智能计算芯片导论期末课程设计报告
——基于 RISCV 向量扩展指令集的 VPU 设计

2025 年 6 月 19 日

22307130445 贾梓越

1 设计目标

使用 Verilog 搭建一个基于 RISCV 向量扩展指令集的 VPU。使用汇编语言实现 8*8 的矩阵乘加运算，使用 Vivado 进行仿真验证，并进行性能分析，测试延迟，仿真功耗以及 LUT/DSP/BRAM 的占用率。最后尝试对架构和软件进行优化，并分析优化的效果。

2 设计架构

2.1 整体架构

整体架构如图 1 所示，VPU 一共分为四级流水线，每个指令的执行需要经过四个周期：取指，译码并取数，执行或访存，最后写回。

指令缓存用于存储需要执行的指令，由一个指令计数器 PC 作为读取地址逐个读取，每个指令的位宽为 32 位，与 RISCV 的指令集兼容。

指令读出后经过译码器，将指令根据指令类型拆分出为立即数、标量和矢量的源寄存器地址和目标寄存器地址，送往立即数生成模块、标量寄存器和矢量寄存器。标量寄存器和矢量寄存器用于存储指令执行过程中的中间结果。设计的数据位宽为 32，向量寄存器的向量长度为 8，因此每个单元位宽为 32*8=256 位。向量寄存器和标量寄存器各 32 个，对应 5 位寄存器地址，这也与 RISCV 指令集的寄存器地址位宽相同。由于立即数需要与标量寄存器的读取结果相加，或者直接写入标量寄存器中，因此在译码时将立即数的位宽拓展为 32 位，便于和一般数据相加。

标量寄存器读取的数据有三种通路：（1）和立即数产生单元的输出数据相加后作为访存的地址访问标量缓存或者矢量缓存；（2）直接输入乘加运算单元进行计算；（3）写入标量缓存

矢量寄存器读取的数据则有两种通路：（1）直接输入乘加运算单元进行计算；（2）写入矢量缓存。

除了基本的存储和计算单元，还需要一个整体的控制单元负责控制各个存储器的读写使能信号和寄存器的写使能信号，以及一些数据选择器的选择信号。

2.2 指令集设计

矢量处理器可以进行矢量和标量的运算，支持 RISCV 矢量扩展的 SIMD 指令。使用的指令集包括 MOV，LOAD，VLOAD，MAC，STORE，VSTORE。格式如图 2 所示。

