软件所智能软件中心PLCT实验室 王鹏 实习生

2020//04/15

目 录

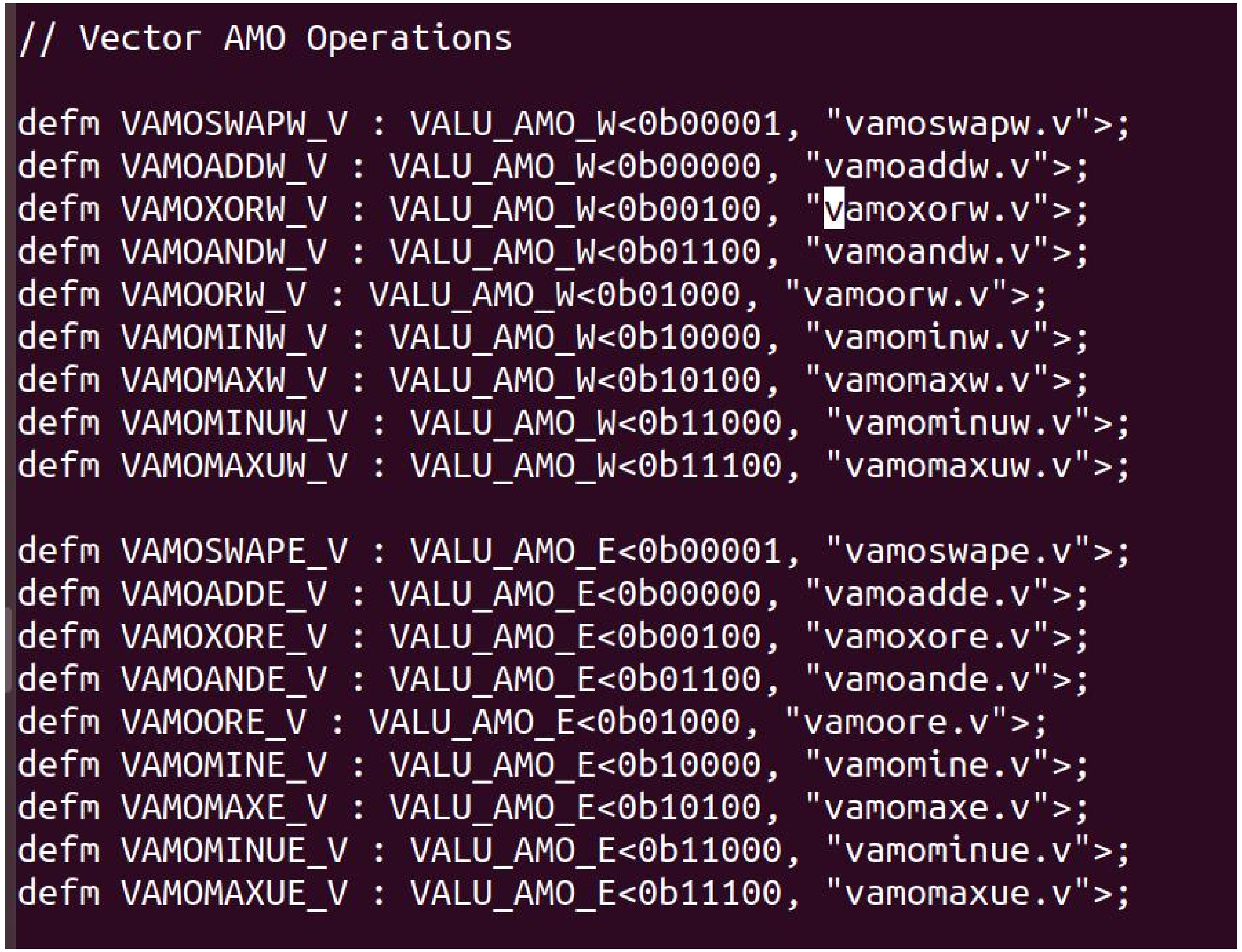
• 01 玄铁C910中vector和rvv对比

011 VAMO指令

（玄铁C910中Vector指令5.7.10 - 5.7.36），riscv-v-spec 0.7.1中Table 13定义了amoop，但是具体的指令格式都没有定义，vamo*所用的Inst(6-0)= 0101111，拿原子指令AMO没用完的部分定义的。本课件中的所有代码截图都来自rvv-llvm源代码。

Table 13. amoop

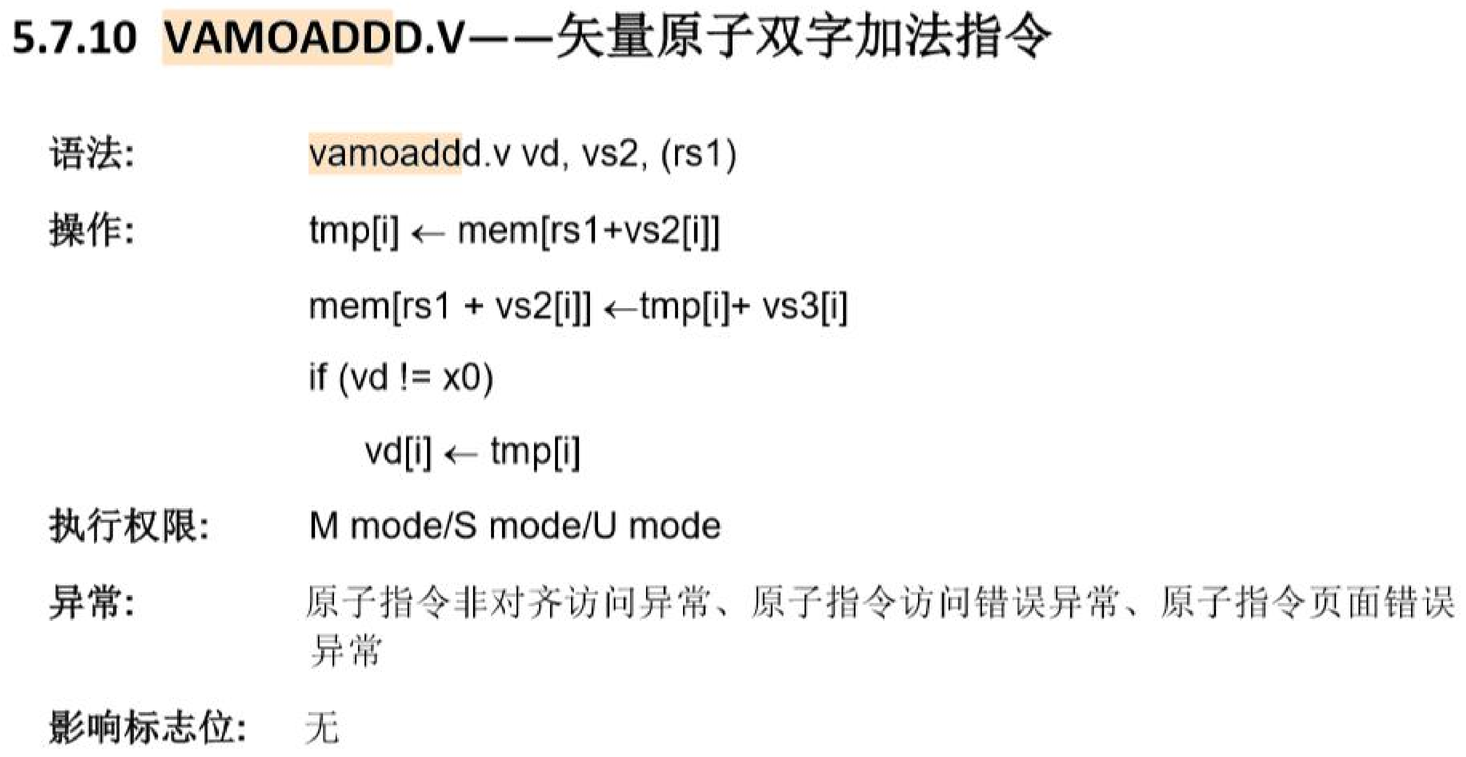
amoop					opcode
0	0	0	0	1	vamoswap
0	0	0	0	0	vamoadd
0	0	1	0	0	vamoxor
0	1	1	0	0	vamoand
0	1	0	0	0	vamoor
1	0	0	0	0	vamomin
1	0	1	0	0	vamomax
1	1	0	0	0	vamominu
1	1	1	0	0	vamomaxu



```
// Vector AMO Operations

defm VAMOSWAPW_V : VALU_AMO_W<0b00001, "vamoswapw.v">;
defm VAMOADDW_V : VALU_AMO_W<0b00000, "vamoaddw.v">;
defm VAMOXORW_V : VALU_AMO_W<0b00100, "vamoxorw.v">;
defm VAMOANDW_V : VALU_AMO_W<0b01100, "vamoandw.v">;
defm VAMOORW_V : VALU_AMO_W<0b01000, "vamoorw.v">;
defm VAMOMINW_V : VALU_AMO_W<0b10000, "vamominw.v">;
defm VAMOMAXW_V : VALU_AMO_W<0b10100, "vamomaxw.v">;
defm VAMOMINUW_V : VALU_AMO_W<0b11000, "vamominuw.v">;
defm VAMOMAXUW_V : VALU_AMO_W<0b11100, "vamomaxuw.v">;

defm VAMOSWAPE_V : VALU_AMO_E<0b00001, "vamoswape.v">;
defm VAMOADDE_V : VALU_AMO_E<0b00000, "vamoadde.v">;
defm VAMOXORE_V : VALU_AMO_E<0b00100, "vamoxore.v">;
defm VAMOANDE_V : VALU_AMO_E<0b01100, "vamoande.v">;
defm VAMOORE_V : VALU_AMO_E<0b01000, "vamoore.v">;
defm VAMOMINE_V : VALU_AMO_E<0b10000, "vamomine.v">;
defm VAMOMAXE_V : VALU_AMO_E<0b10100, "vamomaxe.v">;
defm VAMOMINUE_V : VALU_AMO_E<0b11000, "vamominue.v">;
defm VAMOMAXUE_V : VALU_AMO_E<0b11100, "vamomaxue.v">;
```

rvv-llvm

RISCVInstrInfoV.td



5.7.10 VAMOADDD.V——矢量原子双字加法指令

语法:	vamoaddd.v vd, vs2, (rs1)

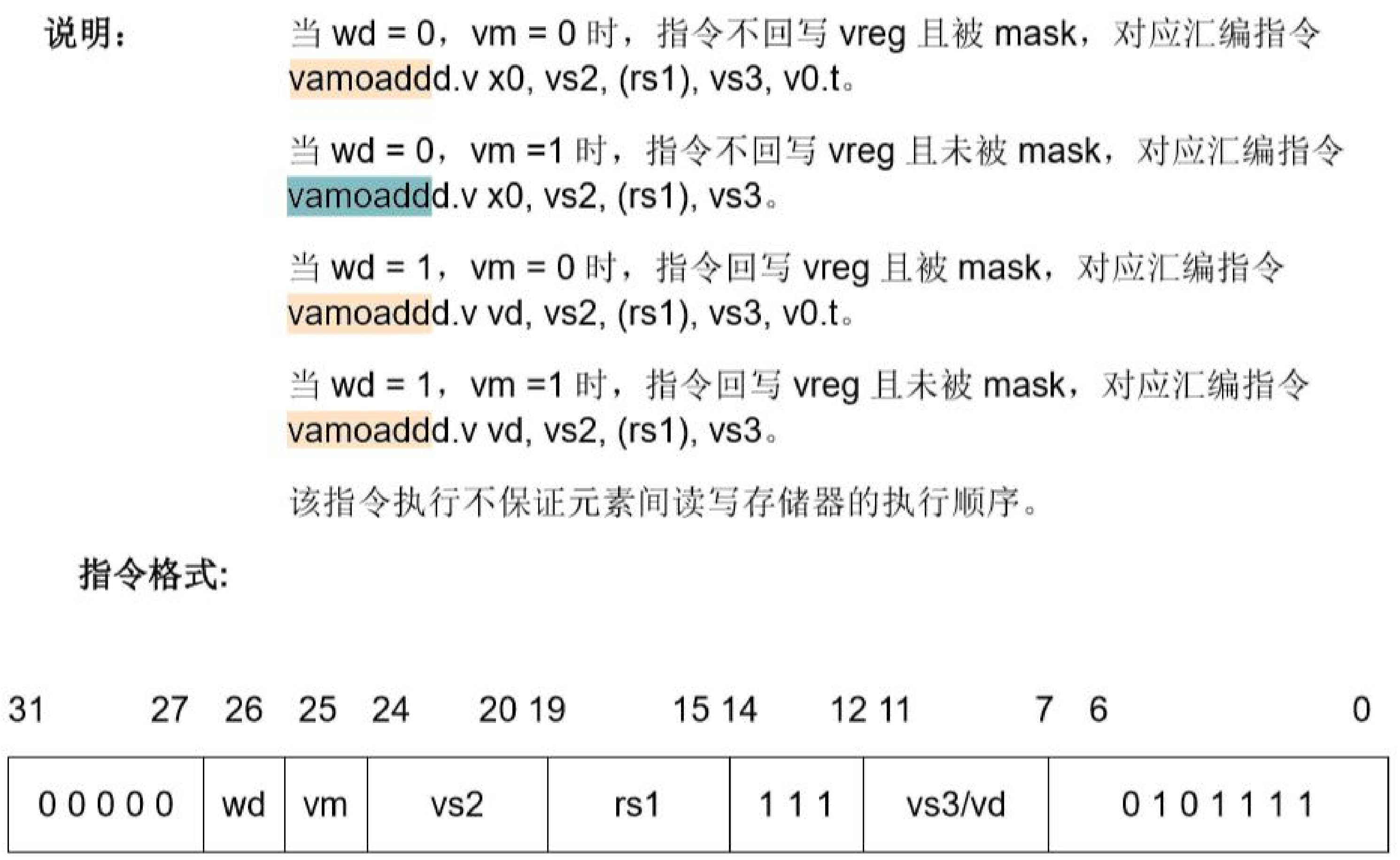
操作:	tmp[i] ← mem[rs1+vs2[i]]
	mem[rs1 + vs2[i]] ←tmp[i]+ vs3[i]
	if (vd != x0)
	    vd[i] ← tmp[i]

