precision of the data type. For example, addition and multiplication for data type bf16 can be performed in IEEE single-precision.) **A**, **B**, and **C** are vector register identifiers that are used to represent the **A**, **B**, and **C** matrices in the operation, respectively. **A** and **B** are input matrices. **C** is the result matrix of the computation. **A** identifies a pair of vector registers, whereas **B** and **C** identify single registers. When $\lambda = 2\kappa$, both **A** registers are used to store a $\lambda \times \lambda$ section of **A**. When $\lambda = \kappa$, we only use the first register in the pair. Additional arguments can be present in different instructions.

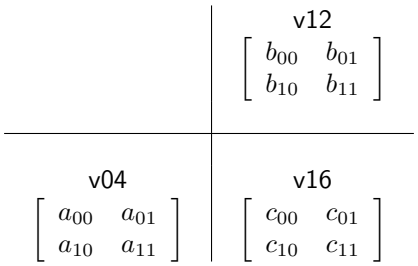
As an example, let $T =$ fp32 and let vlen $= 64$ ($\lambda = 2\kappa$). An instruction that performs a BLAS single-precision rank-1 update (sger) on a matrix **C** using input matrices **A** and **B** would look something like (the instruction is specified in more detail below)

$$\mathsf{mger}\langle \mathsf{fp32}, \times, +\rangle(\mathsf{v04}, \mathsf{v12}, \mathsf{v16}, \ldots)$$

where the vector register pair (v4, v5) is used to store two tiles of matrix **A**, v12 is used to store one tile of matrix **B**, and v16 is used to store one tile of matrix **C**. Geometrically, the computation looks like this:



If we now set vlen = 128 ($\lambda = \kappa$), the computation from the same instruction would look like this:



3.1.1 mger – rank-1 update

Syntax: mger$\langle T, \otimes, \oplus \rangle(\mathbf{A}, \mathbf{B}, \mathbf{C}, j, i)$

Arguments: $\mathbf{A}$ is a $\lambda \times \lambda$ section of an input matrix, contained in two vector registers (one of which may be superfluous); $\mathbf{B}$ is a $\lambda \times \kappa$ tile of another input matrix, contained in one vector register; $\mathbf{C}$ is a $\lambda \times \kappa$ tile of the output matrix, contained in another vector register; i identifies one row of $\mathbf{B}$, provided in an index register; j identifies one column of $\mathbf{A}$, provided in an index register; $i, j \in [0, \kappa)$.

Semantics: This instruction computes $\mathbf{C} \leftarrow \mathbf{C} \oplus \mathbf{A}(:, j) \otimes \mathbf{B}(i, :)$.

3.1.2 mgemm – matrix multiplication

Syntax: mgemm$\langle T, \otimes, \oplus \rangle(\mathbf{A}, \mathbf{B}, \mathbf{C}, K)$

Arguments: $\mathbf{A}$ is a $\lambda \times \lambda$ section of an input matrix, contained in two vector registers (one of which may be superfluous); $\mathbf{B}$ is a $\lambda \times \kappa$ tile of another input matrix, contained in one vector register; $\mathbf{C}$ is a $\lambda \times \kappa$ tile of the output matrix, contained in another vector register; K, provided in an index registers, specifies the *depth* of the computation.

Semantics: This instruction computes $\mathbf{for}(k \leftarrow 0; k < K; k \leftarrow k + 1)\mathbf{C} \leftarrow \mathbf{C} \oplus \mathbf{A}(:, k) \otimes \mathbf{B}(k, :)$.

4 Case study: sgemm

The BLAS sgemm routine computes $\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}$, where $\mathbf{C}$ is a $M \times N$ matrix, $\mathbf{A}$ is a $M \times K$ matrix, $\mathbf{B}$ is a $K \times N$ matrix, α and β are scalars. All data types are single-precision floating-point numbers. (We are ignoring the matrix transpose variants.)

Pseudo code for a vlen agnostic implementation of sgemm is shown in Figure 1. It consists of a double-nested parallel loop across the rows and columns of result matrix $\mathbf{C}$, Each iteration of the loop nest computes an $m \times n$ panel of $\mathbf{C}$ in the *micro-kernel* μsgemm, with $m = 4\lambda$ and $n = 4\kappa$. (For simplicity, we let M and N be integer multiples of m and n, respectively, so there are no loop remainders.)

