“乘影”GPGPU 指令集架构

文档手册

v2025.03.21

简介

本文档从指令集架构及软硬件接口角度描述了乘影GPGPU的设计内容。
乘影GPGPU指令集以RISC-V向量扩展（后文简称为RVV）为核心设计GPGPU，相比RISC-V标量指令，具有更丰富的表达含义，可以实现访存特性表征、区分workgroup和thread操作等功能。核心思想是在编译器层面以v指令作为thread的行为描述，并将thread->warp/workgroup的公共数据合并为标量指令。硬件上一个warp就是一个RVV程序，通常向量元素长度为num_thread，同时又将workgroup中统一执行的公共地址计算、跳转等作为标量指令执行，即Vector-Thread架构。硬件将warp分时映射到RVV处理器的lane上去执行。
乘影SIMT架构计算单元采用SIMD（Vector）执行方式，以workgroup（或分支状态下的warp_split）执行。RVV指令集在变长上有三个方面的体现：硬件vlen改变；SEW元素宽度改变；LMUL分组改变。本架构特点在于这三个参数在编译期都已固定，元素数目大部分情况也固定为num_thread。

术语表

- SM：streaming multiprocessor，流多处理器单元
- sGPR：scalar general purpose register，标量寄存器
- vGPR：vector general purpose register，向量寄存器

GPGPU中常用概念及释义如下表。

cuda	opencl	解释
globalmem	globalmem	全局内存，用__global描述，可以被kernel的所有线程访问到
constantmem	constantmem	常量内存，用__constant描述，是全局地址空间的一部分
localmem	privatemem	私有内存，各thread自己的变量，和内核参数，是全局内存的一部分
sharedmem	localmem	局部内存，用__local描述，供同一work-group间的线程进行数据交换
grid	NDRange	用于描述一个kernel内的线程块/工作组的数量
block/CTA	workgroup	工作组，在SM上执行的基本单位
warp	wavefront(AMD)	32个thread组成一个warp，仅对硬件可见
thread	work-item	线程/工作项，是OpenCL C编程时描述的最小单位。

编程模型和驱动程序功能

这部分从乘影作为device和OpenCL编程框架的交互来介绍OpenCL程序在执行前后的行为。乘影的编程模型兼容OpenCL，即乘影的硬件层次与OpenCL的执行模型具有一一对应的关系。此外，OpenCL编程框架需要提供一系列运行时实现，完成包括任务分配，设备内存空间管理，命令队列管理等功能。

任务执行模型

在OpenCL中，计算平台分为主机端（host）和设备端（device），其中device端执行内核（kernel），host端负责交互，资源分配和设备管理等。Device上的计算资源可以划分为计算单元（computing units，CU），CU可以进一步划分为处理单元（processing elements，PE），设备上的计算都在PE中完成。
乘影的一个SM对应OpenCL的一个计算单元CU，每个向量车道（lane）对应OpenCL的一个处理单元PE。
Kernel在设备上执行时，会存在多个实例（可以理解为多线程程序中的每个线程），每个实例称为一个工作项（work-item），work-item会组织为工作组（work-group），work-group中的work-item可以并行执行，要说明的是，OpenCL只保证了单个work-group中的work-item可以并发执行，并没有保证work-group之间可以并发执行，尽管实际情况中通常work-group是并行的。
在乘影的实现中，一个work-item对应一个线程，在执行时将会占用一个向量车道，由于硬件线程会被组织成线程组（warp）锁步执行，而warp中的线程数量是固定的，因此一个work-group在映射到硬件时可能对应多个warp，但这些组成一个work-group的warp将会保证在同一个SM上执行。

驱动提供的功能

乘影的驱动分为两层，一层是OpenCL的运行时环境，一层是硬件驱动，运行时环境基于OpenCL的一个开源实现pocl实现，主要管理命令队列（command queue），创建和管理buffer，管理OpenCL事件（events）和同步。硬件驱动是对物理设备的一层封装，主要是为运行时环境提供一些底层接口，并将软件数据结构转换为硬件的端口信号，这些底层接口包括分配、释放、读写设备内存，在host和device端进行buffer的搬移，控制device开始执行等。运行时环境利用硬件驱动的接口实现和物理设备的交互。
每个kernel都有一些需要硬件获取的信息，这些信息由运行时创建，以buffer的形式拷贝到device端的内存空间。目前运行时环境创建的buffer包括：

- NDRange的metadata buffer和kernel机器码
- kernel的argument data buffer，即每个kernel参数的具体输入
- kernel_arg_buffer，保存的内容为device端的指针，其值为kernel的argument data buffer在device端的地址。
- 为private mem、print buffer分配的空间

其中，metadata buffer保存kernel的一些属性，具体内容为：

```
cl_int clEnqueueNDRangeKernel(cl_command_queue command_queue,
                              cl_kernel kernel,                                 //kernel_entry_ptr & kernel_arg_ptr
                              cl_uint work_dim,                                 //work_dim
                              const size_t *global_work_offset, //global_work_offset_x/y/z
                              const size_t *global_work_size,   //global_work_size_x/y/z
                              const size_t *local_work_size,    //local_work_size_x/y/z
                              cl_uint num_events_in_wait_list,
                              const cl_event *event_wait_list,
                              cl_event *event)
/*
#define KNL_ENTRY 0
#define KNL_ARG_BASE 4
#define KNL_WORK_DIM 8
#define KNL_GL_SIZE_X 12
#define KNL_GL_SIZE_Y 16
#define KNL_GL_SIZE_Z 20
#define KNL_LC_SIZE_X 24
#define KNL_LC_SIZE_Y 28
#define KNL_LC_SIZE_Z 32
#define KNL_GL_OFFSET_X 36
#define KNL_GL_OFFSET_Y 40
#define KNL_GL_OFFSET_Z 44
#define KNL_PRINT_ADDR 48
#define KNL_PRINT_SIZE 52
*/
```