执行权限:	M mode/S mode/U mode
异常:	原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常
影响标志位:	无

C910矢量指令集

VAMOADDD的Inst(14-12)=0b111

中国科学院软件研究所 Institute of Software Chinese Academy of Sciences
智能软件研究中心 Intelligent Software Research Center



说明：

当 wd = 0， vm = 0 时，指令不回写 vreg 且被 mask，对应汇编指令 vamoaddd.v x0, vs2, (rs1), vs3, v0.t。

当 wd = 0， vm =1 时，指令不回写 vreg 且未被 mask，对应汇编指令 vamoaddd.v x0, vs2, (rs1), vs3。

当 wd = 1， vm = 0 时，指令回写 vreg 且被 mask，对应汇编指令 vamoaddd.v vd, vs2, (rs1), vs3, v0.t。

当 wd = 1， vm =1 时，指令回写 vreg 且未被 mask，对应汇编指令 vamoaddd.v vd, vs2, (rs1), vs3。

该指令执行不保证元素间读写存储器的执行顺序。

指令格式:

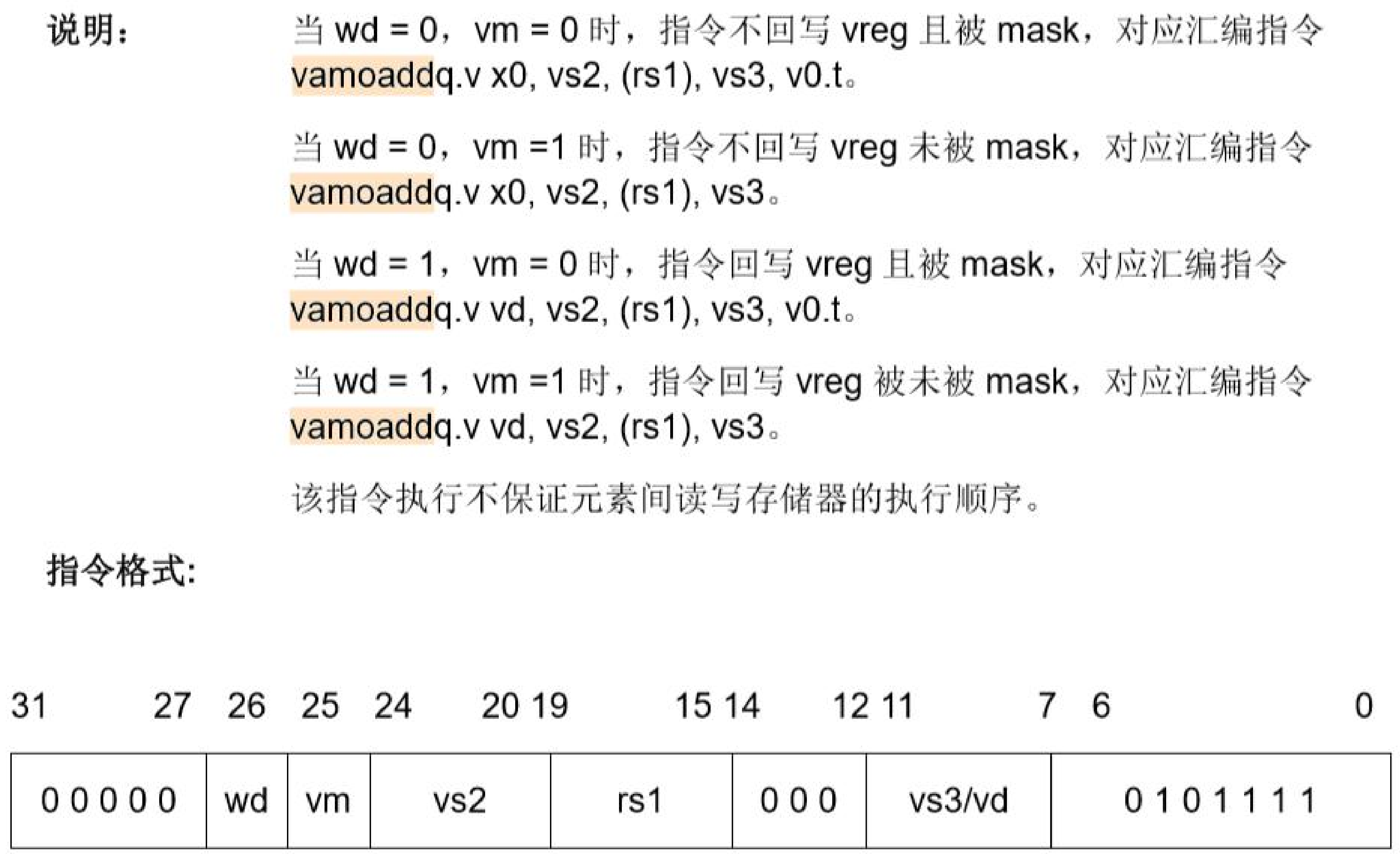
31 27	26	25	24 20	19 15	14 12	11 7	6 0
0 0 0 0 0	wd	vm	vs2	rs1	1 1 1	vs3/vd	0 1 0 1 1 1 1

C910矢量指令集中W/D/Q的Inst(31-27)=0b00000都一样

VAMOADDW的Inst(14-12)=0b110

5.7.11 VAMOADDQ.V——矢量原子四字加法指令

语法:	vamoaddq.v vd, vs2, (rs1), vs3
操作:	tmp[i] ← mem[rs1+vs2[i]]
	mem[rs1 + vs2[i]] ←tmp[i]+ vs3[i]
	if (vd != x0)
	vd[i] ← tmp[i]
执行权限:	M mode/S mode/U mode
异常:	原子指令非对齐访问异常、原子指令访问错误异常、原子指令页面错误异常
影响标志位:	无



说明：

当 wd = 0，vm = 0 时，指令不回写 vreg 且被 mask，对应汇编指令 vamoaddq.v x0, vs2, (rs1), vs3, v0.t。

当 wd = 0，vm =1 时，指令不回写 vreg 未被 mask，对应汇编指令 vamoaddq.v x0, vs2, (rs1), vs3。

当 wd = 1，vm = 0 时，指令回写 vreg 且被 mask，对应汇编指令 vamoaddq.v vd, vs2, (rs1), vs3, v0.t。

当 wd = 1，vm =1 时，指令回写 vreg 被未被 mask，对应汇编指令 vamoaddq.v vd, vs2, (rs1), vs3。

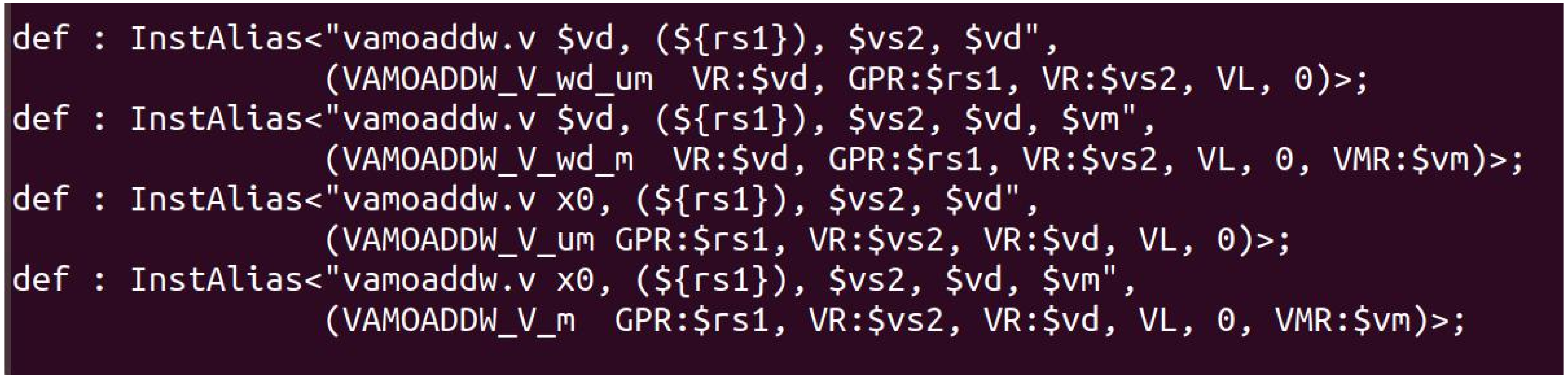
该指令执行不保证元素间读写存储器的执行顺序。

指令格式:

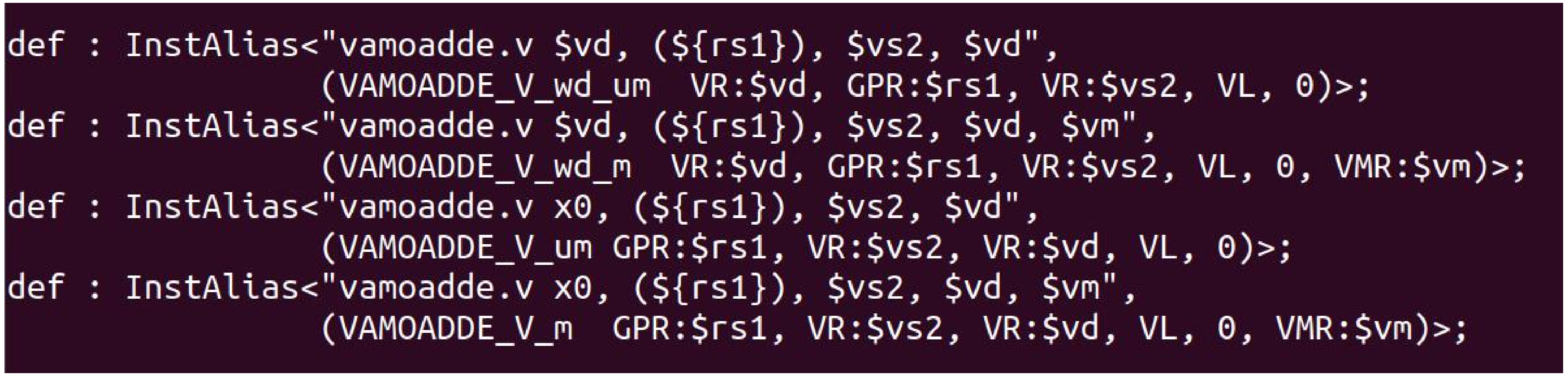
31 27	26	25	24 20	19 15	14 12	11 7	6 0
0 0 0 0 0	wd	vm	vs2	rs1	0 0 0	vs3/vd	0 1 0 1 1 1 1