Pseudo code for the micro-kernel in shown in Figure 2. It corresponds to the mapping of matrices to vector registers shown in Figure 3 (for vlem = 128) and Figure 4 (for vlen = 64). The assignments are the same, but for $\lambda = \kappa$ four of the vector registers assigned to $\mathbf{A}$ are not used. Vector registers v04–v11 store a $m \times \lambda$ panel of $\mathbf{A}$, vector registers v12–v15 store a $\lambda \times n$ panel of $\mathbf{B}$, and vector registers v16–v31 store a $m \times n$ panel of $\mathbf{C}$. The code, including the corresponding binary code, produces the correct result independent of the value of vlen. We call this code vlen agnostic.

The micro-kernel works as follows. We first zero the contents of the $m \times n$ panel of $\mathbf{C}$ in registers v16–v31, using standard RISC-V vector instructions. We then iterate over the inner dimension, with each iteration processing λ columns of $\mathbf{A}$ and λ rows of $\mathbf{B}$. We load the panel of $\mathbf{A}$ in registers v04–V11. This can be

procedure sgemm$(M, N, K, \alpha, \mathbf{A}, \mathbf{B}, \beta, \mathbf{C})$
    $m \leftarrow 4\lambda$
    $n \leftarrow 4\kappa$
    foreach$(I \leftarrow 0; I < M; I \leftarrow I + m)$
        foreach$(J \leftarrow 0; J < N; J \leftarrow J + n)$
            μsgemm$(K, \alpha, \mathbf{A}(I : m. :), \mathbf{B}(:, J : n), \beta, \mathbf{C}(I : m, J : n)$
        end
    end
end

Figure 1: The (M, N, K) BLAS routine sgemm can be implemented as a two-dimensional parallel loop, each iteration computing a $m \times n$ panel of the result matrix $\mathbf{C}$. The computation of each panel is performed by a micro-kernel that takes as input a block of m rows of $\mathbf{A}$ and a block of n columns of $\mathbf{B}$.

accomplished either with existing RISC-V indexed vector loads or with a new two-dimensional strided vector load. Correspondingly, we load the panel of $\mathbf{B}$ in registers v12–v15. We then perform 16 mgemm instructions, each updating a tile from the panel of $\mathbf{C}$. At the end of the loop we will have computed $\mathbf{AB}$. The scaling by scalar α can be accomplished with existing RISC-V vector instructions, as can the additional of $\beta\mathbf{C}$. The loading and storing of the panel of $\mathbf{C}$ can again use either indexed vector load instructions or a new two-dimensional strided vector load.

The binary code in Figure 2 is agnostic to the value of vlen, producing the correct result in every case. The computational intensity (η), defined as the ratio of floating-point operations by the number of elements transferred, can be computed as follows. Each iteration of the micro-kernel loads $4\lambda^2$ elements of $\mathbf{A}$ and $4\lambda\kappa$ elements of $\mathbf{B}$. Each mgemm instruction performs $\lambda^2\kappa$ floating-point multiply-add operations. Therefore, each iteration of the micro-kernel performs $32\lambda^2\kappa$ floating-point operations. Therefore, the computational intensity is

$$\eta = \frac{32\lambda^2\kappa}{4\lambda(\lambda+\kappa)} = 8\frac{\lambda\kappa}{\lambda+\kappa} = \begin{cases} 4\lambda & (\lambda = \kappa) \\ \frac{8}{3}\lambda & (\lambda = 2\kappa) \end{cases} .$$

The computational intensity scales with λ, which is proportional to the square root of vlen. Furthermore, the computational intensity is highest when the registers can hold a square tile.

procedure μsgemm$(K, \alpha, \mathbf{A}, \mathbf{B}, \beta, \mathbf{C})$

$$\begin{bmatrix} \mathsf{v16} & \mathsf{v17} & \mathsf{v18} & \mathsf{v19} \\ \mathsf{v20} & \mathsf{v21} & \mathsf{v22} & \mathsf{v23} \\ \mathsf{v24} & \mathsf{v25} & \mathsf{v26} & \mathsf{v27} \\ \mathsf{v28} & \mathsf{v29} & \mathsf{v30} & \mathsf{v31} \end{bmatrix} \leftarrow 0$$

for$(k \leftarrow 0; k < K; k \leftarrow k + \lambda)$

$(\mathsf{v04}, \mathsf{v05}) \leftarrow \mathbf{A}(0 \times \lambda : \lambda, k : \lambda)$

$(\mathsf{v06}, \mathsf{v07}) \leftarrow \mathbf{A}(1 \times \lambda : \lambda, k : \lambda)$

$(\mathsf{v08}, \mathsf{v09}) \leftarrow \mathbf{A}(2 \times \lambda : \lambda, k : \lambda)$

$(\mathsf{v10}, \mathsf{v11}) \leftarrow \mathbf{A}(3 \times \lambda : \lambda, k : \lambda)$