Kernel启动时的行为

任务启动时，硬件驱动需要传递给物理device一些信号，这些信号有：

信号	含义
PTBR	page table base addr
CSR_KNL	metadata buffer base addr
CSR_WGID	当前workgroup在SM中的id，仅供硬件辨识
LDS_SIZE	localmem_size，编译器提供workgroup需要占用的localmem空间。 privatemem_size默认按照每个线程1kB来分配。
VGPR_SIZE	vGPR_usage，编译器提供workgroup实际使用的vGPR数目（对齐4）
SGPR_SIZE	sGPR_usage，编译器提供workgroup实际使用的sGPR数目（对齐4）
CSR_GIDX/Y/Z	workgroup idx in NDRange
host_wf_size	一个warp中thread数目
host_num_wf	一个workgroup中warp数目

kernel的参数由前述的kernel_arg_buffer传递，该buffer中会按顺序准备好kernel的argument，包括具体参数值或其它buffer的地址。在NDRange的metadata中仅提供kernel_arg_buffer的地址knl_arg_base。 kernel函数执行前会先执行start.S：

```
# start.S
_start:
  # set global pointer register
  .option push
  .option norelax
  la gp, __global_pointer$
  .option pop

  # allocate warp and per-thread level stack pointers, both
  # stacks grows upwards
  li t4,32
  vsetvli t4,t4,e32,m1,ta,ma
  li t4,0x2000
  csrrs t4, mstatus, t4
  li t4, 0
  csrr t1, CSR_WID
  csrr t2, CSR_LDS
  li t3, 1024           # 1M size for single warp
  mul t1, t1, t3        # sp wid * warp_size
  add sp, t1, t2        # sp points to baseaddr of local memory of each SM
  li tp, 0              # tp points to baseaddr for lower bound of private memory(1K) of each thread
  csrr t5, CSR_NUMW
  li t3, 1024
  mul t5, t5, t3
  add s0, t2, t5        # s0 points to local memory base addr in a workgroup

  # clear BSS segment
  la      a0, _edata
  la      a2, _end
  beq     a0, a2, 2f
1:
  sw      zero, (a0)
  addi    a0, a0, 4
  bltu    a0, a2, 1b

2:
  csrr t0, CSR_KNL                      # get addr of kernel metadata
  lw t1, KNL_ENTRY(t0)              # get kernel program address
  lw a0, KNL_ARG_BASE(t0)           # get kernel arg buffer base address
  lw t2, KNL_PRINT_ADDR(t0)         # get kernel print buffer address
  lw t3, KNL_PRINT_SIZE(t0)         # get kernel print buffer size
  la t4, BUFFER_ADDR
  la t5, BUFFER_SIZE
  sw t2, 0(t4)
  sw t3, 0(t5)
  la t6, spike_end                  # exception to stop spike
  csrw mtvec, t6
  jalr t1                           # call kernel program

  # call exit routine
  # tail    exit

  # End of warp execution
  endprg x0, x0, x0
```

```
  j spike_end
  .size  _start, .-_start


  .section ".tohost","aw",@progbits
  .align 6
  .globl tohost
  tohost: .dword 0
  .align 6
  .globl fromhost
  fromhost: .dword 0

  .text
  .global spike_end
  .type spike_end,function
spike_end:
  li t1,1
  la t0,tohost
  sw t1,0(t0)
# end.S
end:
        endprg
```

约定kernel的打印信息通过print buffer向host传递。print buffer的地址和大小在metadata_buffer中提供，运行中的thread完成打印后，将所属warp的CSR_PRINT置位。host轮询到有未处理信息时，将print buffer从设备侧取出处理，并将CSR_PRINT复位。

栈空间说明

由于OpenCL不允许在Kernel中使用malloc等动态内存函数，也不存在堆，因此可以让栈空间向上增长。tp用于各thread私有寄存器不足时压栈（即vGPR spill stack slots），sp用于公共数据压栈，（即sGPR spill stack slots，实际上sGPR spill stack slots将作为localmem的一部分），在编程中显式声明了__local标签的数据也会存在localmem中。 编译器提供localmem的数据整体使用量（按照sGPR spill 1kB，结合local数据的大小，共同作为localmem_size），供硬件完成workgroup的分配。

参数传递ABI

对于kernel函数，a0是参数列表的基址指针，第一个clSetKernelArg设置的显存起始地址存入a0 register，kernel默认从该位置开始加载参数。 对于非kernel函数，使用v0-v31和stack pointer传递参数，v0-v31作为返回值。

寄存器

寄存器设置

架构寄存器数目为sGPR（标量）64个，vGPR（向量）256个，元素宽度均为32位。
当需要64位数据时，数据以偶对齐的寄存器对（Register Pair）的形式进行存储，数据低32位存储在GPR[n]中，高32位存储在GPR[n+1]中。
目前硬件中物理寄存器数目为sGPR（标量）256个，vGPR（向量）1024个，GPU硬件负责实现架构寄存器到硬件寄存器的映射。
编译器提供vGPR、sGPR的实际使用数目（4的倍数），硬件会根据实际使用情况分配更多的workgroup同时调度。
从RISC-V Vector的视角来看寄存器堆，vGPR共有256个，宽度vlen固定为线程数目num_thread乘以32位，即相当于通过vsetvli指令设置SEW=32bit，ma，ta，LMUL为1。而从SIMT的视角来看寄存器堆，每个thread至多拥有256个宽度32位的vGPR。一个简单的理解是，将向量寄存器堆视作一个256行、num_thread列的二维数组，数组的每行是一个vGPR，而每列是一个thread最多可用的寄存器。OpenCL中定义了一些向量类型，这些向量类型需要使用分组寄存器的形式表达，即float16在寄存器堆中以列存储，占用16个vGPR的各32位。这部分工作由编译器进行展开。
workgroup拥有64个sGPR，整个workgroup只需做一次的操作，如kernel中的地址计算，会使用sGPR；如果有发生分支的情况，则使用vGPR，例如非kernel函数的参数传递。

一些特殊寄存器：

- x0：0寄存器；
- x1/ra：返回PC寄存器；
- x2/sp：栈指针 / local mem基址；
- x4/tp：private mem基址。

参数传递：
对于kernel函数，a0是参数列表的基址指针，第一个clSetKernelArg设置的显存起始地址存入a0 register，kernel默认从该位置开始加载参数。