C910矢量指令集

VAMOADDQ的Inst(14-12)=0b000

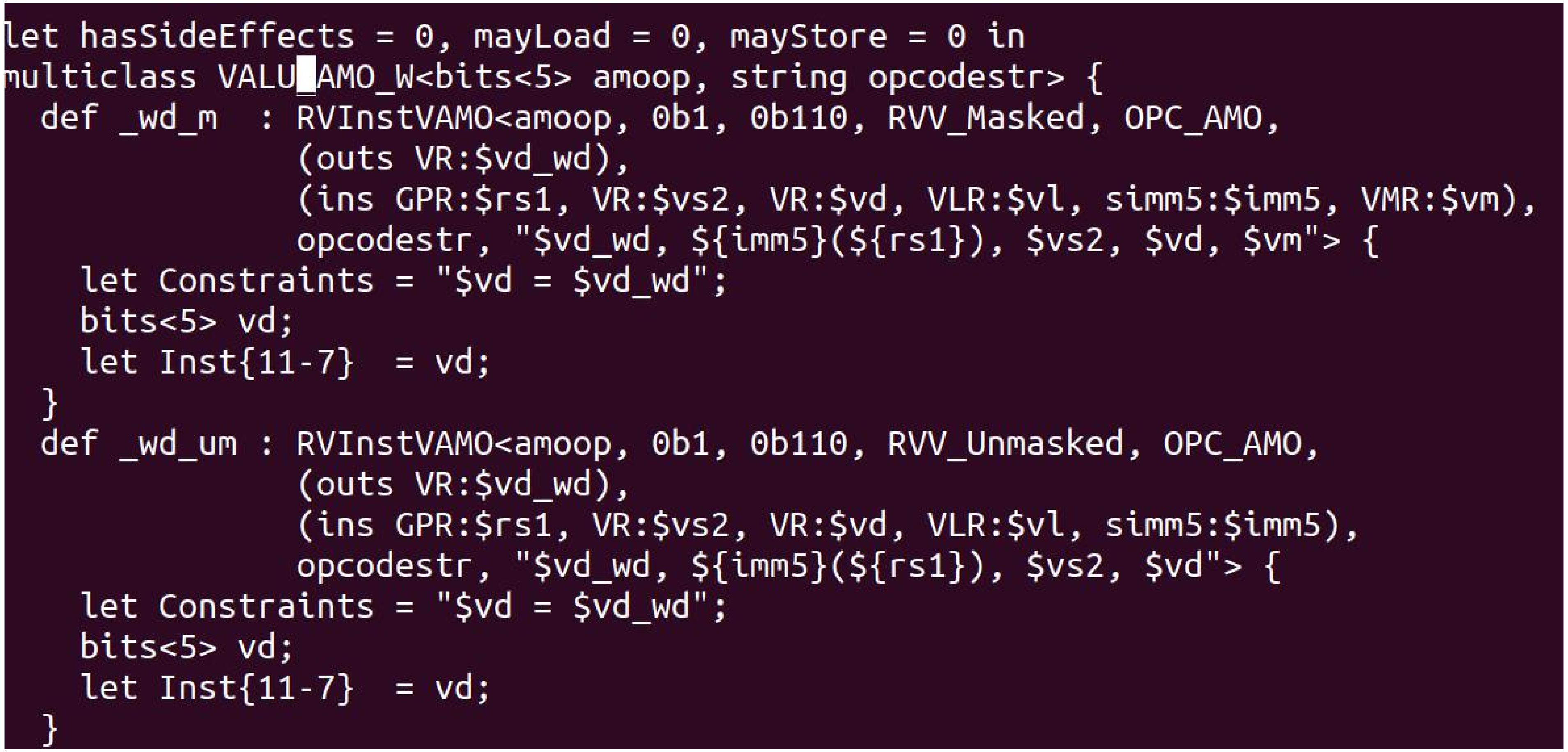


```
def : InstAlias<"vamoaddw.v $vd, (${rs1}), $vs2, $vd",
                (VAMOADDW_V_wd_um  VR:$vd, GPR:$rs1, VR:$vs2, VL, 0)>;
def : InstAlias<"vamoaddw.v $vd, (${rs1}), $vs2, $vd, $vm",
                (VAMOADDW_V_wd_m  VR:$vd, GPR:$rs1, VR:$vs2, VL, 0, VMR:$vm)>;
def : InstAlias<"vamoaddw.v x0, (${rs1}), $vs2, $vd",
                (VAMOADDW_V_um GPR:$rs1, VR:$vs2, VR:$vd, VL, 0)>;
def : InstAlias<"vamoaddw.v x0, (${rs1}), $vs2, $vd, $vm",
                (VAMOADDW_V_m  GPR:$rs1, VR:$vs2, VR:$vd, VL, 0, VMR:$vm)>;
```



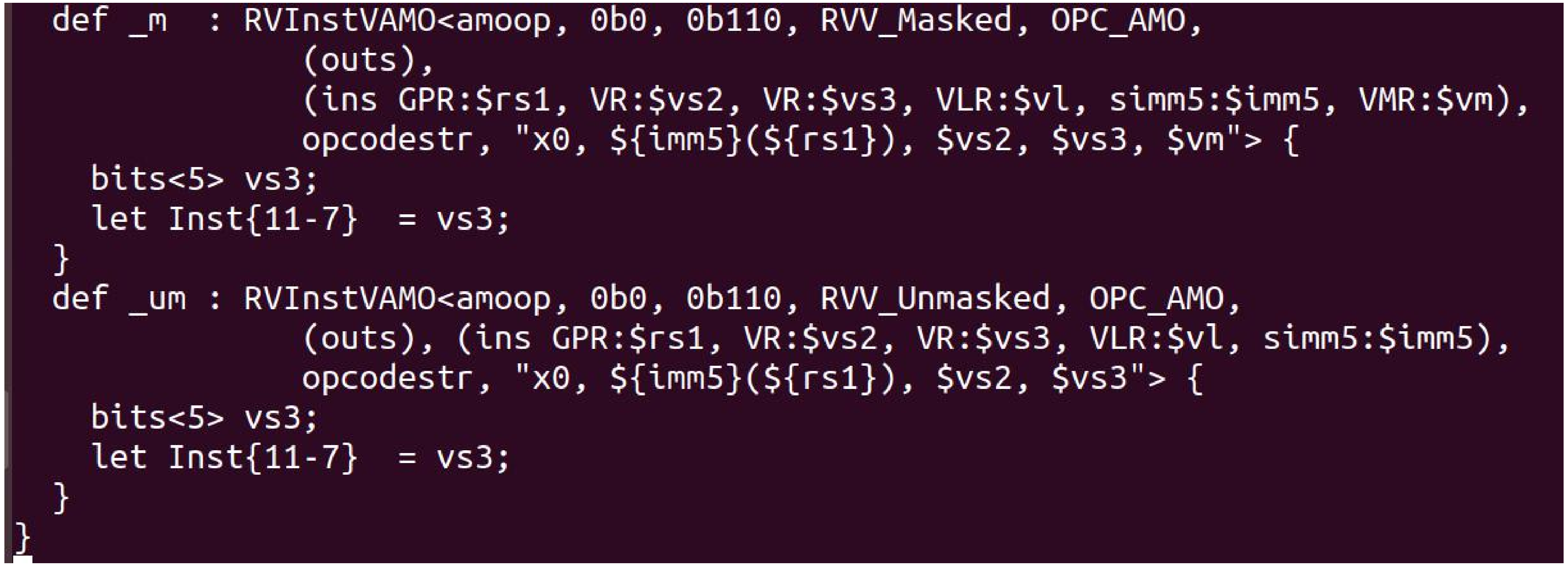
```
def : InstAlias<"vamoadde.v $vd, (${rs1}), $vs2, $vd",
                (VAMOADDE_V_wd_um  VR:$vd, GPR:$rs1, VR:$vs2, VL, 0)>;
def : InstAlias<"vamoadde.v $vd, (${rs1}), $vs2, $vd, $vm",
                (VAMOADDE_V_wd_m  VR:$vd, GPR:$rs1, VR:$vs2, VL, 0, VMR:$vm)>;
def : InstAlias<"vamoadde.v x0, (${rs1}), $vs2, $vd",
                (VAMOADDE_V_um GPR:$rs1, VR:$vs2, VR:$vd, VL, 0)>;
def : InstAlias<"vamoadde.v x0, (${rs1}), $vs2, $vd, $vm",
                (VAMOADDE_V_m  GPR:$rs1, VR:$vs2, VR:$vd, VL, 0, VMR:$vm)>;
```

RISCVInstrInfoV.td中VAMOADDW/E的Inst(31-27)=0b00000都一样



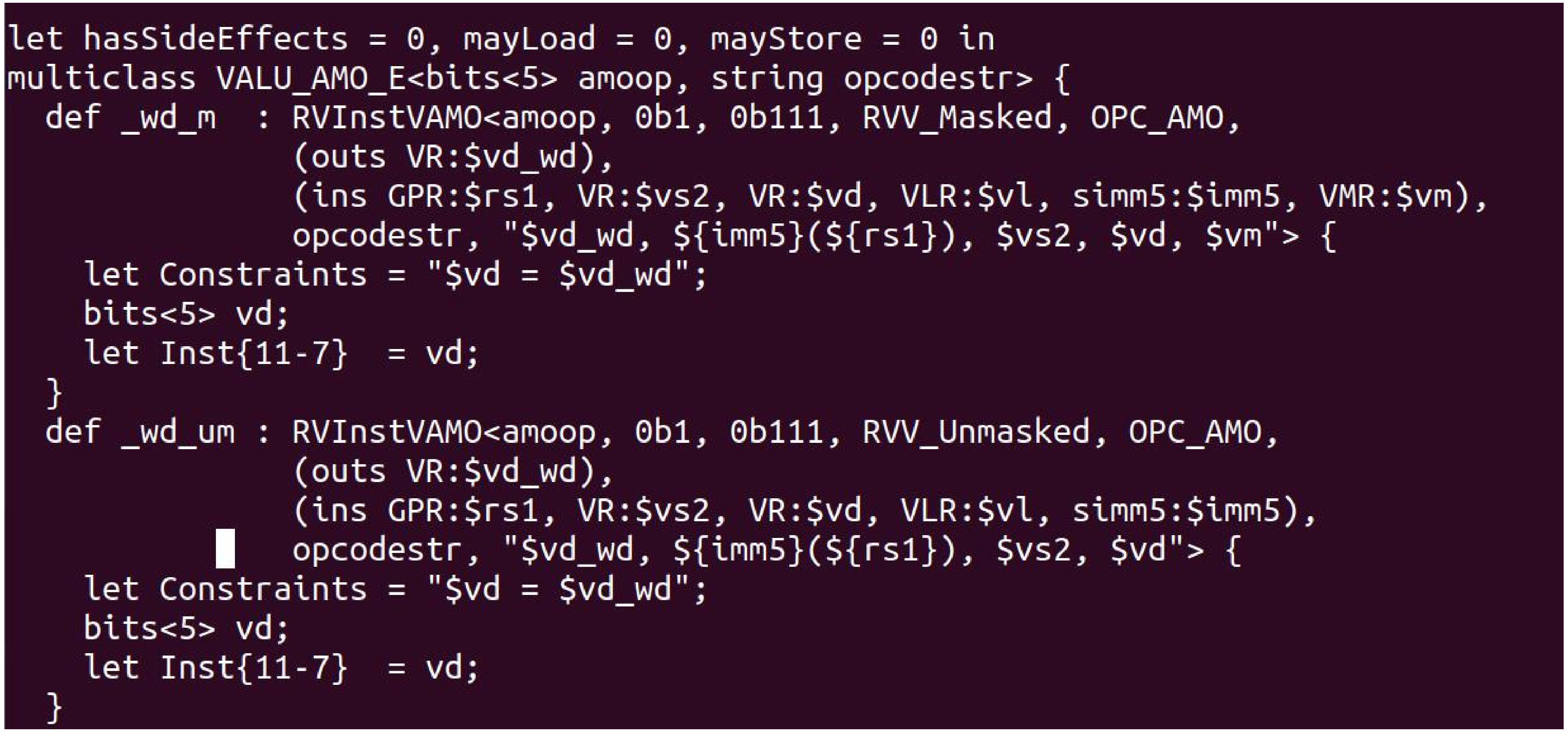
```
let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
multiclass VALU AMO_W<bits<5> amoop, string opcodestr> {
  def _wd_m  : RVInstVAMO<amoop, 0b1, 0b110, RVV_Masked, OPC_AMO,
             (outs VR:$vd_wd),
             (ins GPR:$rs1, VR:$vs2, VR:$vd, VLR:$vl, simm5:$imm5, VMR:$vm),
             opcodestr, "$vd_wd, ${imm5}(${rs1}), $vs2, $vd, $vm"> {
    let Constraints = "$vd = $vd_wd";
    bits<5> vd;
    let Inst{11-7}  = vd;
  }
  def _wd_um : RVInstVAMO<amoop, 0b1, 0b110, RVV_Unmasked, OPC_AMO,
             (outs VR:$vd_wd),
             (ins GPR:$rs1, VR:$vs2, VR:$vd, VLR:$vl, simm5:$imm5),
             opcodestr, "$vd_wd, ${imm5}(${rs1}), $vs2, $vd"> {
    let Constraints = "$vd = $vd_wd";
    bits<5> vd;
    let Inst{11-7}  = vd;
  }
```