$\mathsf{v12} \leftarrow \mathbf{B}(k : \lambda, 0 \times \kappa : \kappa)$

$\mathsf{v13} \leftarrow \mathbf{B}(k : \lambda, 1 \times \kappa : \kappa)$

$\mathsf{v14} \leftarrow \mathbf{B}(k : \lambda, 2 \times \kappa : \kappa)$

$\mathsf{v15} \leftarrow \mathbf{B}(k : \lambda, 3 \times \kappa : \kappa)$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v04}, \mathsf{v05}), \mathsf{v12}, \mathsf{v16}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v04}, \mathsf{v05}), \mathsf{v13}, \mathsf{v17}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v04}, \mathsf{v05}), \mathsf{v14}, \mathsf{v18}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v04}, \mathsf{v05}), \mathsf{v15}, \mathsf{v19}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v06}, \mathsf{v07}), \mathsf{v12}, \mathsf{v20}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v06}, \mathsf{v07}), \mathsf{v13}, \mathsf{v21}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v06}, \mathsf{v07}), \mathsf{v14}, \mathsf{v22}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v06}, \mathsf{v07}), \mathsf{v15}, \mathsf{v23}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v08}, \mathsf{v09}), \mathsf{v12}, \mathsf{v24}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v08}, \mathsf{v09}), \mathsf{v13}, \mathsf{v25}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v08}, \mathsf{v09}), \mathsf{v14}, \mathsf{v26}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v08}, \mathsf{v09}), \mathsf{v15}, \mathsf{v27}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v10}, \mathsf{v11}), \mathsf{v12}, \mathsf{v28}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v10}, \mathsf{v11}), \mathsf{v13}, \mathsf{v29}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v10}, \mathsf{v11}), \mathsf{v14}, \mathsf{v30}, \max(\lambda, K - k))$

$\mathsf{mgemm}\langle \mathsf{fp32}, \times, + \rangle((\mathsf{v10}, \mathsf{v11}), \mathsf{v15}, \mathsf{v31}, \max(\lambda, K - k))$

end

$(\mathsf{v16}, \mathsf{v17}, \ldots, \mathsf{v31}) \leftarrow \alpha \times (\mathsf{v16}, \mathsf{v17}, \ldots, \mathsf{v31})$

$$\mathbf{C} \leftarrow \begin{bmatrix} \mathsf{v16} & \mathsf{v17} & \mathsf{v18} & \mathsf{v19} \\ \mathsf{v20} & \mathsf{v21} & \mathsf{v22} & \mathsf{v23} \\ \mathsf{v24} & \mathsf{v25} & \mathsf{v26} & \mathsf{v27} \\ \mathsf{v28} & \mathsf{v29} & \mathsf{v30} & \mathsf{v31} \end{bmatrix} + \beta \times \mathbf{C}$$

end

Figure 2: The (M, N, K) BLAS routine sgemm can be implemented as a two-dimensional parallel loop, each iteration computing a $m \times n$ panel of the result matrix $\mathbf{C}$. The computation of each panel is performed by a micro-kernel that takes as input a block of m rows of $\mathbf{A}$ and a block of n columns of $\mathbf{B}$.

	v12	v13	v14	v15
	$\begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}$	$\begin{bmatrix} b_{02} & b_{03} \\ b_{12} & b_{13} \end{bmatrix}$	$\begin{bmatrix} b_{04} & b_{05} \\ b_{14} & b_{15} \end{bmatrix}$	$\begin{bmatrix} b_{06} & b_{07} \\ b_{16} & b_{17} \end{bmatrix}$
v04 $\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}$	v16 $\begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$	v17 $\begin{bmatrix} c_{02} & c_{03} \\ c_{12} & c_{13} \end{bmatrix}$	v18 $\begin{bmatrix} c_{04} & c_{05} \\ c_{14} & c_{15} \end{bmatrix}$	v19 $\begin{bmatrix} c_{06} & c_{07} \\ c_{16} & c_{17} \end{bmatrix}$
v06 $\begin{bmatrix} a_{20} & a_{21} \\ a_{30} & a_{31} \end{bmatrix}$	v20 $\begin{bmatrix} c_{20} & c_{21} \\ c_{30} & c_{31} \end{bmatrix}$	v21 $\begin{bmatrix} c_{22} & c_{23} \\ c_{32} & c_{33} \end{bmatrix}$	v22 $\begin{bmatrix} c_{24} & c_{25} \\ c_{34} & c_{35} \end{bmatrix}$	v23 $\begin{bmatrix} c_{26} & c_{27} \\ c_{36} & c_{37} \end{bmatrix}$
v08 $\begin{bmatrix} a_{40} & a_{41} \\ a_{50} & a_{51} \end{bmatrix}$	v24 $\begin{bmatrix} c_{40} & c_{41} \\ c_{50} & c_{51} \end{bmatrix}$	v25 $\begin{bmatrix} c_{42} & c_{43} \\ c_{52} & c_{53} \end{bmatrix}$	v26 $\begin{bmatrix} c_{44} & c_{45} \\ c_{54} & c_{55} \end{bmatrix}$	v27 $\begin{bmatrix} c_{46} & c_{47} \\ c_{56} & c_{57} \end{bmatrix}$
v10 $\begin{bmatrix} a_{60} & a_{61} \\ a_{70} & a_{71} \end{bmatrix}$	v28 $\begin{bmatrix} c_{60} & c_{61} \\ c_{70} & c_{71} \end{bmatrix}$	v29 $\begin{bmatrix} c_{62} & c_{63} \\ c_{72} & c_{73} \end{bmatrix}$	v30 $\begin{bmatrix} c_{64} & c_{65} \\ c_{74} & c_{75} \end{bmatrix}$	v31 $\begin{bmatrix} c_{66} & c_{67} \\ c_{76} & c_{77} \end{bmatrix}$