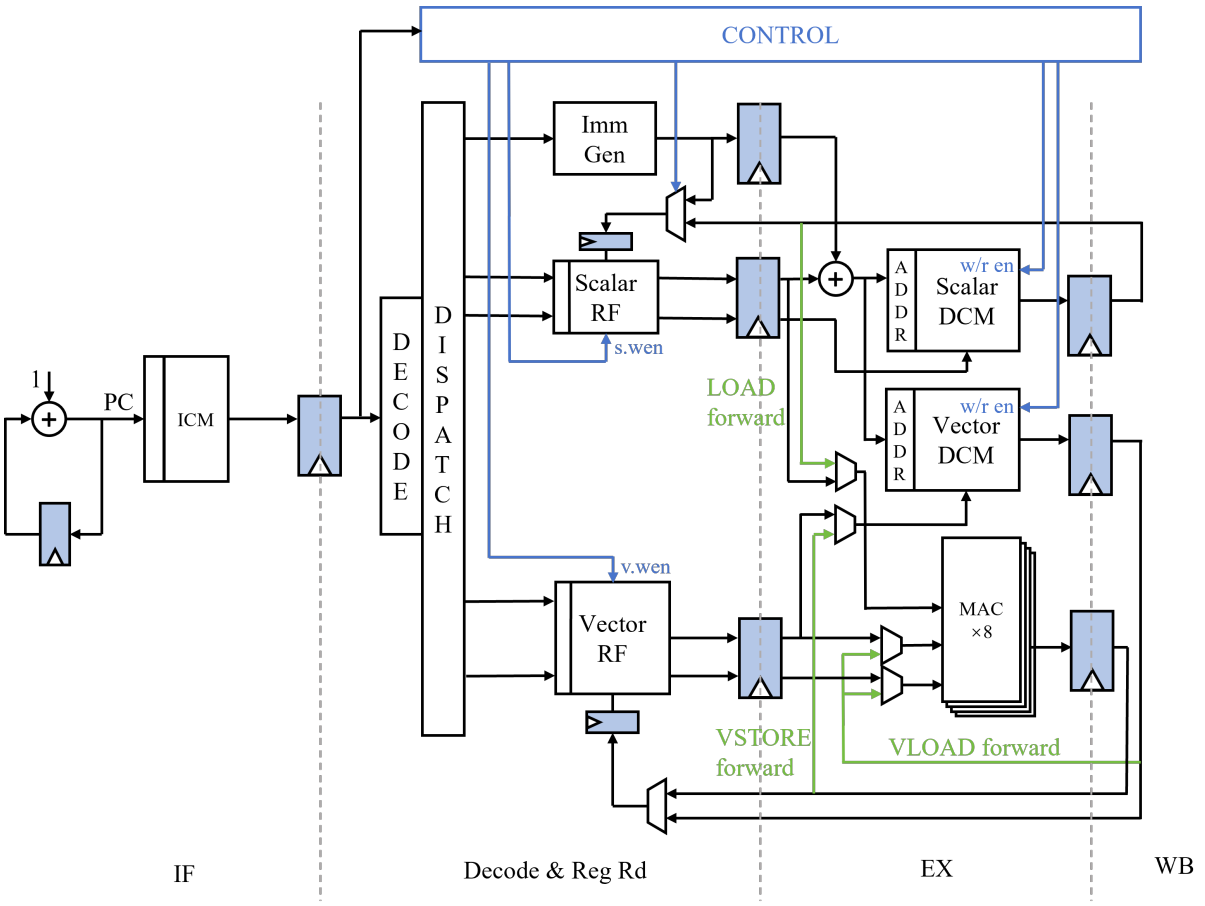


图 1: 架构图

func7															func3						opcode							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
imm[19:0]																				rd					0110111							MOV
imm[11:0]												rs1					010			rd					0000011							LOAD
imm[11:0]												rs1					011			vrd					0000011							VLOAD
0000001/0100001							rs1					vrs1					000			vrd					0010011							MAC
imm[11:5]							rs1					rs1					010			imm[4:0]					0100011							STORE
imm[11:5]							vrs1					rs1					011			imm[4:0]					0100011							VSTORE

图 2: 指令集格式设计

MOV 指令使用的操作码对应 RISCV 指令集中的 AUIPC，将立即数直接加载到标量寄存器中，可以表示为

```
imm -> ScalarRF[rd]
```

LOAD 和 VLOAD 指令的操作码都对应 RISCV 指令集中的 LOAD，用 func3 码段进行区分，将源寄存器中的数加上立即数偏移量作为访存的地址，将存储器的读取结果写入寄存器堆中，可以表示为

```
ScalarDCM[ScalarRF[rs1] + imm] -> ScalarRF[rd]
```

或者

```
VectorDCM[ScalarRF[rs1] + imm] -> VectorRF[vrd]
```

MAC 指令的操作码对应 RISCV 指令集中的计算类指令，为了和其他指令冲突，将 func7 的最低为设为 1，同时 func7 的第 6 位作为配置位，如果是 1 则是 reset 模式，是 0 则是计算模式。计算时先将目标矢量寄存器的数据和另一个矢量寄存器的数据读出，即使用 vrd 和 vrs1 作为矢量寄存器的地址读数，进行乘累加运算后再写回到目标矢量寄存器中。此时矢量寄存器需要一次读出两个数据，因此需要两个端口。可以表示为

```
Scalar[rs1] * VectorRF[vrs1] + VectorRF[vrd(vrs2)] -> VectorRF[vrd]
```

或者

```
VectorRF[vrs1] -> VectorRF[vrd]
```

STORE 和 VSTORE 的操作码对应 RISCV 指令集中的存储类指令，两种指令用 func3 码段进行区分，将源寄存器读出的数据加上立即数偏移量作为访存的地址，将寄存器堆中的数据写入存储器中。STORE 指令用来存储标量，此时标量寄存器需要同时读出访存地址和存储的数据，因此需要两个端口。指令功能可以表示为

```
ScalarRF[rs2]  -> ScalarDCM[ScalarRF[rs1] + imm]
```