RISCVInstrInfoV.td中分别定义了_wd_m，_wd_um，_m和_um这四种情况，分别对应于Inst(26-25)=wd，Inst(25-24)=vm



```
  def _m  : RVInstVAMO<amoop, 0b0, 0b110, RVV_Masked, OPC_AMO,
                 (outs),
                 (ins GPR:$rs1, VR:$vs2, VR:$vs3, VLR:$vl, simm5:$imm5, VMR:$vm),
                 opcodestr, "x0, ${imm5}(${rs1}), $vs2, $vs3, $vm"> {
    bits<5> vs3;
    let Inst{11-7}  = vs3;
  }
  def _um : RVInstVAMO<amoop, 0b0, 0b110, RVV_Unmasked, OPC_AMO,
                 (outs), (ins GPR:$rs1, VR:$vs2, VR:$vs3, VLR:$vl, simm5:$imm5),
                 opcodestr, "x0, ${imm5}(${rs1}), $vs2, $vs3"> {
    bits<5> vs3;
    let Inst{11-7}  = vs3;
  }
}
```

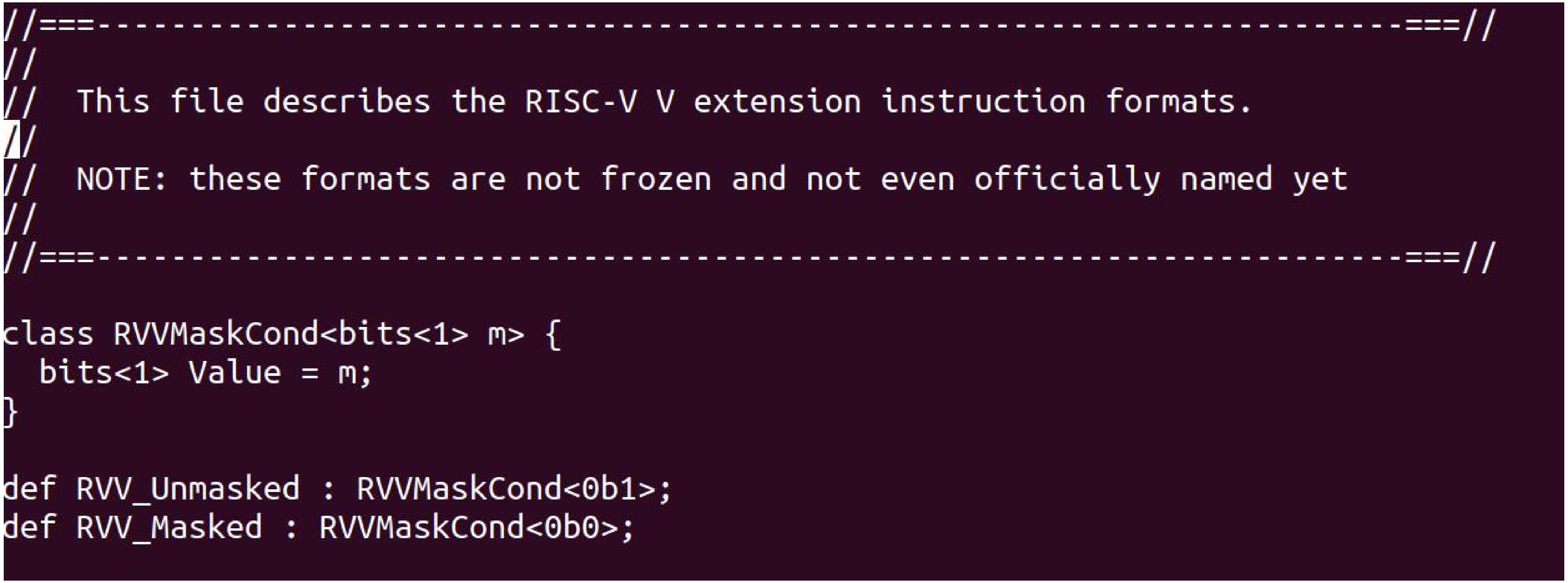
wd=1,指令回写vreg， vm=0指令被mask



```
let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
multiclass VALU_AMO_E<bits<5> amoop, string opcodestr> {
  def _wd_m  : RVInstVAMO<amoop, 0b1, 0b111, RVV_Masked, OPC_AMO,
               (outs VR:$vd_wd),
               (ins GPR:$rs1, VR:$vs2, VR:$vd, VLR:$vl, simm5:$imm5, VMR:$vm),
               opcodestr, "$vd_wd, ${imm5}(${rs1}), $vs2, $vd, $vm"> {
    let Constraints = "$vd = $vd_wd";
    bits<5> vd;
    let Inst{11-7}  = vd;
  }
  def _wd_um : RVInstVAMO<amoop, 0b1, 0b111, RVV_Unmasked, OPC_AMO,
               (outs VR:$vd_wd),
               (ins GPR:$rs1, VR:$vs2, VR:$vd, VLR:$vl, simm5:$imm5),
               opcodestr, "$vd_wd, ${imm5}(${rs1}), $vs2, $vd"> {
    let Constraints = "$vd = $vd_wd";
    bits<5> vd;
    let Inst{11-7}  = vd;
  }
```

RISCVInstrInfoV.td

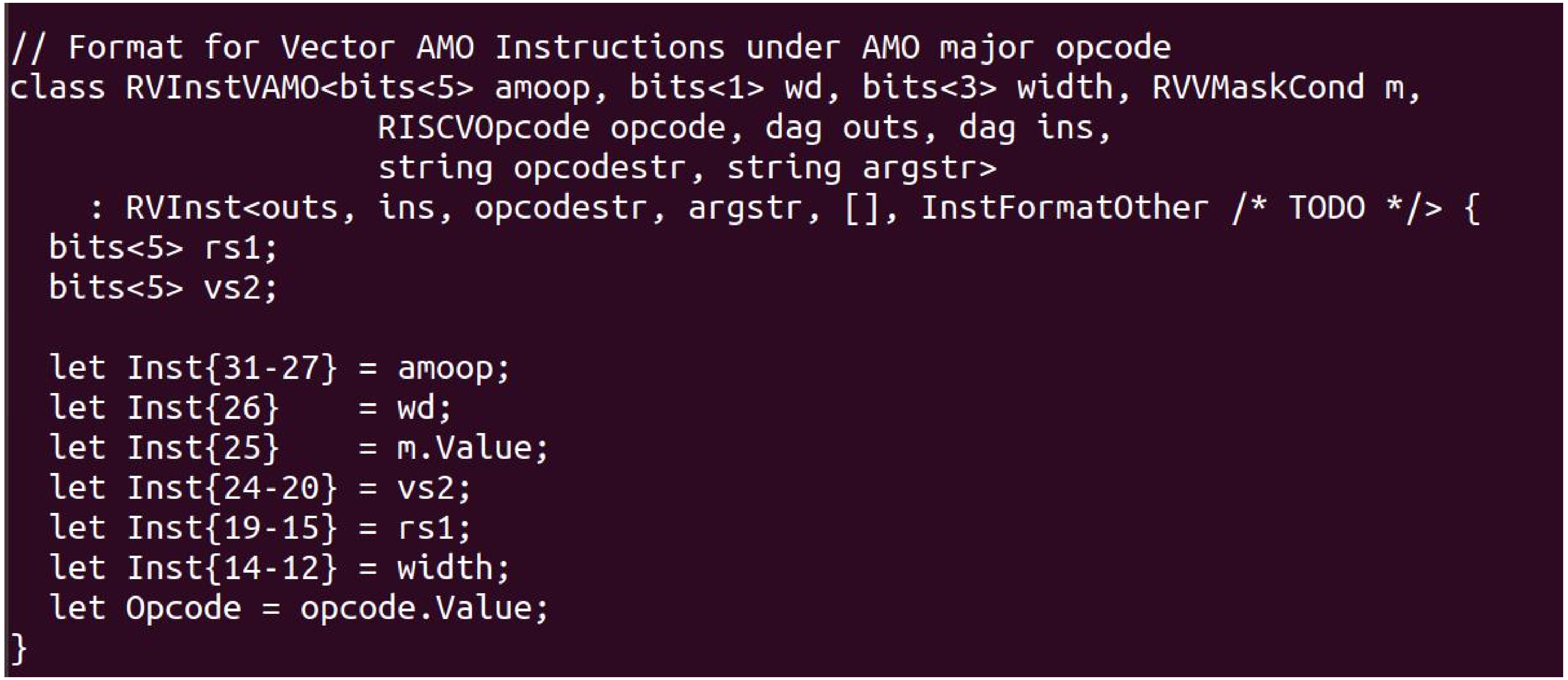
VALU_AMO_E中Inst(14-12)=0b111，其与玄铁C910中矢量原子双字加法指令重合



```
//===----------------------------------------------------------------===//
//
//  This file describes the RISC-V V extension instruction formats.
//
//  NOTE: these formats are not frozen and not even officially named yet
//
//===----------------------------------------------------------------===//

class RVVMaskCond<bits<1> m> {
  bits<1> Value = m;
}

def RVV_Unmasked : RVVMaskCond<0b1>;
def RVV_Masked : RVVMaskCond<0b0>;
```

mask在RISC-V V extension instruction formats中的定义

RISCVInstrFormatsV.td

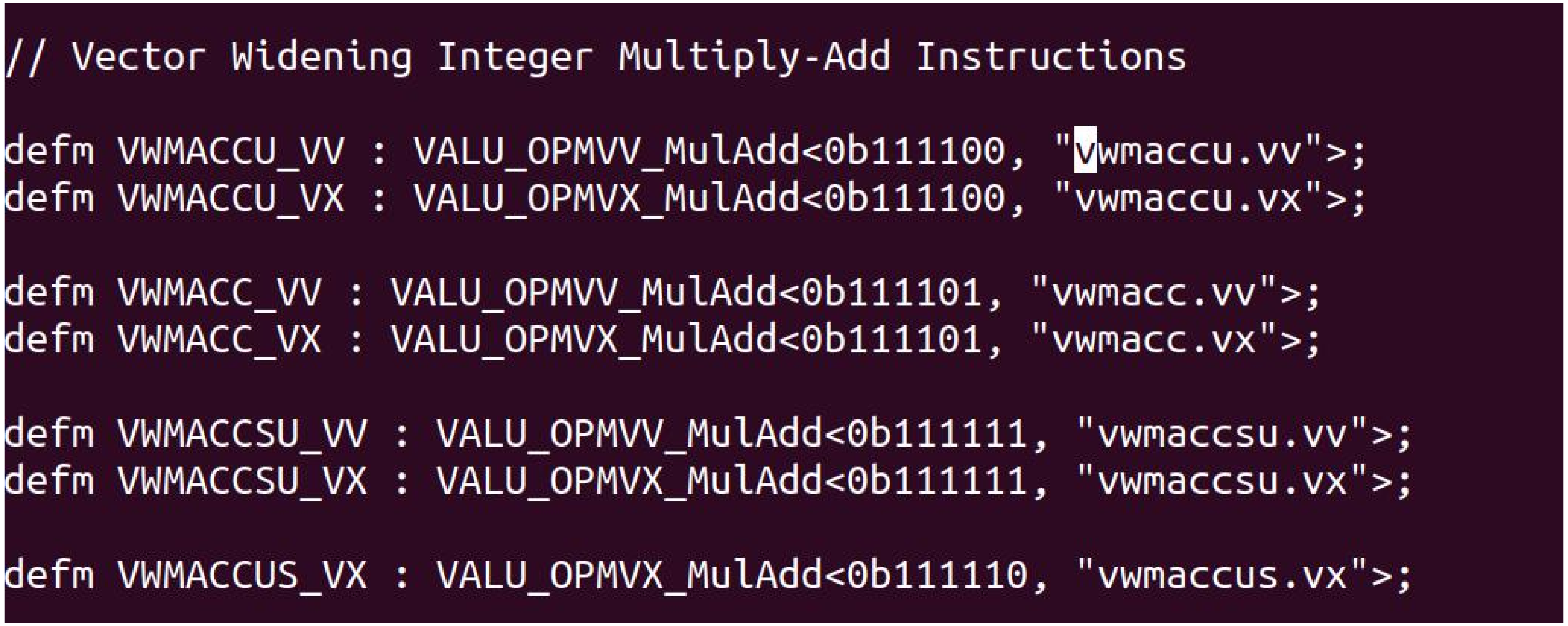


```
// Format for Vector AMO Instructions under AMO major opcode
class RVInstVAMO<bits<5> amoop, bits<1> wd, bits<3> width, RVVMaskCond m,
                   RISCVOpcode opcode, dag outs, dag ins,
                   string opcodestr, string argstr>
    : RVInst<outs, ins, opcodestr, argstr, [], InstFormatOther /* TODO */> {
  bits<5> rs1;
  bits<5> vs2;

  let Inst{31-27} = amoop;
  let Inst{26}    = wd;
  let Inst{25}    = m.Value;
  let Inst{24-20} = vs2;
  let Inst{19-15} = rs1;
  let Inst{14-12} = width;
  let Opcode = opcode.Value;
}
```