自定义CSR

description	name	addr
该warp中id最小的thread id，其值为CSR_WID*CSR_NUMT，配合vid.v可计算其它thread id。	CSR_TID	0x800
该workgroup中的warp总数	CSR_NUMW	0x801
一个warp中的thread总数	CSR_NUMT	0x802
该workgroup的metadata buffer的baseaddr	CSR_KNL	0x803
该SM中本warp对应的workgroup id	CSR_WGID	0x804
该workgroup中本warp对应的warp id	CSR_WID	0x805
该workgroup分配的local memory的baseaddr，同时也是该warp的xgpr spill stack基址	CSR_LDS	0x806
该warp分配的private memory的baseaddr，同时是该thread的vgpr spill stack基址	CSR_PDS	0x807

description	name	addr
该workgroup在NDRange中的x id	CSR_GIDX	0x808
该workgroup在NDRange中的y id	CSR_GIDY	0x809
该workgroup在NDRange中的z id	CSR_GIDZ	0x80a
向print buffer打印时用于与host交互的CSR	CSR_PRINT	0x80b
重汇聚pc寄存器	CSR_RPC	0x80c

注：在汇编器中可以使用小写后缀来表示对应的CSR，例如用tid代替CSR_TID。

指令集架构

指令集范围

选用RV32V扩展作为基本指令集，支持指令集范围为：RV32I, M, A, zfinx, zve32f, RV64I, A.
V扩展指令集主要支持独立数据通路的指令，不支持RVV原有的shuffle, narrow, gather, reduction指令。
下表列出目前支持的标准指令范围，有变化指令已声明。

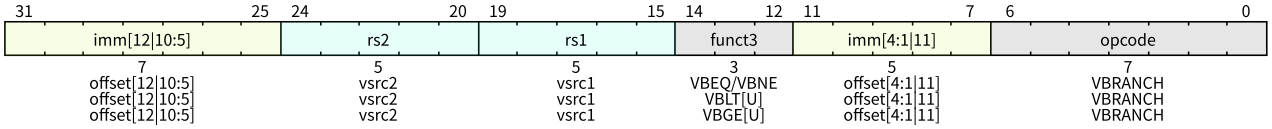
	乘影支持情况	指令变化
RV32I	不支持ecall ebreak	
RV32M F	支持RV32M zfinx zve32f	
RV32A	支持	
RV64I	部分支持	语义有变化，见后续章节
RV64A	部分支持	语义有变化，见后续章节
RV32V-Register State	仅支持LMUL=1和2	
RV32V-ConfigureSetting	支持计算vl，可通过该选项配置支持不同宽度元素	
RV32V-LoadsAndStores	支持vle32.v vlse32.v vluxei32.v访存模式	vle8等指令语义改为 “各thread向向量寄存器元素位置写入“， 而非连续写入
RV32V-IntergerArithmetic	支持绝大多数int32计算指令	vmv.x.s语义改为 “各thread均向标量寄存器写入”， 而非总由向量寄存器idx_0写入， 多线程同时写入是未定义行为， 正确性由程序员保证； vmv.s.x语义改为与vmv.v.x一致
RV32V-FixedPointArithmetic	添加int8支持，视应用需求再添加其它类型	
RV32V-FloatingPointArithmetic	支持绝大多数fp32指令，添加fp64 fp16支持	
RV32V-WideningIntergerArithmetic	支持绝大多数计算指令	2*SEW位宽元素通过寄存器对实现
RV32V-ReductionOperations	视应用和编译器需求再考虑添加， 例如需要支持OpenCL2.0中的work_group_reduce时	
RV32V-Mask	支持各lane独立计算和设置mask的指令	vmsle等指令语义改为 “各thread向向量寄存器元素位置写入”， 而非连续写入
RV32V-Permutation	不支持，视应用和编译器需求再考虑添加	

	乘影支持情况	指令变化
RV32V-ExceptionHandling	不支持，视应用和编译器需求再考虑添加	

自定义指令

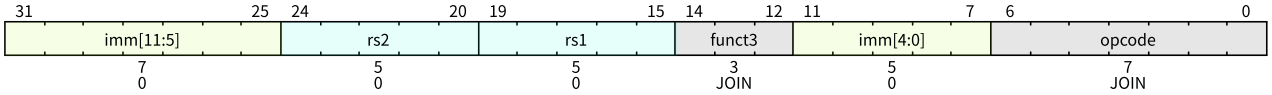
分支控制指令

VBRANCH分支指令使用B型指令格式。12位B型有符号立即数进行符号位扩展后加上当前的PC值给出else路径的起始位置PC，读取CSR_RPC的值rpc，根据向量寄存器计算比较结果操作SIMT线程分支管理栈。



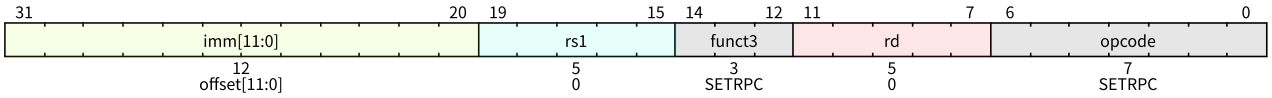
线程分支指令将比较两个向量寄存器，如果线程对应的元素相等，VBEQ的这些线程将跳转到else PC处执行，其余元素不等的线程将继续执行PC+4，分支两条路径的跳转、聚合和掩码控制由SIMT栈控制，将rpc，else PC和元素比较结果压入栈中，当全部活跃线程对应元素均相等或不等时，不触发栈操作，全部跳转到else PC或继续执行PC+4。当vs1和vs2中的操作数不等时，VBNE将判断对应线程跳转执行PC else还是PC+4。VBLT和VBLTU将分别使用有符号数和无符号数比较vs1和vs2，当对应的向量元素有vs1小于vs2时，对应线程跳转至PC else，剩余活跃线程继续执行PC+4。VBGE和VBGEU将分别使用有符号数和无符号数比较vs1和vs2，当对应的向量元素有vs1大于等于vs2时，对应线程跳转至PC else，剩余活跃线程继续执行PC+4。