或者

```
VectorRF[vrs1] -> VectorDCM[ScalarRF[rs1] + imm]
```

2.3 指令执行过程

控制模块的功能比较复杂，使能和选择信号的产生需要结合指令内容和不同流水级，按照正确的时序给出，这与每个指令的执行顺序有关。下面将结合每个指令的执行过程具体说明控制信号的产生逻辑。

2.3.1 MOV 指令

译码器接收到指令的同时，经过一个组合电路产生立即数，同时标量寄存器的写地址（rd）也生成完毕，下一个周期写入标量寄存器。因此，控制单元需要在接收到指令的同时产生标量寄存器的写使能信号（ScalarRegWriteEn），才能在下一个周期写入数据，同时标量寄存器输入数据的选择器（ScalarRegRdSel）也要切换到立即数的通路。

因此，对于 MOV 指令，rd，ScalarRegWriteEn 和 ScalarRegRdSel 都是组合逻辑直接产生的，与译码在同一周期完成。

2.3.2 LOAD 指令

译码器收到指令，下一个周期从标量寄存器中读出数据并和立即数相加，此时需要存储器的读使能信号（ScalarMemRead）有效，下一个周期从存储器中读出数据，再下一个周期写回寄存器。

写回地址（rd）和写使能（ScalarRegWriteEn）需要在收到指令两个周期后输入到标量寄存器。实际上寄存器输入的选择信号也需要在两个周期后产生，但是因为没有其他情况需要用立即数直接写入寄存器，所以只要保持选择存储器的数据通路即可。

因此，对于 LOAD 指令，ScalarMemRead 在收到指令的下一个周期有效，rd 和 ScalarRegWriteEn 需要经过两个周期的延时后才能更新。

2.3.3 VLOAD 指令

VLOAD 和 LOAD 的执行过程类似，接收到指令的下一个周期读出寄存器的数据并与偏移量相加产生地址，此时需要存储器的读使能信号（VectorMemRead）有效，下一个周期访问矢量存储器得到数据，再下一个周期写回矢量寄存器，此时需要写回的数据选择器（VectorRegRdSel）选择矢量存储器的数据通路。

因此，和 LOAD 指令同理，VectorMemRead 在收到指令的下一个周期有效，vrd，VectorRegWriteEn 和 VectorRegRdSel 需要经过两个周期的延时后才能更新。

2.3.4 MAC 指令

译码器接收到指令的下一个周期从矢量寄存器中读出两个矢量数据，从标量寄存器中读出一个标量数据，下一个周期进行乘累加运算，再下一个周期写回到矢量寄存器，此时需要写回的数据选择器（VectorRegRdSel）选择 MAC 的数据通路。