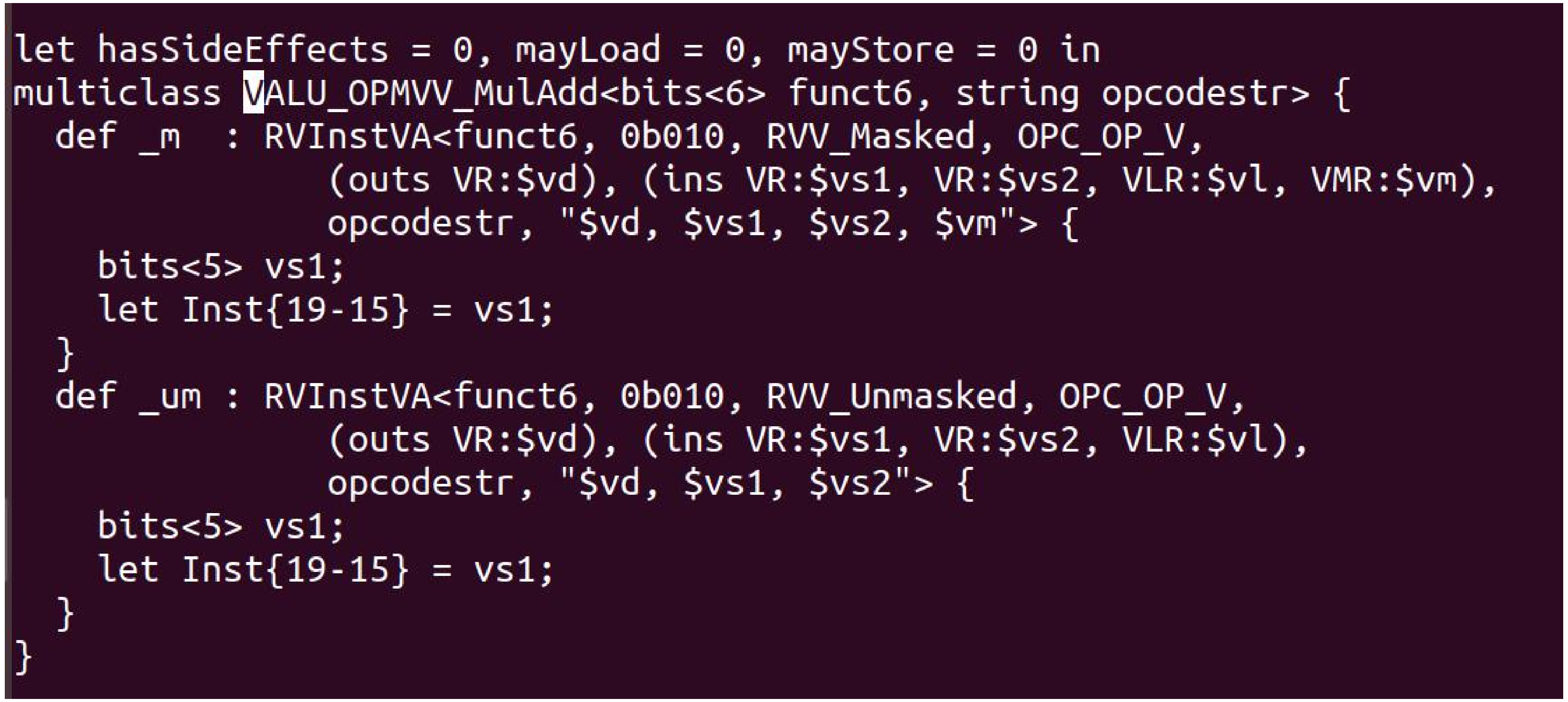
RISCVInstrFormatsV.td

012 VWSMACC*指令

5.7.346 VWSMACC.VV——矢量有符号带饱和扩位乘累加指令
5.7.347 VWSMACC.VX——矢量标量有符号带饱和扩位乘累加指令
5.7.348 VWSMACCSU.VV——矢量带饱和取反有符号无符号扩位乘累减指令
5.7.349 VWSMACCSU.VX——矢量标量带饱和取反有符号无符号扩位乘累减指令
5.7.350 VWSMACCU.VV——矢量无符号带饱和扩位乘累加指令
5.7.351 VWSMACCU.VX——矢量标量无符号带饱和扩位乘累加指令
5.7.352 VWSMACCUS.VX——矢量标量带饱和取反有符号无符号扩位乘累减指令

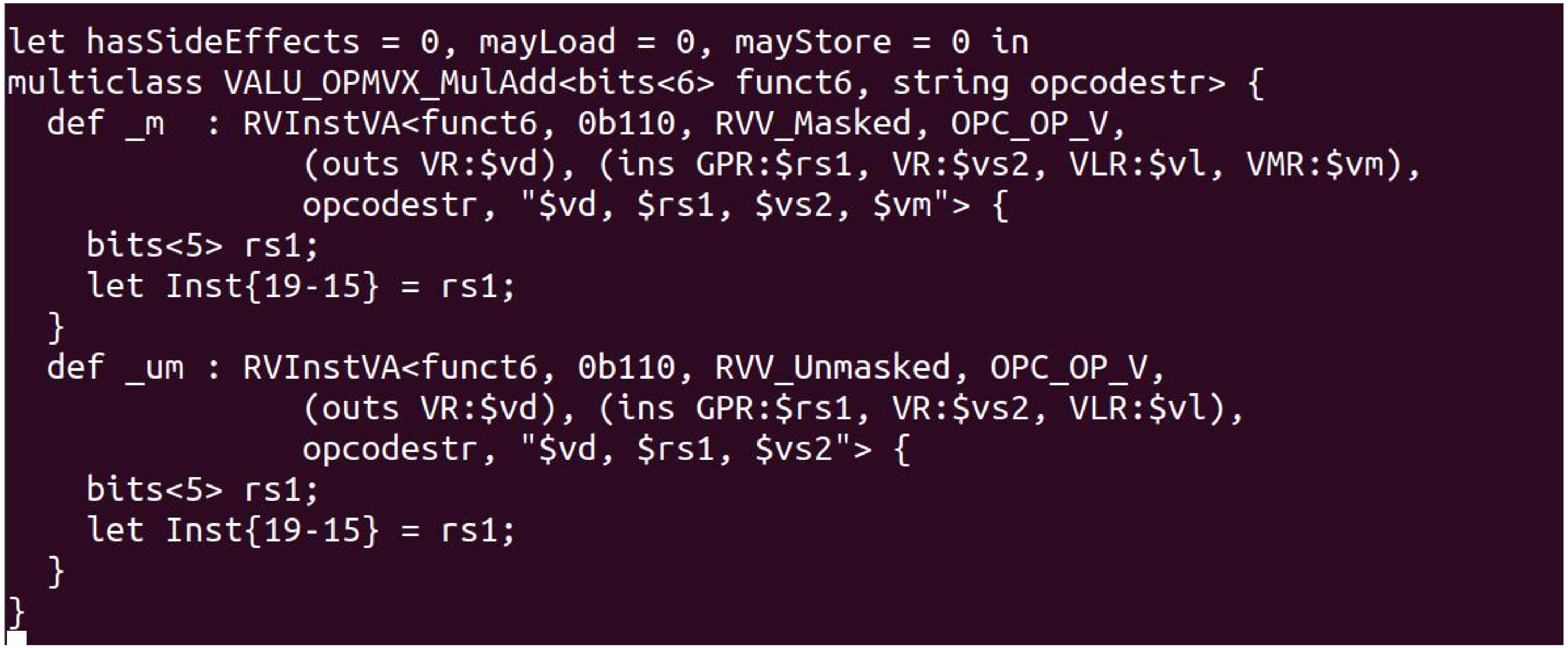

```
// Vector Widening Integer Multiply-Add Instructions

defm VWMACCU_VV : VALU_OPMVV_MulAdd<0b111100, "vwmaccu.vv">;
defm VWMACCU_VX : VALU_OPMVX_MulAdd<0b111100, "vwmaccu.vx">;

defm VWMACC_VV : VALU_OPMVV_MulAdd<0b111101, "vwmacc.vv">;
defm VWMACC_VX : VALU_OPMVX_MulAdd<0b111101, "vwmacc.vx">;

defm VWMACCSU_VV : VALU_OPMVV_MulAdd<0b111111, "vwmaccsu.vv">;
defm VWMACCSU_VX : VALU_OPMVX_MulAdd<0b111111, "vwmaccsu.vx">;

defm VWMACCUS_VX : VALU_OPMVX_MulAdd<0b111110, "vwmaccus.vx">;
```

RISCVInstrInfoV.td



```
let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
multiclass VALU_OPMVV_MulAdd<bits<6> funct6, string opcodestr> {
  def _m  : RVInstVA<funct6, 0b010, RVV_Masked, OPC_OP_V,
                (outs VR:$vd), (ins VR:$vs1, VR:$vs2, VLR:$vl, VMR:$vm),
                opcodestr, "$vd, $vs1, $vs2, $vm"> {
    bits<5> vs1;
    let Inst{19-15} = vs1;
  }
  def _um : RVInstVA<funct6, 0b010, RVV_Unmasked, OPC_OP_V,
                (outs VR:$vd), (ins VR:$vs1, VR:$vs2, VLR:$vl),
                opcodestr, "$vd, $vs1, $vs2"> {
    bits<5> vs1;
    let Inst{19-15} = vs1;
  }
}
```

RISCVInstrInfoV.td



```
let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
multiclass VALU_OPMVX_MulAdd<bits<6> funct6, string opcodestr> {
  def _m  : RVInstVA<funct6, 0b110, RVV_Masked, OPC_OP_V,
               (outs VR:$vd), (ins GPR:$rs1, VR:$vs2, VLR:$vl, VMR:$vm),
               opcodestr, "$vd, $rs1, $vs2, $vm"> {
    bits<5> rs1;
    let Inst{19-15} = rs1;
  }
  def _um : RVInstVA<funct6, 0b110, RVV_Unmasked, OPC_OP_V,
               (outs VR:$vd), (ins GPR:$rs1, VR:$vs2, VLR:$vl),
               opcodestr, "$vd, $rs1, $vs2"> {
    bits<5> rs1;
    let Inst{19-15} = rs1;
  }
}
```

RISCVInstrInfoV.td

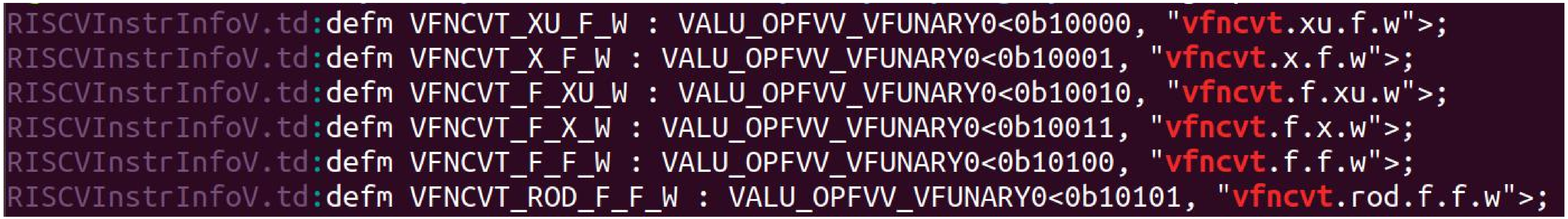
• 02 玄铁C910和rvv对比补充

2 C910有但没有具体的指令格式

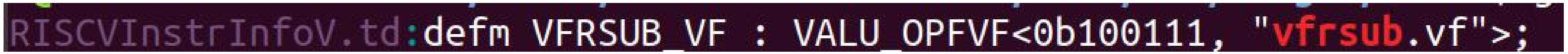
vfncvt.f.f.v vd, vs2, vm   # Convert double-width float to single-width float. 矢量浮点缩位转换成浮点指令

vfrsub.vf vd, vs2, rs1, vm # Scalar-vector vd[i] = f[rs1] - vs2[i]