线程分支汇聚join指令采用S型指令格式，默认源操作数、立即数均为0。



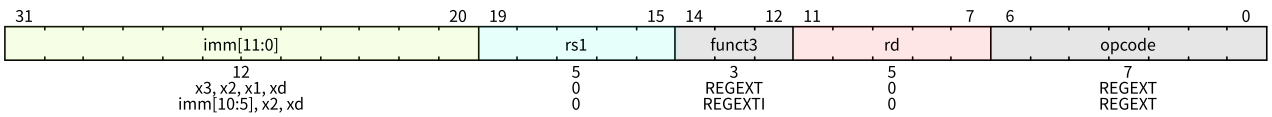
JOIN指令将对比当前指令PC与SIMT栈顶重汇聚PC是否相等，若相等，则设置活跃线程掩码为当前栈顶掩码项，跳转至栈顶else PC处执行指令，若不等，则不做任何操作。

重汇聚PC设置指令SETRPC将12位立即数进行符号位扩展并与源操作数相加后，将结果写入CSR_RPC的csr寄存器和目的寄存器。



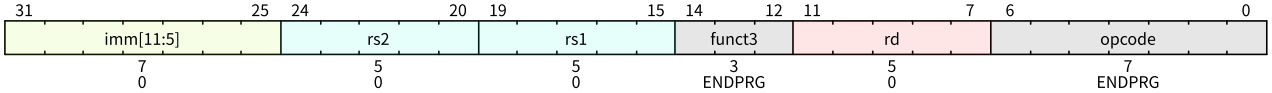
寄存器扩展指令

REGEXT和REGEXTI指令用于扩展它之后一条指令的寄存器编码及立即数编码。在REGEXT指令中，12位的立即数被分拆为四段3位数，分别拼接在下一条指令中寄存器rs3/vs3、rs2/vs2、rs1/vs1、rd/vd编码的高位上。REGEXTI指令则面向使用了5位立即数的指令，前缀包含6位的立即数高位，以及rs2/vs2、rd/vd编码的高位。11立即数由前缀指令中的立即数高位与5位立即数直接拼接得到。
此外，在不使用扩展的情况下，向量浮点运算的vs3就是vd的[11:7]段，但进行向量扩展时，vs3和vd的高3位分离存储。

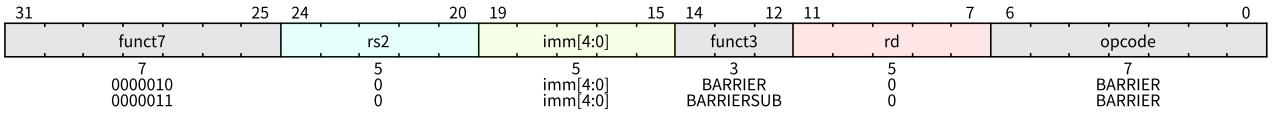


同步和任务控制指令

ENDPRG指令需要显式插入到Kernel末尾，以指示当前warp执行结束。只能在无分支的条件下使用。



BARRIER对应OpenCL的barrier(cl_mem_fence_flags flags)和work_group_barrier(cl_mem_fence_flags flags, [memory_scope scope])函数，实现同一workgroup内的thread间数据同步。memory_scope缺省值为 memory_scope_work_group。

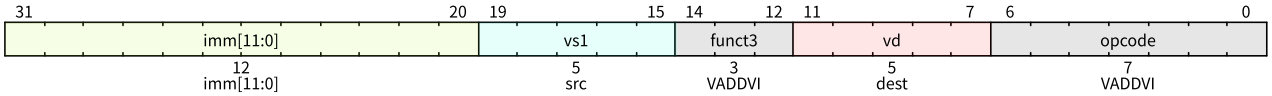


五位立即数字段编码如下：

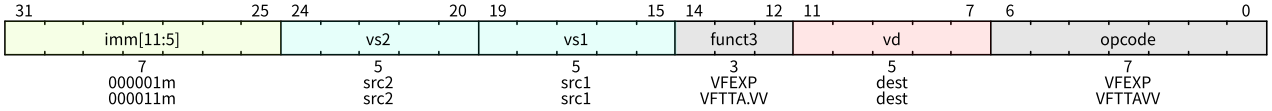
imm[4:3]	00	01	10	11
memory_scope	work_group(default)	work_item	device	all_svm_devices
imm[2:0]	**imm[2]=1**	**imm[1]=1**	**imm[0]=1**	**imm[2:0]=000**
CLK_X_MEM_FENCE	IMAGE	GLOBAL	LOCAL	USE_CNT

开启 opencl_c_subgroups 特性后，则改为barriersub指令，对应memory_scope=subgroup的情况，此时imm[4:3]固定为0，cl_mem_fence_flags为imm[2:0]，与barrier指令一致。

自定义计算指令



VADD12.VI将无符号数imm加到向量寄存器vs1赋予向量寄存器vd。



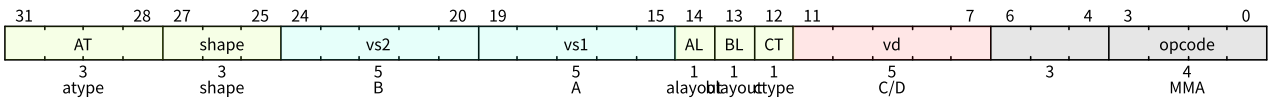
VFEXP计算exp(v2)赋予向量寄存器vd。VFTTAVV计算向量vs1和vs2的卷积，加上向量寄存器vd后赋予vd。

自定义矩阵乘加指令

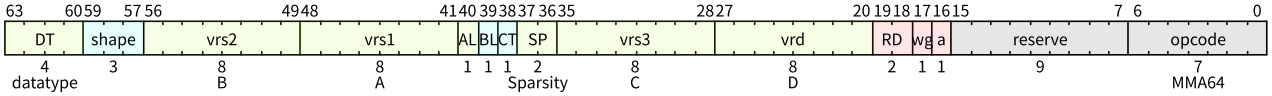
MMA指令用于计算矩阵乘加：D=A×B+C。A、B、C、D四个矩阵精度和维度可配置，其中A的维度为M×K，B的维度为K×N，C和D的维度为M×N，支持范围如下表：