因此，对于 MAC 指令，vrd，VectorRegWriteEn 和 VectorRegRdSel 需要经过两个周期的延时后才能更新。

2.3.5 STORE 指令

接收到指令后，下一个周期从标量寄存器读出需要存储的数据和存储地址，此时存储器的写使能信号（ScalarMemWrite）有效，下一个周期写入标量存储器。

因此，对于 STORE 指令，ScalarMemWrite 在收到指令的下一个周期更新。

2.3.6 VSTORE 指令

VSTORE 和 STORE 执行过程相似，只是写入的存储器变为矢量存储器。需要写使能信号（VectorMemWrite）在收到指令的下一个周期更新。

2.4 前馈设计

当出现写后读（RAW）时，可能导致数据冒险，在前一条指令未写回时就读出目标寄存器的数据，造成错误的结果。分析指令间的依赖关系，主要的 RAW 数据冒险分为两类：

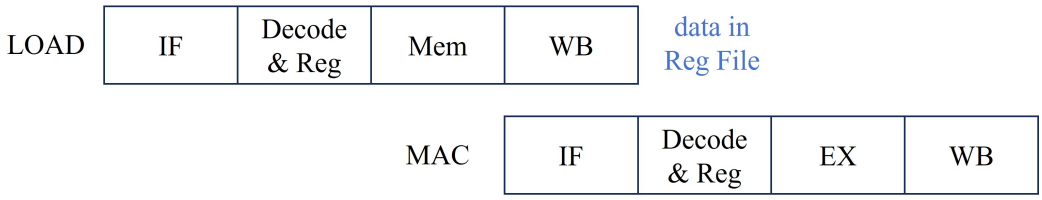
2.4.1 MAC 指令造成的 RAW

执行 MAC 指令时操作数还没有加载到寄存器中，就被读出作为乘加运算的输入。

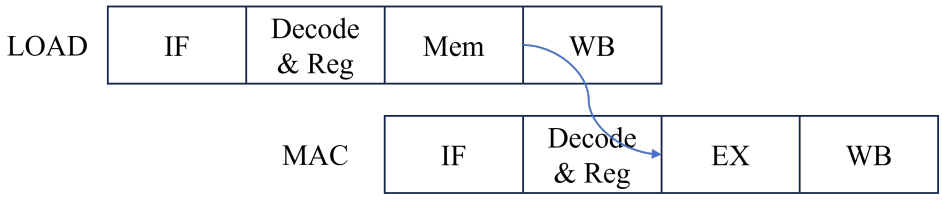
通过分析时序关系，第一个周期译码器接受到加载指令，第二个周期存储器地址生成，第三个周期数据读出，第四个周期数据写回，而译码器接收到 MAC 指令的同时就需要读取寄存器的数。因此，从加载指令发出开始计算，需要经过三个周期再发出 MAC 指令才能避免 RAW，如图 3（a）所示。

具体而言，有分为两种情况，分别如图 3（b）和（c）所示，情况 1 的加载指令在 MAC 指令前两个周期发出，当 MAC 需要读取操作数时，加载的数据正要写回，无法赶上，需要从访存的结果经过一个周期前馈到 MAC 模块。情况 2 的加载指令在 MAC 指令前一个周期发出，当 MAC 需要读取操作数时，加载的数据刚刚从存储器输出，需要直接前馈到 MAC 模块。

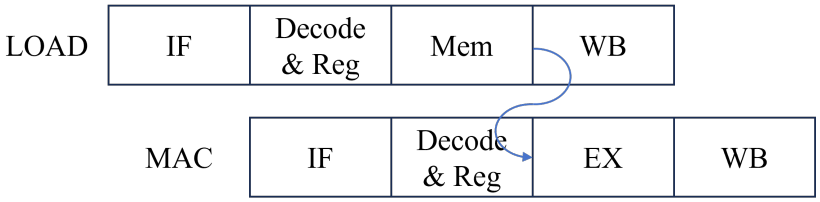
需要实现这个前馈的控制，需要由控制器比对当前指令和之前一条指令，以及再之前一条指令的目标寄存器地址，来判断是否需要前馈，并选择正确的前馈通路。



(a) 无 RAW



(b) RAW 情况 1



(c) RAW 情况 2

图 3: MAC 指令造成的 RAW 数据冒险情况

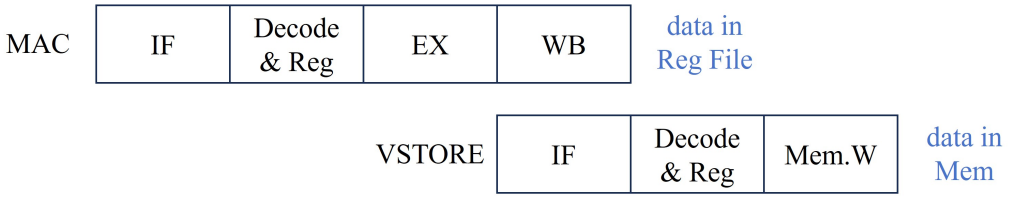
2.4.2 VSTORE 造成的 RAW

执行 VSTORE 指令时，由于之前的运算操作还未写回，导致矢量寄存器还没有加载到数据，就被读出作为存储的地址。

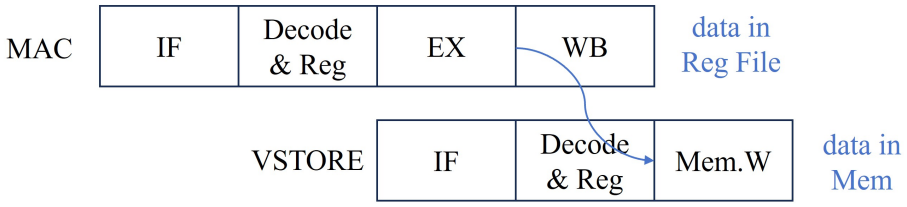
一般而言，不会出现加载数据后立刻存储相同数据的情况，更常见的情况是计算完成后将结果存到存储器。在这个架构中，不支持标量的计算，因此不考虑标量的存储导致的 RAW。