标量矢量浮点减法指令



```
RISCVInstrInfoV.td:defm VFNCVT_XU_F_W : VALU_OPFVV_VFUNARY0<0b10000, "vfncvt.xu.f.w">;
RISCVInstrInfoV.td:defm VFNCVT_X_F_W : VALU_OPFVV_VFUNARY0<0b10001, "vfncvt.x.f.w">;
RISCVInstrInfoV.td:defm VFNCVT_F_XU_W : VALU_OPFVV_VFUNARY0<0b10010, "vfncvt.f.xu.w">;
RISCVInstrInfoV.td:defm VFNCVT_F_X_W : VALU_OPFVV_VFUNARY0<0b10011, "vfncvt.f.x.w">;
RISCVInstrInfoV.td:defm VFNCVT_F_F_W : VALU_OPFVV_VFUNARY0<0b10100, "vfncvt.f.f.w">;
RISCVInstrInfoV.td:defm VFNCVT_ROD_F_F_W : VALU_OPFVV_VFUNARY0<0b10101, "vfncvt.rod.f.f.w">;
```



```
RISCVInstrInfoV.td:defm VFRSUB_VF : VALU_OPFVF<0b100111, "vfrsub.vf">;
```

RISCVInstrInfoV.td

rvv-llvm没有定义vfncvt.f.f.v指令

3 C910没有RVV手册有

vmseq.vi vd, vs2, imm, vm # vector-immediate

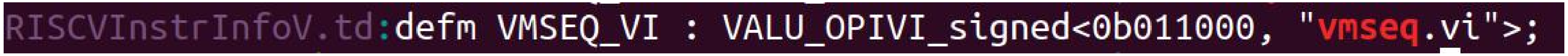
vmsgeu和vmsge（Set if greater than or equal）指令

vdotu.vv vd, vs2, vs1, vm # Vector-vector Unsigned dot-product

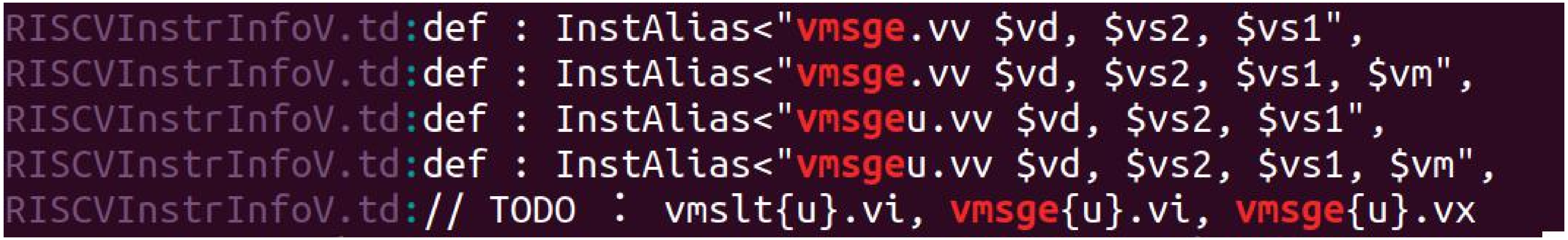
vdot.vv vd, vs2, vs1, vm   # Vector-vector Signed dot-product

vfdot.v vd, vs2, vs1, vm   # vd[i][31:0] += vs2[i][31:16] * vs1[i][31:16]+vs2[i][15:0] * vs1[i][15:0]

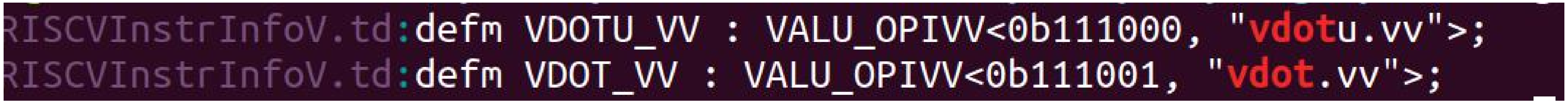
vfdot.vv vd, vs2, vs1, vm   # Vector-vector



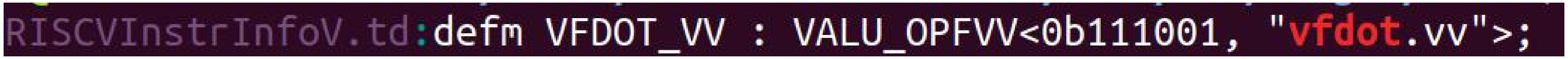
```
RISCVInstrInfoV.td:defm VMSEQ_VI : VALU_OPIVI_signed<0b011000, "vmseq.vi">;
```



```
RISCVInstrInfoV.td:def : InstAlias<"vmsge.vv $vd, $vs2, $vs1",
RISCVInstrInfoV.td:def : InstAlias<"vmsge.vv $vd, $vs2, $vs1, $vm",
RISCVInstrInfoV.td:def : InstAlias<"vmsgeu.vv $vd, $vs2, $vs1",
RISCVInstrInfoV.td:def : InstAlias<"vmsgeu.vv $vd, $vs2, $vs1, $vm",
RISCVInstrInfoV.td:// TODO : vmslt{u}.vi, vmsge{u}.vi, vmsge{u}.vx
```



```
RISCVInstrInfoV.td:defm VDOTU_VV : VALU_OPIVV<0b111000, "vdotu.vv">;
RISCVInstrInfoV.td:defm VDOT_VV : VALU_OPIVV<0b111001, "vdot.vv">;
```



```
RISCVInstrInfoV.td:defm VFDOT_VV : VALU_OPFVV<0b111001, "vfdot.vv">;
```