A/B type	shape	C/D type
FP32	m8n8k8	FP32
	m16n8k8	
	m16n8k16	
FP16	m8n8k8	FP16
	m16n8k8	
	m16n8k16	
	m8n8k8	FP32
	m16n8k8	
	m16n8k16	
BF16	m8n8k8	FP32
	m16n8k8	
	m16n8k16	
FP8(E4M3)	m16n8k32	INT32
FP8(E5M2)	m16n8k32	
INT8	m8n8k16	
	m16n16k16	
INT4	m8n8k32	
	m16n8k32	
	m16n8k64	
Binary	m8n8k128	

指令中，vd字段同时作为源寄存器C和目的寄存器D的索引。



64bit指令字长矩阵乘加



MMA64指令作为MMA指令的扩展，功能上与MMA指令对齐，扩展的指令编码字段用于：扩展寄存器编码至8位，支持三个源操作数编码，C、D矩阵分别存储于不同的寄存器。MMA64指令支持FP16、BF16、INT8、INT4、Binary、FP8、FP4等精度；支持指定浮点运算不同舍入模式和整型运算是否输出饱和；预留空间支持稀疏性运算。

MMA64指令字段名称包括opcode（7位，标识指令类型为MMA64）、reserve（9位，保留字段）、a（1位，指示A操作数来源）、wg（1位，warp group标志位）、RD（2位，舍入模式/饱和溢出标志位）、vrd（8位，目标寄存器D编码）、vrs3（8位，源寄存器C编码）、SP（2位，稀疏性标志位）、CT（1位，C/D矩阵数据类型标志位）、BL（1位，B矩阵排列方式标志位）、AL（1位，A矩阵排列方式标志位）、vrs1（8位，源寄存器A编码）、vrs2（8位，源寄存器B编码）、shape（3位，矩阵维度标志位）以及DT（4位，数据类型标志位），这些字段共同定义了指令的详细行为和参数配置。

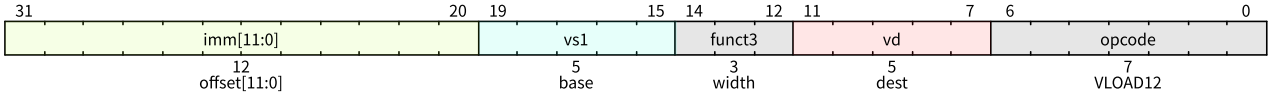
MMA64指令中的DT与CT分别指示矩阵A、B、C和D中的元素的数据类型，4位DT指示A、B矩阵数据精度，1位CT指示C、D矩阵精度。累加FP16数据时，矩阵A、B数据是FP16精度；累加INT32精度时，矩阵A、B数据是INT8、INT4、Binary精度；累加FP32精度时，矩阵A、B数据是FP16、BF16、FP8(E4M3\E5M2)、FP6(E3M2\E2M3)、FP4(E2M1\E3M0)等精度。计算规模指令3位，依数据精度变化而变化。

AL与BL标志位指示AB矩阵在寄存器的排布方式，A矩阵在AL=0时行优先，AL=1时列优先；B矩阵在BL=1时行优先，BL=0时列优先。RD（舍入模式\饱和溢出）标志位包含2位：对于C/D操作数位浮点精度时，此标志位表示舍入模式：00=rn，01=rz，10=rm，11=rp；对于C\D为整型操作数时，此标志位表示是否饱和：00=上溢饱和；01=不做限制。MMA64指令wg标志位为1时，指示本条指令是warp group层次指令，计算规模相较warp层级计算指令扩展，具体尺寸随精度不同而不同。在warp group计算层级下，定义a(R/SMEM)标志位：标志位为1时，表明A操作数来自寄存器；此标志位为0时，表明A操作数来自SMEM；两种情况下B操作数来自于SMEM。当操作数来自于SMEM时，寄存器中数据寄存器存储A矩阵描述符。计算精度、稀疏性、矩阵排列、舍入模式与warp level计算指令约定保持一致。

自定义立即数访存指令

自定义立即数访存指令支持每个线程访问global、local、private地址空间，指令根据访存地址计算方式的不同，分为flat、global、local、private四个系列。

Flat访存方式是通用的访存方式，可以通过flat访存指令访问到global、local、private内存空间。内存空间将以统一的地址空间进行编址，通过对源寄存器中代表的目标地址（addr）进行空间范围检查，确定具体落在哪个内存空间。

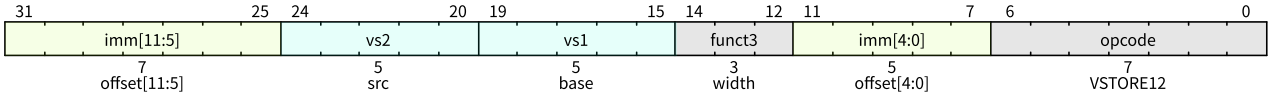


与标准RISC-V指令一致，自定义flat访存指令的地址空间按字节寻址且为小端格式。每个线程有效字节地址为Addr，将内存中以Addr为起始的元素复制到向量寄存器vd。VLW12指令从内存中加载32位值的向量到vd中。VLH12从内存中加载16位值，然后将其符号扩展到32位，再存储到vd中。VLHU12从内存中加载16位值，然后将其无符号扩展到32位，再存储到vd中。VLB12和VLBU12对8位值有类似定义。
其中Addr的计算根据空间范围检查的结果采用不同的方式计算，空间范围检查的对象为vs1中的数据，通过地址大小判断访存的目标空间。Addr计算方式与目标空间的关系如下表：

目标空间	Addr
global	Addr=vs1 + offset
local	Addr=vs1 + offset
private	Addr=((vs1+offset)[31:2]<<2) * num_thread_per_warp + (vs1+offset)[1:0] + thread_idx_in_warp<<2 + csr_pds[wid]