和前一种 RAW 类似，MAC 指令在发出后需要经过三个周期才能发出 VSTORE 指令，如图 4（a）所示。具体的 RAW 有两种情况，如图 4（a）和（b）所示，情况 1 中，需要将 MAC 的计算结果延时一个周期后前馈到矢量存储器。尽管看似仍有一个周期的延时，但是

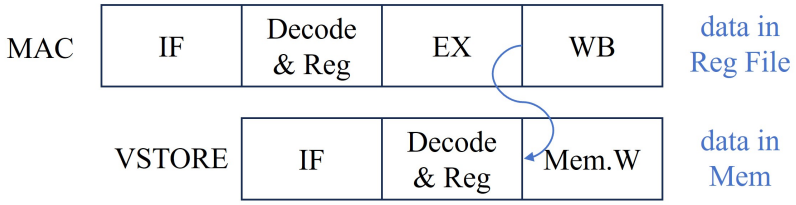
如果不使用前馈，而是写回寄存器，就会消耗两个周期，因此前馈仍然是必要的。情况 2 中，对于 VSTORE 指令而言，MAC 的结果出现的更晚，需要直接前馈到矢量存储器的输入端。



(a) 无 RAW



(b) RAW 情况 1



(c) RAW 情况 2

图 4: VSTORE 指令造成的 RAW 数据冒险情况

在实际架构中，为了应对 MAC 造成的 RAW，添加了 VLOAD Forward 和 LOAD Forward 数据通路以及相应的数据选择器，分别用于矢量和标量的前馈。为了应对 VSTORE 造成的前馈，添加了 VSTORE Forward 数据通路以及相应的数据选择器。由于位置有限，架构图中没有进一步画出控制信号和每种前馈通路的不同情况，只画出通路的示意图。

3 设计实现

3.1 顶层模块

顶层模块包括了指令缓存，译码器，立即数生成模块，标量寄存器和矢量寄存器，标量缓存和矢量缓存，乘加单元和控制单元。在模块内部实现了指令计数器，数据选择器，以及各个模块的连线，按照架构图的模式搭建，使各个模块构成一个完整的 VPU。

模块提供了外部交互的接口，分为三种模式，写入、执行和读取。写入模式下，VPU 不工作，由外部对指令缓存、标量存储器和向量存储器进行数据写入，在这个模式下实现了 VPU 的编程和数据的初始化；执行模式下，VPU 正常工作，从指令缓存逐条读取指令，执行指令并更新寄存器和存储器；读取模式下，VPU 停止工作，外部可以从标量缓存和矢量缓存中读取数据，用于验证计算结果。具体端口说明如图 5 所示。

信号名	类型	位宽	描述
clk	input	1	时钟信号
rst_n	input	1	异步复位信号，低电平有效
write_mode	input	1	写模式有效信号
icm_wr_addr	input	ICM_ADDR_WIDTH	指令缓存写入地址
scalar_wr_addr	input	DCM_ADDR_WIDTH	标量寄存器写入地址
vector_wr_addr	input	VDCM_ADDR_WIDTH	矢量寄存器写入地址
inst_data	input	INST_WIDTH	指令写入数据
scalar_wr_data	input	DATA_WIDTH	标量写入数据
vector_wr_data	input	VREG_DATA_WIDTH* VECTOR_LENGTH	矢量写入数据
start	input	1	为1时VPU开始工作
read_mode	input	1	读模式有效信号
scalar_rd_addr	input	DCM_ADDR_WIDTH	标量存储器读取地址
vector_rd_addr	input	VDCM_ADDR_WIDTH	矢量存储器读取地址
ScalarReadData	output	DATA_WIDTH	标量读取数据
VectorReadData	output	VREG_DATA_WIDTH * VECTOR_LENGTH	矢量读取数据
参数名	默认参数值	参数描述	
ICM_ADDR_WIDTH	10	指令缓存地址位数	
ICM_NUMBER	2**ICM_ADDR_WIDTH	指令缓存大小	
INST_WIDTH	32	指令位宽	
REG_ADDR_WIDTH	5	标量寄存器地址位宽	
DCM_ADDR_WIDTH	10	标量存储器地址位宽	
REG_NUMBER	2**REG_ADDR_WIDTH	标量寄存器大小	
DCM_NUMBER	2**DCM_ADDR_WIDTH	标量存储器大小	
DATA_WIDTH	32	数据位宽	
VREG_ADDR_WIDTH	5	矢量寄存器地址位宽	
VDCM_ADDR_WIDTH	10	矢量存储器地址位宽	
VREG_NUMBER	2**VREG_ADDR_WIDTH	矢量寄存器大小	
VDCM_NUMBER	2**VDCM_ADDR_WIDTH	矢量存储器大小	
VREG_DATA_WIDTH	32	矢量数据单数据位宽	
VECTOR_LENGTH	8	矢量长度	