vfdot.v原本在rvv中，但是rvv-llvm源代码中没有介绍。

• 03 rvv-llvm代码简单介绍

llvm/lib/Target/RISCV目录下相关文件

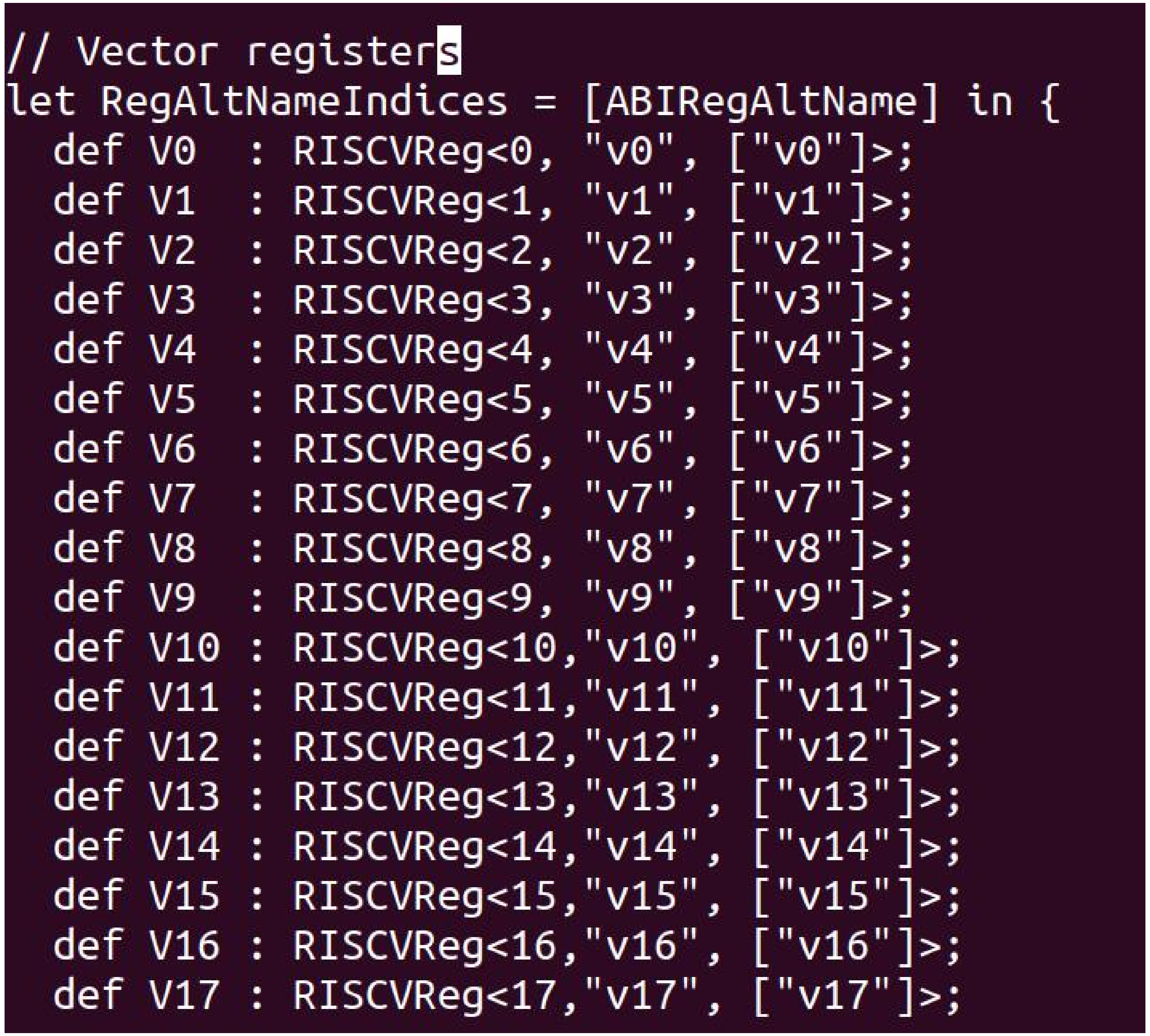
llvm/test/MC/RISCV目录下相关文件

rvv-mask-valid.s

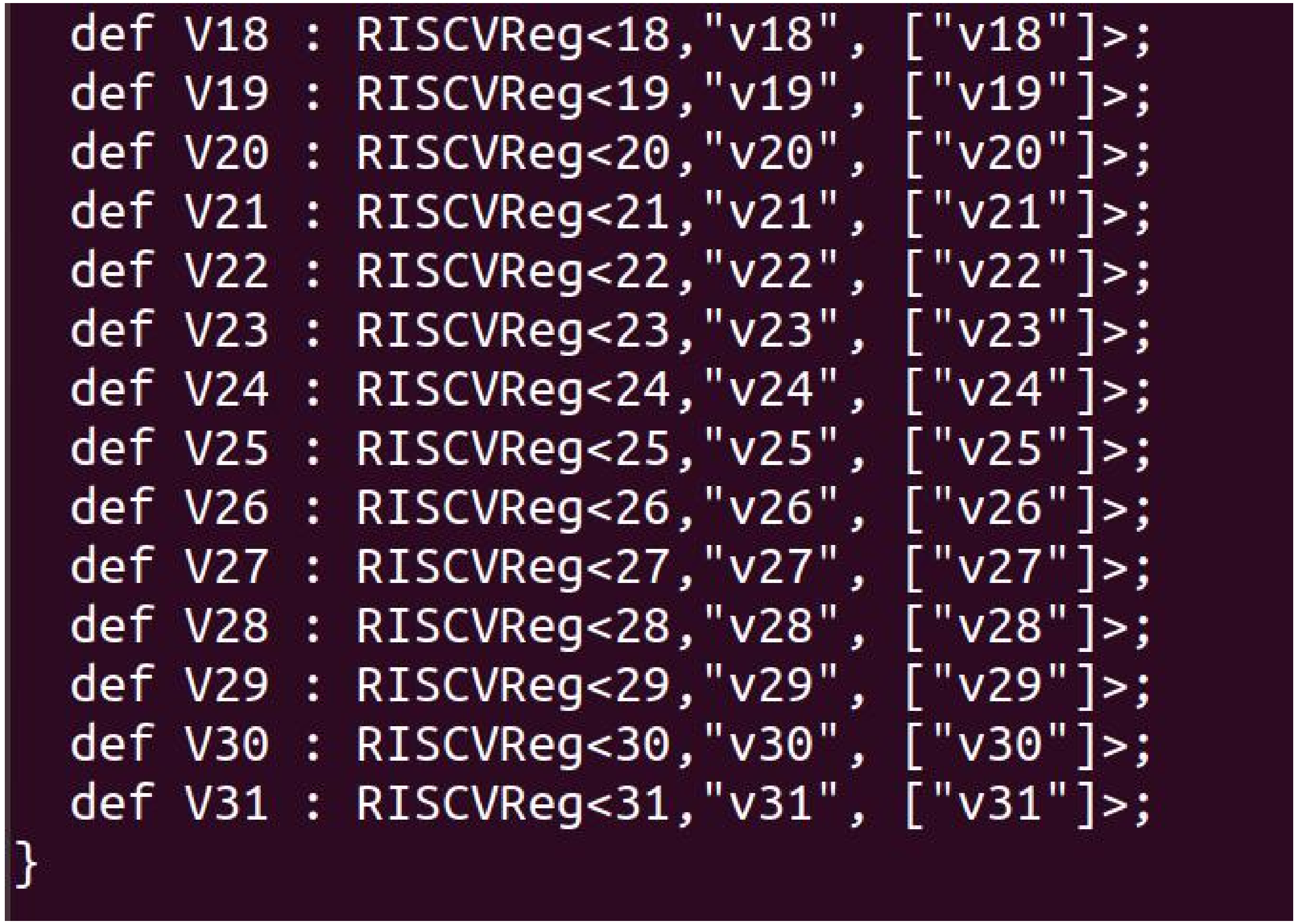
rvv-pseudo-mask-valid.s

rvv-pseudo-valid.s

rvv-valid.s

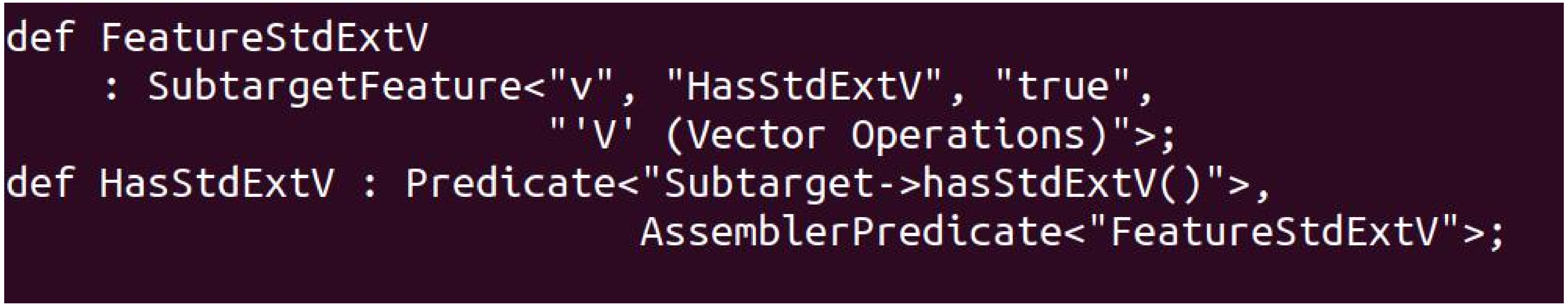


```
// Vector registers
let RegAltNameIndices = [ABIRegAltName] in {
  def V0  : RISCVReg<0, "v0", ["v0"]>;
  def V1  : RISCVReg<1, "v1", ["v1"]>;
  def V2  : RISCVReg<2, "v2", ["v2"]>;
  def V3  : RISCVReg<3, "v3", ["v3"]>;
  def V4  : RISCVReg<4, "v4", ["v4"]>;
  def V5  : RISCVReg<5, "v5", ["v5"]>;
  def V6  : RISCVReg<6, "v6", ["v6"]>;
  def V7  : RISCVReg<7, "v7", ["v7"]>;
  def V8  : RISCVReg<8, "v8", ["v8"]>;
  def V9  : RISCVReg<9, "v9", ["v9"]>;
  def V10 : RISCVReg<10,"v10", ["v10"]>;
  def V11 : RISCVReg<11,"v11", ["v11"]>;
  def V12 : RISCVReg<12,"v12", ["v12"]>;
  def V13 : RISCVReg<13,"v13", ["v13"]>;
  def V14 : RISCVReg<14,"v14", ["v14"]>;
  def V15 : RISCVReg<15,"v15", ["v15"]>;
  def V16 : RISCVReg<16,"v16", ["v16"]>;
  def V17 : RISCVReg<17,"v17", ["v17"]>;
```

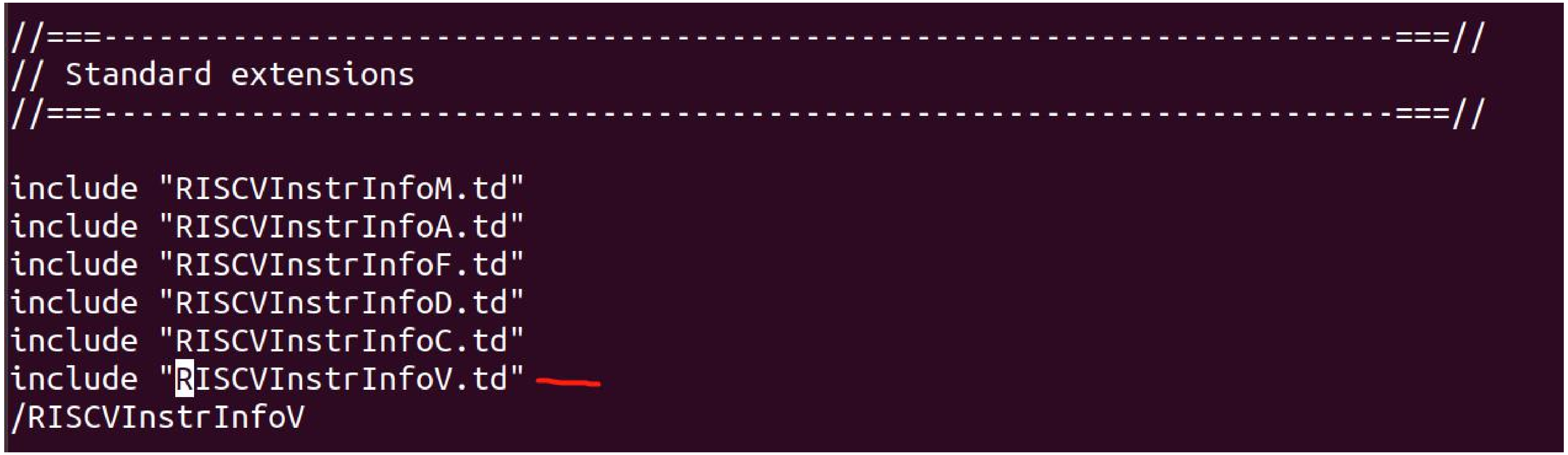


```
  def V18 : RISCVReg<18,"v18", ["v18"]>;
  def V19 : RISCVReg<19,"v19", ["v19"]>;
  def V20 : RISCVReg<20,"v20", ["v20"]>;
  def V21 : RISCVReg<21,"v21", ["v21"]>;
  def V22 : RISCVReg<22,"v22", ["v22"]>;
  def V23 : RISCVReg<23,"v23", ["v23"]>;
  def V24 : RISCVReg<24,"v24", ["v24"]>;
  def V25 : RISCVReg<25,"v25", ["v25"]>;
  def V26 : RISCVReg<26,"v26", ["v26"]>;
  def V27 : RISCVReg<27,"v27", ["v27"]>;
  def V28 : RISCVReg<28,"v28", ["v28"]>;
  def V29 : RISCVReg<29,"v29", ["v29"]>;
  def V30 : RISCVReg<30,"v30", ["v30"]>;
  def V31 : RISCVReg<31,"v31", ["v31"]>;
}
```

RISCVRegisterInfo.td



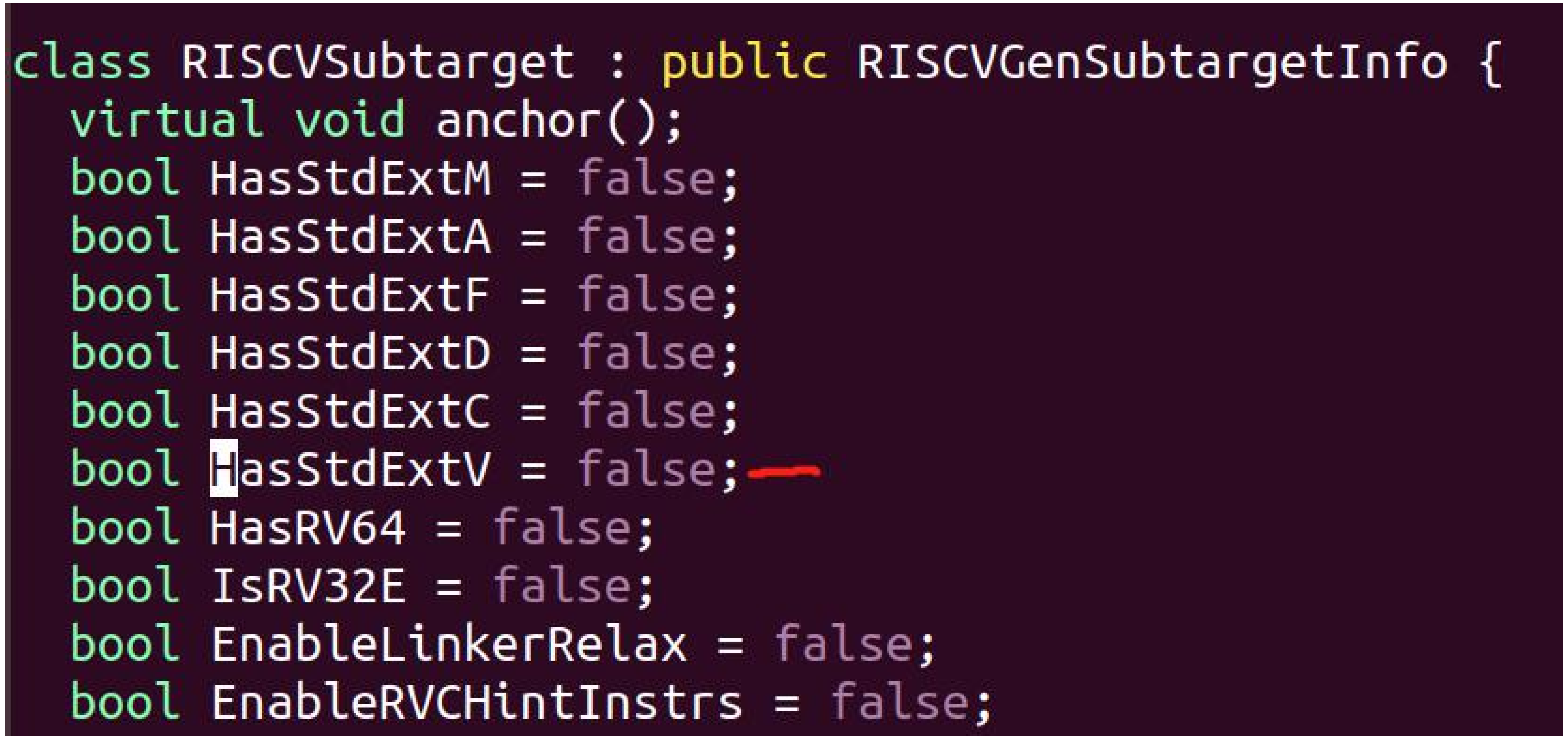
```
def FeatureStdExtV
    : SubtargetFeature<"v", "HasStdExtV", "true",
                       "'V' (Vector Operations)">;
def HasStdExtV : Predicate<"Subtarget->hasStdExtV()">,
                 AssemblerPredicate<"FeatureStdExtV">;
```



```
//===----------------------------------------------------------------------===//
// Standard extensions
//===----------------------------------------------------------------------===//

include "RISCVInstrInfoM.td"
include "RISCVInstrInfoA.td"
include "RISCVInstrInfoF.td"
include "RISCVInstrInfoD.td"
include "RISCVInstrInfoC.td"
include "RISCVInstrInfoV.td"
/RISCVInstrInfoV
```