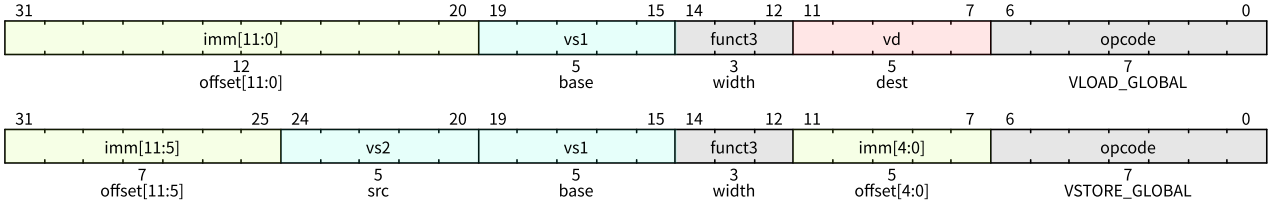
通过对vs1中的数据进行空间范围检查的方式如下：

vs1范围	目标空间
gds_base ≤ vs1 ≤ gds_limit	global
lds_base ≤ vs1 ≤ lds_limit	local
vs1[31:24] == 8'h00	private



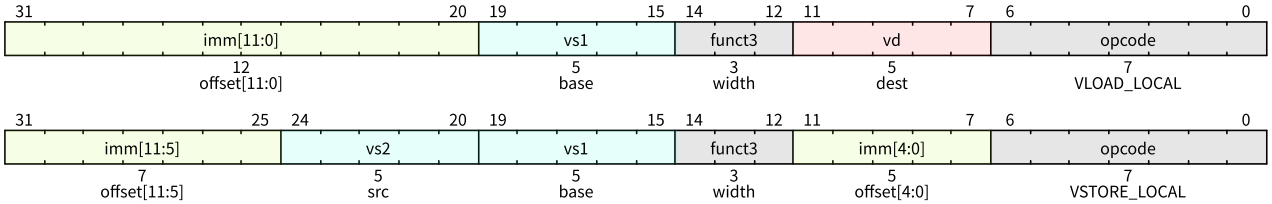
每个线程Addr计算方式同上，VSW12、VSH12和VSB12指令分别将向量寄存器vs2的低位中的32位、16位和8位值的向量长度个数据存入以Addr起始的内存空间。

global访存方式是用于访存全局内存的方式。
自定义global访存指令的地址空间按字节寻址且为小端格式。每个线程有效字节地址为Addr，将内存中以Addr为起始的元素复制到向量寄存器vd。VLW_GLOBAL指令从全局内存中加载32位值的向量到vd中。VLH_GLOBAL从全局内存中加载16位值，然后将其符号扩展到32位，再存储到vd中。VLHU_GLOBAL从全局内存中加载16位值，然后将其无符号扩展到32位，再存储到vd中。VLB_GLOBAL和VLBU_GLOBAL对8位值有类似定义。其中，Addr = vs1 + offset + csr_gds.



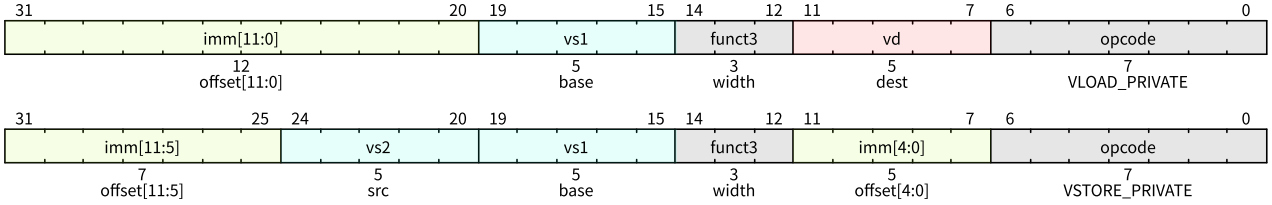
每个线程Addr计算方式同上，VSW_GLOBAL、VSH_GLOBAL和VSB_GLOBAL指令分别将向量寄存器vs2的低位中的32位、16位和8位值的向量长度个数据存入以Addr起始的全局内存空间。

local访存方式是用于访存共享内存的方式。
自定义local访存指令的地址空间按字节寻址且为小端格式。每个线程有效字节地址为Addr，将内存中以Addr为起始的元素复制到向量寄存器vd。VLW_LOCAL指令从共享内存中加载32位值的向量到vd中。VLH_LOCAL从共享内存中加载16位值，然后将其符号扩展到32位，再存储到vd中。VLHU_LOCAL从共享内存中加载16位值，然后将其无符号扩展到32位，再存储到vd中。VLB_LOCAL和VLBU_LOCAL对8位值有类似定义。其中，Addr = vs1 + offset + csr_lds.



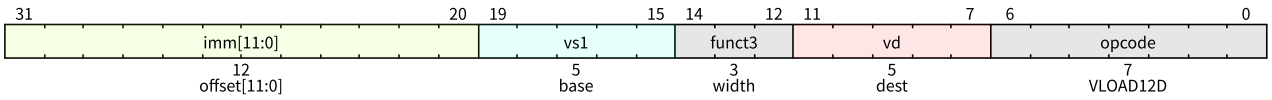
每个线程Addr计算方式同上，VSW_LOCAL、VSH_LOCAL和VSB_LOCAL指令分别将向量寄存器vs2的低位中的32位、16位和8位值的向量长度个数据存入以Addr起始的共享内存空间。

private访存方式是用于访存私有内存的方式。
自定义PRIVATE访存指令的地址空间按字节寻址且为小端格式。每个线程有效字节地址为Addr，将内存中以Addr为起始的元素复制到向量寄存器vd。VLW_PRIVATE指令从私有内存中加载32位值的向量到vd中。VLH_PRIVATE从私有内存中加载16位值，然后将其符号扩展到32位，再存储到vd中。VLHU_PRIVATE从私有内存中加载16位值，然后将其无符号扩展到32位，再存储到vd中。VLB_PRIVATE和VLBU_PRIVATE对8位值有类似定义。其中，Addr =((vs1+offset)[31:2]<<2) * num_thread_per_warp + (vs1+offset)[1:0] + thread_idx_in_warp<<2 + csr_pds[wid].