图 5: 顶层模块端口说明

4 测试结果

4.1 指令测试

4.2 整体测试

测试输入为三个 8*8 的矩阵，一个存储在向量缓存中，两个存储在标量缓存中，计算矩阵的乘加运算，测试数据为随机生成，如下：

$$Matrix_1 = \begin{bmatrix} -69 & 95 & 73 & -55 & 17 & 32 & -16 & 24 \\ 100 & -93 & 56 & 41 & 47 & 83 & -69 & 4 \\ 77 & -83 & -26 & -78 & -14 & -27 & 75 & 1 \\ -54 & 62 & 88 & 13 & -18 & 39 & 0 & 97 \\ -12 & 85 & 58 & -80 & 44 & 53 & -99 & 66 \\ -37 & 7 & 99 & 25 & -61 & 18 & 55 & -92 \\ -70 & 49 & -34 & 81 & 60 & -47 & 28 & -85 \\ -2 & 100 & -59 & 36 & -77 & 72 & 11 & -63 \end{bmatrix} \tag{1}$$

$$Matrix_2 = \begin{bmatrix} -88 & 14 & 67 & -99 & 53 & 80 & -41 & 22 \\ -7 & 91 & -62 & 38 & 100 & -56 & 19 & -84 \\ -35 & 60 & 27 & -90 & 45 & 8 & -30 & 73 \\ 59 & -13 & 92 & -75 & 31 & -68 & 85 & -24 \\ -96 & 70 & 2 & 99 & -50 & 63 & -17 & 44 \\ 81 & -28 & 54 & -61 & 12 & 97 & -79 & 6 \\ -58 & 35 & 100 & -87 & 29 & -32 & 76 & -9 \\ 40 & -95 & 21 & 65 & -73 & 58 & -12 & 90 \end{bmatrix} \tag{2}$$

$$Matrix_3 = \begin{bmatrix} 81 & -44 & 56 & -90 & 13 & 67 & -38 & 72 \\ -99 & 35 & -73 & 40 & 86 & -65 & 17 & 33 \\ 27 & -88 & 62 & -41 & 19 & 77 & -56 & 84 \\ -13 & 95 & -22 & 59 & -80 & 36 & 48 & -71 \\ 61 & -24 & 79 & -92 & 55 & 12 & -38 & 70 \\ -47 & 81 & -66 & 28 & -35 & 99 & -21 & 10 \\ 53 & -60 & 44 & -85 & 72 & -18 & 25 & -97 \\ -31 & 67 & -49 & 90 & -76 & 38 & 63 & -29 \end{bmatrix} \tag{3}$$

理论结果为

$$\begin{bmatrix} 2536 & 10184 & -12880 & 10589 & 4754 & -370 & -6590 & 467 \\ -1416 & -6030 & 15437 & -15656 & -3770 & 24255 & -16693 & 16696 \\ -15013 & -4803 & 8524 & -8827 & -5310 & 10138 & -2581 & 7361 \\ 10759 & -1475 & 195 & 1010 & 1908 & 339 & -2034 & 7817 \\ 2223 & 3924 & -17353 & 19132 & -1204 & 15105 & -19722 & 7905 \\ 1614 & 11706 & 6412 & -24730 & 15511 & -13354 & 5683 & -6116 \\ -2752 & 14897 & -2553 & 6538 & 5696 & -20747 & 17572 & -15723 \\ 13700 & 4095 & -1153 & -10369 & 17911 & -10515 & 4088 & -22369 \end{bmatrix} \tag{4}$$