Figure 3: When vlen $= 128$, each iteration of the micro-kernel updates an 8×8 panel of $\mathbf{C}$ with the product of an 8×2 panel of $\mathbf{A}$ by a 2×8 panel of $\mathbf{B}$. There are 8 vectors assigned to $\mathbf{A}$ (v04–v11), 4 vectors assigned to $\mathbf{B}$ (v12–v15), and 16 registers assigned to $\mathbf{C}$ (v16–v31).

		v12 $\begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}$	v13 $\begin{bmatrix} b_{01} \\ b_{11} \end{bmatrix}$	v14 $\begin{bmatrix} b_{02} \\ b_{12} \end{bmatrix}$	v15 $\begin{bmatrix} b_{03} \\ b_{13} \end{bmatrix}$
v04 $\begin{bmatrix} a_{00} \\ a_{10} \end{bmatrix}$	v05 $\begin{bmatrix} a_{01} \\ a_{11} \end{bmatrix}$	v16 $\begin{bmatrix} c_{00} \\ c_{10} \end{bmatrix}$	v17 $\begin{bmatrix} c_{01} \\ c_{11} \end{bmatrix}$	v18 $\begin{bmatrix} c_{02} \\ c_{12} \end{bmatrix}$	v19 $\begin{bmatrix} c_{03} \\ c_{13} \end{bmatrix}$
v06 $\begin{bmatrix} a_{20} \\ a_{30} \end{bmatrix}$	v07 $\begin{bmatrix} a_{21} \\ a_{31} \end{bmatrix}$	v20 $\begin{bmatrix} c_{20} \\ c_{30} \end{bmatrix}$	v21 $\begin{bmatrix} c_{21} \\ c_{31} \end{bmatrix}$	v22 $\begin{bmatrix} c_{22} \\ c_{32} \end{bmatrix}$	v23 $\begin{bmatrix} c_{23} \\ c_{33} \end{bmatrix}$
v08 $\begin{bmatrix} a_{40} \\ a_{50} \end{bmatrix}$	v09 $\begin{bmatrix} a_{41} \\ a_{51} \end{bmatrix}$	v24 $\begin{bmatrix} c_{40} \\ c_{50} \end{bmatrix}$	v25 $\begin{bmatrix} c_{41} \\ c_{51} \end{bmatrix}$	v26 $\begin{bmatrix} c_{42} \\ c_{52} \end{bmatrix}$	v27 $\begin{bmatrix} c_{43} \\ c_{53} \end{bmatrix}$
v10 $\begin{bmatrix} a_{60} \\ a_{70} \end{bmatrix}$	v11 $\begin{bmatrix} a_{61} \\ a_{71} \end{bmatrix}$	v28 $\begin{bmatrix} c_{60} \\ c_{70} \end{bmatrix}$	v29 $\begin{bmatrix} c_{61} \\ c_{71} \end{bmatrix}$	v30 $\begin{bmatrix} c_{62} \\ c_{72} \end{bmatrix}$	v31 $\begin{bmatrix} c_{63} \\ c_{73} \end{bmatrix}$

Figure 4: When vlen $= 64$, each iteration of the micro-kernel updates an 8×4 panel of $\mathbf{C}$ with the product of an 8×2 panel of $\mathbf{A}$ by a 2×4 panel of $\mathbf{B}$. Vector register assignments for matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are the same as with vlen $= 128$, but in this case v05, v07, v09, and v11 are actually used.