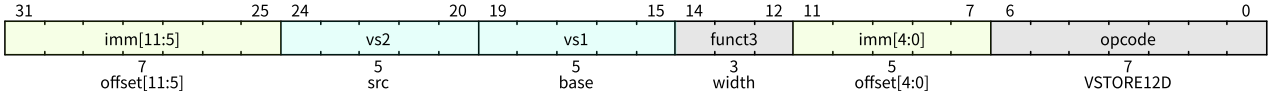


每个线程Addr计算方式同上，VSW_PRIVATE、VSH_PRIVATE和VSB_PRIVATE指令分别将向量寄存器vs2的低位中的32位、16位和8位值的向量长度个数据存入以Addr起始的私有内存空间。

自定义64位地址空间立即数访存指令



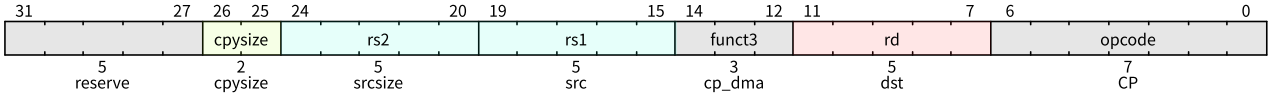
自定义64位访存指令的地址空间按字节寻址且为小端格式，vs1表示偶对齐的寄存器对。每个线程有效字节地址addr=[vs1+1:vs1]+offset，将内存中以addr为起始的元素复制到向量寄存器vd。VLW12D指令从内存中加载32位值的向量到vd中。VLH12D从内存中加载16位值，然后将其符号扩展到32位，再存储到vd中。VLHU12D从内存中加载16位值，然后将其无符号扩展到32位，再存储到vd中。VLB12和VLBU12对8位值有类似定义。



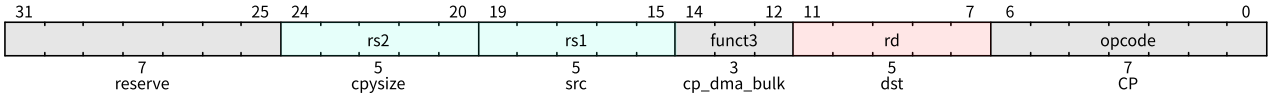
每个线程addr=[vs1+1:vs1]+offset，VSW12D、VSH12D和VSB12D指令分别将向量寄存器vs2的低位中的32位、16位和8位值的向量长度个数据存入以addr起始的内存空间。

自定义异步数据拷贝指令

cp_dma指令指定了数据在二级缓存中的源地址、在共享内存的目的地址、以及拷贝规模 (copysize) 和源规模 (srcsize)。指令中的数据地址对齐到1字节。拷贝规模指定的是拷贝数据的总量，在这条指令中规定拷贝的数据量只能是4、8、16字节。源规模指定的是拷贝数据中有效数据的字节数，因此源规模不能大于拷贝规模。拷贝规模大于源规模的部分数据用0替换。



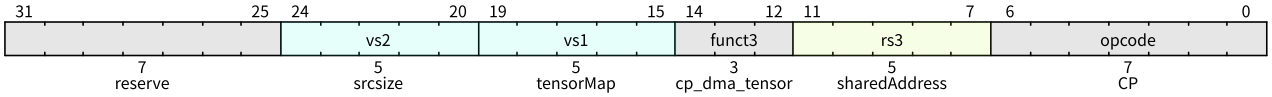
cp_dma_bulk指令用于大规模数据搬运。指令规定源地址、目的地址和拷贝规模都对齐到16字节。指令读取3个标量寄存器中保存的拷贝规模、源地址和目的地址，进行异步大规模数据拷贝。拷贝的规模范围是4到2^{32} – 4字节。



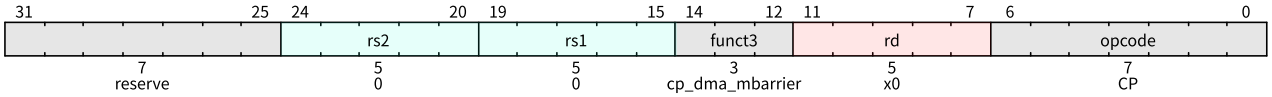
cp_dma_tensor指令用于从全局内存中将一个张量类型的数据拷贝到共享内存中。寻址box和张量需要的参数很多，指令执行时认为这些参数已经被保存在向量寄存器中。寄存器中每个位置是一个xLen比特数，可以直接读
取。参数如下表所示：

数据信息	信息功能	保存位置
datatype	张量数据类型及宽度	vs1[0]
tensorRank	张量维度，不大于5	vs1[1]
globalAddress	全局内存中的张量起始地址	vs1[2]
globalDim1-5	维度1-5张量元素数量，不为零且小于等于2^{32}	vs1[3]-vs1[7]

数据信息	信息功能	保存位置
globalStride1-5	维度1-5步长，必须为16倍数且小于2^{40}	vs1[8]-vs1[12]
BoxAddress	Box起始地址	vs2[0]
boxDim1-5	沿维度1-5遍历的元素数量，不为零且小于等于256	vs2[1]-vs2[5]
elementStrides1-5	沿维度1-5的迭代步长，不为零且小于等于8	vs2[6]-vs2[10]
interleave	数据交织模式	vs2[11]
swizzle	共享内存中的swizzling模式	vs2[12]
L2promotion	二级缓存字节粒度	vs2[13]
oobfill	指定用0还是NaN元素来填充越界元素	vs2[14]
sharedAddress	共享内存中存储起始地址	rs3



cp_dma_mbarrier用于异步拷贝指令同步，功能为：让线程束执行到这条指令且DMA完成数据搬运后，再执行下面的指令,同时将线程束每4个分为一组，只有一个组内当4个线程束全部执行到栅栏指令和DMA数据搬运完成，这个组才能执行后面的指令。



自定义访存前缀指令

PREFIX_MEMORY系列指令将作为前缀指令指示后续访存指令的访存空间、缓存策略、读取数据宽度，若pair字段为0，则地址位宽（32/64）将根据其接续访存指令所指定的地址位宽进行访存，若为1，则按照64位地址空间访存，若其后一条指令非访存指令，则当前指令失效。