测试用的汇编代码如下：

```
// line1
MOV     R1,     0x0     // R1 = 0x0
VLOAD   VR2,    R0,     0x8     // VR2 = Matrix_3[0][:]
VMAC    VR3,    R2,     VR2, 1  // VR3 = VR2

LOAD    R2,     R1,     0x0     // R2 = Matrix_1[0][0]
VLOAD   VR2,    R0,     0x0     // VR2 = Matrix_2[0][:]
VMAC    VR3,    R2,     VR2, 0  // VR3 = R2 * VR2 + VR3

LOAD    R2,     R1,     0x1     // R2 = Matrix_1[0][1]
VLOAD   VR2,    R0,     0x1     // VR2 = Matrix_2[1][:]
VMAC    VR3,    R2,     VR2, 0  // VR3 = R2 * VR2 + VR3

LOAD    R2,     R1,     0x2     // R2 = Matrix_1[0][2]
VLOAD   VR2,    R0,     0x2     // VR2 = Matrix_2[2][:]
VMAC    VR3,    R2,     VR2, 0  // VR3 = R2 * VR2 + VR3

LOAD    R2,     R1,     0x3     // R2 = Matrix_1[0][3]
VLOAD   VR2,    R0,     0x3     // VR2 = Matrix_2[3][:]
VMAC    VR3,    R2,     VR2, 0  // VR3 = R2 * VR2 + VR3

LOAD    R2,     R1,     0x4     // R2 = Matrix_1[0][4]
VLOAD   VR2,    R0,     0x4     // VR2 = Matrix_2[4][:]
```

```
24      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
25
26      LOAD     R2,   R1,   0x5       // R2 = Matrix_1[0][5]
27      VLOAD   VR2,   R0,   0x5       // VR2 = Matrix_2[5][:]
28      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
29
30      LOAD     R2,   R1,   0x6       // R2 = Matrix_1[0][6]
31      VLOAD   VR2,   R0,   0x6       // VR2 = Matrix_2[6][:]
32      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
33
34      LOAD     R2,   R1,   0x7       // R2 = Matrix_1[0][7]
35      VLOAD   VR2,   R0,   0x7       // VR2 = Matrix_2[7][:]
36      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
37
38      MOV      R3,         0x10      // R3 = 0x10(after matrix3)
39      VSTORE   R3,   VR2             // store VR2 to VectorDCM[16]
40
41      // line2
42      MOV      R1,         0x8       // R1 = 0x8
43      VLOAD   VR2,   R0,   0x9       // VR2 = Matrix_3[1][:]
44      VMAC    VR3,   R2,   VR2,  1   // VR3 = VR2
45
46      LOAD     R2,   R1,   0x0       // R2 = Matrix_1[1][0]
47      VLOAD   VR2,   R0,   0x0       // VR2 = Matrix_2[0][:]
48      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
49
50      LOAD     R2,   R1,   0x1       // R2 = Matrix_1[1][1]
51      VLOAD   VR2,   R0,   0x1       // VR2 = Matrix_2[1][:]
52      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
53
54      LOAD     R2,   R1,   0x2       // R2 = Matrix_1[1][2]
55      VLOAD   VR2,   R0,   0x2       // VR2 = Matrix_2[2][:]
56      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
57
58      LOAD     R2,   R1,   0x3       // R2 = Matrix_1[1][3]
59      VLOAD   VR2,   R0,   0x3       // VR2 = Matrix_2[3][:]
60      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
61
62      LOAD     R2,   R1,   0x4       // R2 = Matrix_1[1][4]
63      VLOAD   VR2,   R0,   0x4       // VR2 = Matrix_2[4][:]
64      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
65
66      LOAD     R2,   R1,   0x5       // R2 = Matrix_1[1][5]
67      VLOAD   VR2,   R0,   0x5       // VR2 = Matrix_2[5][:]
68      VMAC    VR3,   R2,   VR2,  0   // VR3 = R2 * VR2 + VR3
```

```
69
70      LOAD    R2,   R1,   0x6      // R2 = Matrix_1[1][6]
71      VLOAD   VR2,  R0,   0x6      // VR2 = Matrix_2[6][:]
72      VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
73
74      LOAD    R2,   R1,   0x7      // R2 = Matrix_1[1][7]
75      VLOAD   VR2,  R0,   0x7      // VR2 = Matrix_2[7][:]
76      VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
77
78      MOV     R3,         0x11     // R3 = 0x11(after matrix3)
79      VSTORE  R3,   VR2            // store VR2 to VectorDCM[17]
80
81      ...
82
83      // line8
84      MOV     R1,         0x38      // R1 = 0x38
85      VLOAD   VR2,  R0,   0xf      // VR2 = Matrix_3[7][:]
86      VMAC    VR3,  R2,   VR2, 1   // VR3 = VR2
87
88      LOAD    R2,   R1,   0x0      // R2 = Matrix_1[7][0]
89      VLOAD   VR2,  R0,   0x0      // VR2 = Matrix_2[0][:]
90      VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
91
92      LOAD    R2,   R1,   0x1      // R2 = Matrix_1[7][1]
93      VLOAD   VR2,  R0,   0x1      // VR2 = Matrix_2[1][:]
94      VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
95
96      LOAD    R2,   R1,   0x2      // R2 = Matrix_1[7][2]
97      VLOAD   VR2,  R0,   0x2      // VR2 = Matrix_2[2][:]
98      VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
99
100     LOAD    R2,   R1,   0x3      // R2 = Matrix_1[7][3]
101     VLOAD   VR2,  R0,   0x3      // VR2 = Matrix_2[3][:]
102     VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
103
104     LOAD    R2,   R1,   0x4      // R2 = Matrix_1[7][4]
105     VLOAD   VR2,  R0,   0x4      // VR2 = Matrix_2[4][:]
106     VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
107
108     LOAD    R2,   R1,   0x5      // R2 = Matrix_1[7][5]
109     VLOAD   VR2,  R0,   0x5      // VR2 = Matrix_2[5][:]
110     VMAC    VR3,  R2,   VR2, 0   // VR3 = R2 * VR2 + VR3
111
112     LOAD    R2,   R1,   0x6      // R2 = Matrix_1[7][6]
113     VLOAD   VR2,  R0,   0x6      // VR2 = Matrix_2[6][:]
```

```
114     VMAC    VR3,   R2,   VR2,  0    // VR3 = R2 * VR2 + VR3
115
116     LOAD     R2,   R1,   0x7        // R2 = Matrix_1[7][7]
117     VLOAD   VR2,   R0,   0x7        // VR2 = Matrix_2[7][:]
118     VMAC    VR3,   R2,   VR2,  0    // VR3 = R2 * VR2 + VR3
119
120     MOV      R3,         0x17       // R3 = 0x17(after matrix3)
121     VSTORE   R3,   VR2               // store VR2 to VectorDCM[23]
```

计算的结果存放在 VectorDCM 中，程序运行结束后将计算结果读出并在 tcl 窗口中打印，结果如下：

```
run all
Read VDCM Data: 000009e8000027c8ffffcdb00000295d00001292fffffe8effffe642000001d3
Read VDCM Data: fffffa78ffffe87200003c4dffffc2d8fffff14600005ebfffffbecb00004138
Read VDCM Data: ffffc55bffffed3d0000214cffffdd85ffffeb420000279afffff5eb00001cc1
Read VDCM Data: 00002a07fffffa3d000000c3000003f20000077400000153fffff80e00001e89
Read VDCM Data: 000008af00000f54ffffbc3700004abcffffffb4c00003b01ffffb2f600001ee1
Read VDCM Data: 0000064e00002dba0000190cffff9f6600003c97ffffcbd600001633ffffe81c
Read VDCM Data: fffff54000003a31fffff6070000198a00001640ffffaef5000044a4ffffc295
Read VDCM Data: 00003584000000ffffffffb7ffffffd77f000045f7ffffd6ed00000ff8ffffa89f
$finish called at time : 5660 ns : File "D:/Vivado Workplace/RISCV_vector_processor
```

图 6: m32n8k16 矩阵乘法任务分配示意图

对应的十进制数与理想结果相同，表明处理器可以正常工作。

5 优化分析