高12位的立即数被分拆为四段3位数，分别拼接在下一条指令中寄存器rs3/vs3、rs2/vs2、rs1/vs1、rd/vd编码的高位上。

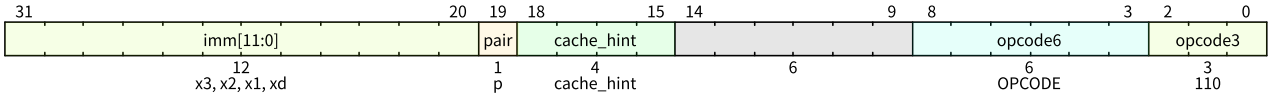
PREFIX_MEMORY指令指示的地址空间及映射关系如下：

地址空间	地址计算	编码
private	Addr=((vs1+offset)[31:2]<<2) * num_thread_per_warp + (vs1+offset)[1:0] + thread_idx_in_warp<<2 + csr_pds[wid] 或者 Addr=(([vs1:vs1+1]+offset)[31:2]<<2) * num_thread_per_warp + ([vs1:vs1+1]+offset)[1:0] + thread_idx_in_warp<<2 + csr_pds[wid]	2
local	addr=(vs1+imm)+csr_lds 或者 addr=([vs1:vs1+1]+imm)+csr_lds	3
global	addr=(vs1+imm)+csr_gds 或者 addr=([vs1:vs1+1]+imm)+csr_gds	1
default	根据访存指令定义进行地址计算	0

OPCODE4字段高2bit将作为地址空间编码，低2bit作为数据宽度编码，若存取数据宽度大于32bit（1word），则从连续的地址空间读取数据存入相邻寄存器，或从相邻寄存器读出数据存入连续地址空间。

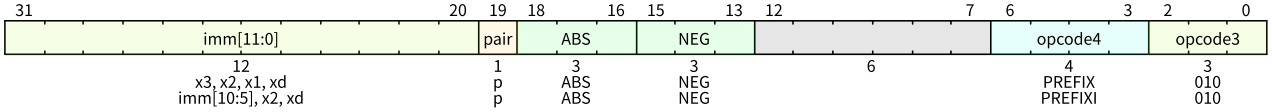
数据宽度（bit）	编码
32	0
64	1
96	2
128	3

指令的cachehint字段将作为硬件缓存策略指导，但不一定生效。



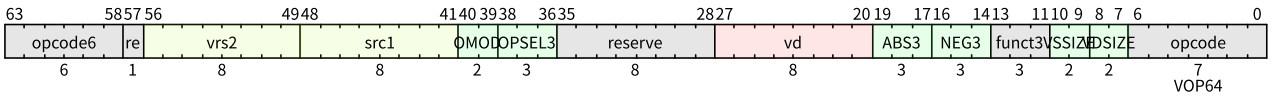
自定义计算前缀指令

PREFIX和PREFIXI指令用于扩展它之后一条指令的寄存器编码及立即数编码，并对下一条指令的计算结果进行modified。在PREFIX指令中，12位的立即数被分拆为四段3位数，分别拼接在下一条指令中寄存器rs3/vs3、rs2/vs2、rs1/vs1、rd/vd编码的高位上。PREFIXI指令则面向使用了5位立即数的指令，前缀包含6位的立即数高位，以及rs2/vs2、rd/vd编码的高位。11立即数由前缀指令中的立即数高位与5位立即数直接拼接得到。
此外，在不使用扩展的情况下，向量浮点运算的vs3就是vd的[11:7]段，但进行向量扩展时，vs3和vd的高3位分离存储。



字段	含义
pair	下一条指令的是否采用寄存器对拼接的64位数据，0-不采用，1-采用
ABS	下一条指令的输入数据是否取绝对值，比特0指示src1，比特1指示src2，比特2指示src3，0-不取绝对值，1-取绝对值
NEG	下一条指令的输入数据是否取反，比特0指示src1，比特1指示src2，比特2指示src3，0-不取反，1-取反

自定义64位计算指令



指令字长为64bit的计算指令，支持整型和浮点的加、减、乘、max、min的操作，主要用于扩展fp16、int8等数据类型的计算，并且通过修饰符支持操作数的取反、取绝对值、圆整、乘系数等操作。
其中各字段描述如下：

NEG：3bit位宽，分别指示3个src（现阶段常规计算指令只有2个操作数，预留1bit给后续扩展）

ABS：3bit位宽OPSEL：3bit位宽，分别指示src1，src2，dst的高低16位

OMOD：0-无系数，1-结果乘2，2-结果乘4，3-结果除以2

VSSIZE：0-16bit，1-32bit，2-64bit，3-保留，源操作数位宽

VDSIZE：0-16bit，1-32bit，2-64bit，3-保留，目的操作数位宽

OPSEL：比特选择，共有3位，分别对应vs2、src1、vd的选择，为0时每个元素低16比特有效，为1时高16比特有效，计算数据宽度大于等于32比特时字段无意义。

指令的funct3和opcode6字段对齐vector的定义，分别表示数据形式（整型或浮点）和运算功能（加、减、乘、max、min）。指令目前保留了9比特保留位，可用于扩展三操作数的计算指令。

RV64I

RV64I相较于RV32I新增的指令及语义总结如下

指令	语义
SLLW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SLLW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SRLW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SRAIW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SRAW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SRAIW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
ADDW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
ADDIW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
SUBW	源操作数与目的操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐
LD	源操作数为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐， 从64位地址空间取出32位数据存入目的寄存器
SD	地址操作数rs1为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐，将rs2中32位数据存入64位地址空间

RV64A

RV64A相较于RV32A新增的指令及语义总结如下：

指令集	语义
RV64A	地址操作数rs1为相邻寄存器对拼接产生的64比特数据，寄存器对拼接偶对齐，rs2及rd为32比特数据

RVV widening

指令范围	语义	是否与RVV定义一致
vwop.vv	源操作数为32比特向量元素，目的操作数为64比特向量元素，64 = 32 op 32	是
vwop.vx	源操作数分别为32比特向量元素和32比特标量元素，目的操作数为64比特向量元素，64 = 32 op 32	是
vwop.wv	源操作数均为64比特向量元素，目的操作数为64比特向量元素，64 = 64 op 64	否

指令范围	语义	是否与RVV定义一致
vwop.wx	源操作数分别为64比特向量元素和32比特标量元素，目的操作数为64比特向量元素，64 = 64 op 64	否