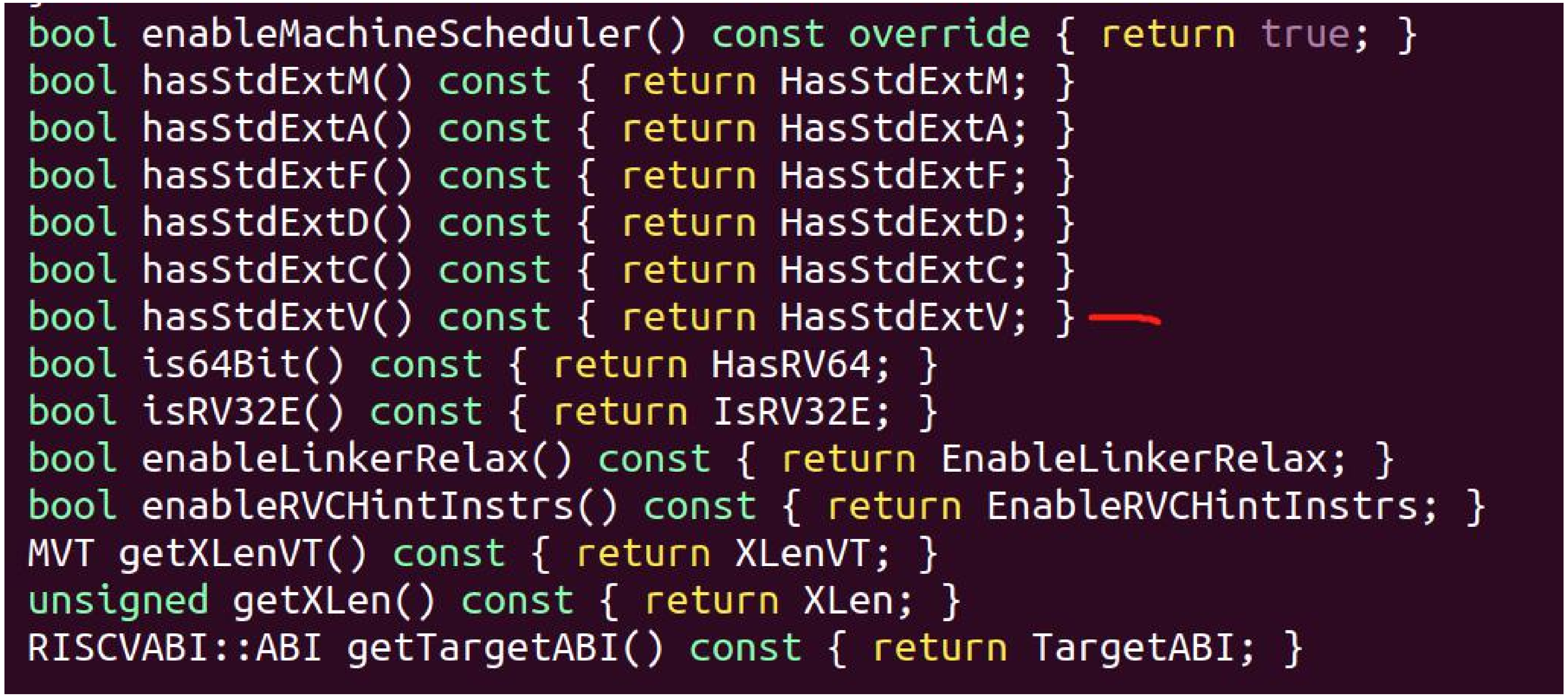
图1：RISCV.td include "llvm/TableGen/SearchableTable.td"

图2：RISCVInstrInfo.td



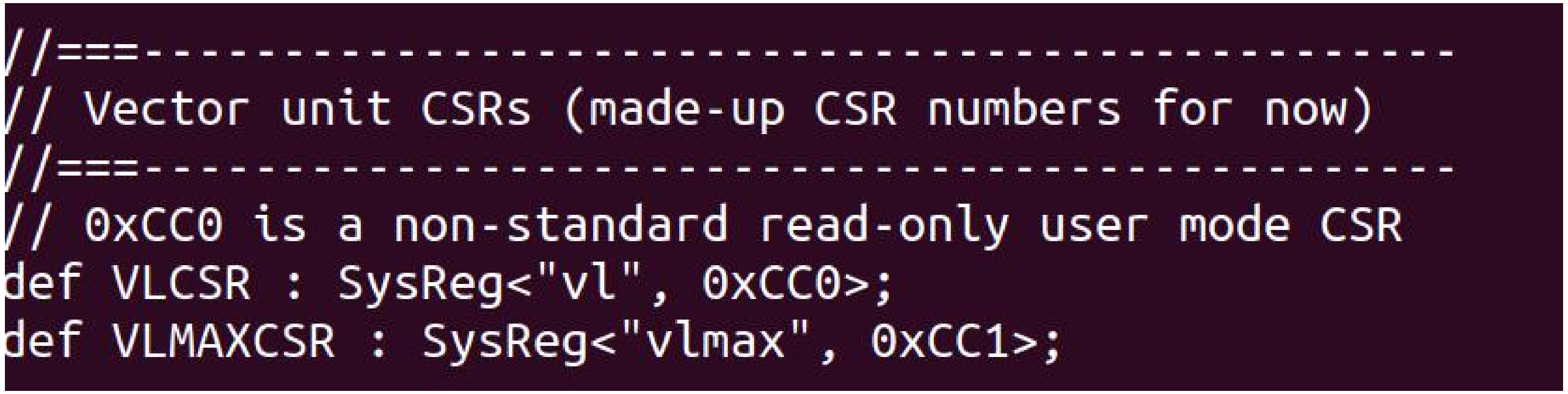
```
class RISCVSubtarget : public RISCVGenSubtargetInfo {
  virtual void anchor();
  bool HasStdExtM = false;
  bool HasStdExtA = false;
  bool HasStdExtF = false;
  bool HasStdExtD = false;
  bool HasStdExtC = false;
  bool HasStdExtV = false;
  bool HasRV64 = false;
  bool IsRV32E = false;
  bool EnableLinkerRelax = false;
  bool EnableRVCHintInstrs = false;
```

RISCVSubtarget.h

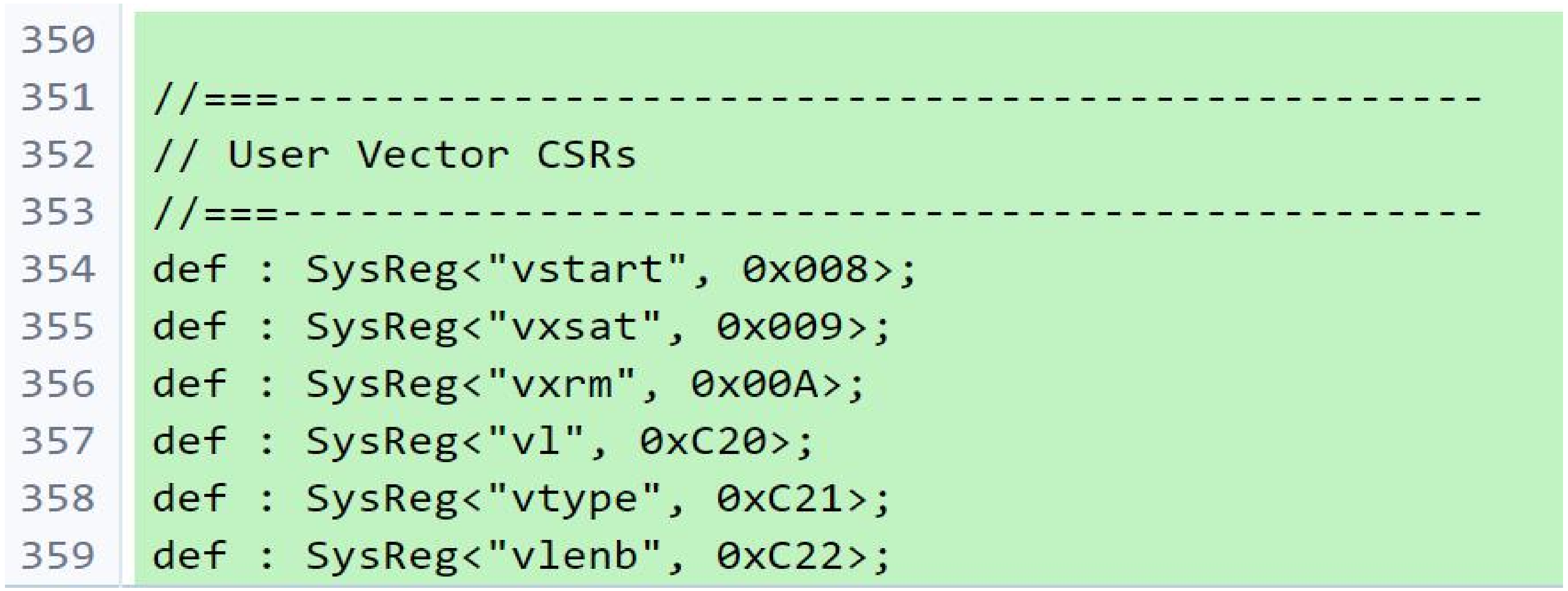


```
bool enableMachineScheduler() const override { return true; }
bool hasStdExtM() const { return HasStdExtM; }
bool hasStdExtA() const { return HasStdExtA; }
bool hasStdExtF() const { return HasStdExtF; }
bool hasStdExtD() const { return HasStdExtD; }
bool hasStdExtC() const { return HasStdExtC; }
bool hasStdExtV() const { return HasStdExtV; }
bool is64Bit() const { return HasRV64; }
bool isRV32E() const { return IsRV32E; }
bool enableLinkerRelax() const { return EnableLinkerRelax; }
bool enableRVCHintInstrs() const { return EnableRVCHintInstrs; }
MVT getXLenVT() const { return XLenVT; }
unsigned getXLen() const { return XLen; }
RISCVABI::ABI getTargetABI() const { return TargetABI; }
```

RISCVSubtarget.h



```
//===------------------------------------------------
// Vector unit CSRs (made-up CSR numbers for now)
//===------------------------------------------------
// 0xCC0 is a non-standard read-only user mode CSR
def VLCSR : SysReg<"vl", 0xCC0>;
def VLMAXCSR : SysReg<"vlmax", 0xCC1>;
```

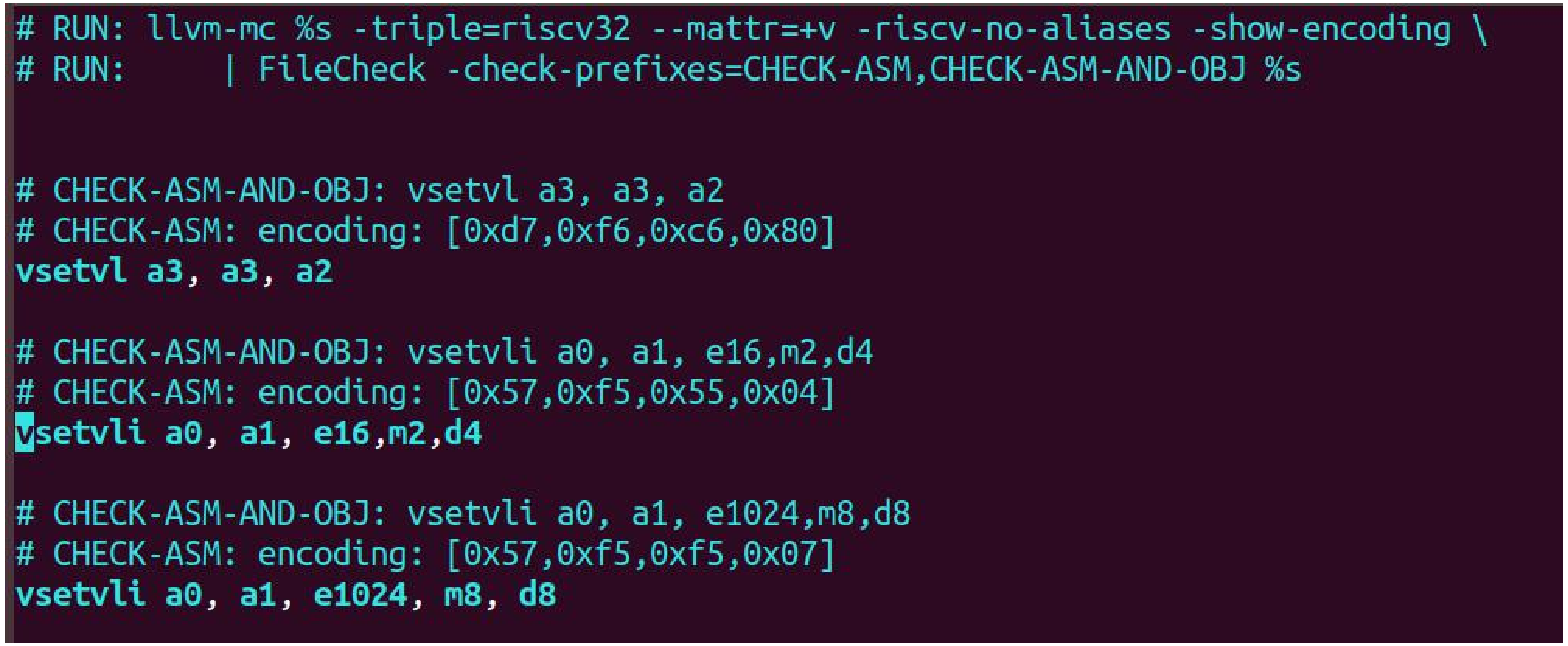


```

//===----------------------------------------------
// User Vector CSRs
//===----------------------------------------------
def : SysReg<"vstart", 0x008>;
def : SysReg<"vxsat", 0x009>;
def : SysReg<"vxrm", 0x00A>;
def : SysReg<"vl", 0xC20>;
def : SysReg<"vtype", 0xC21>;
def : SysReg<"vlenb", 0xC22>;
```

RISCVSystemOperands.td

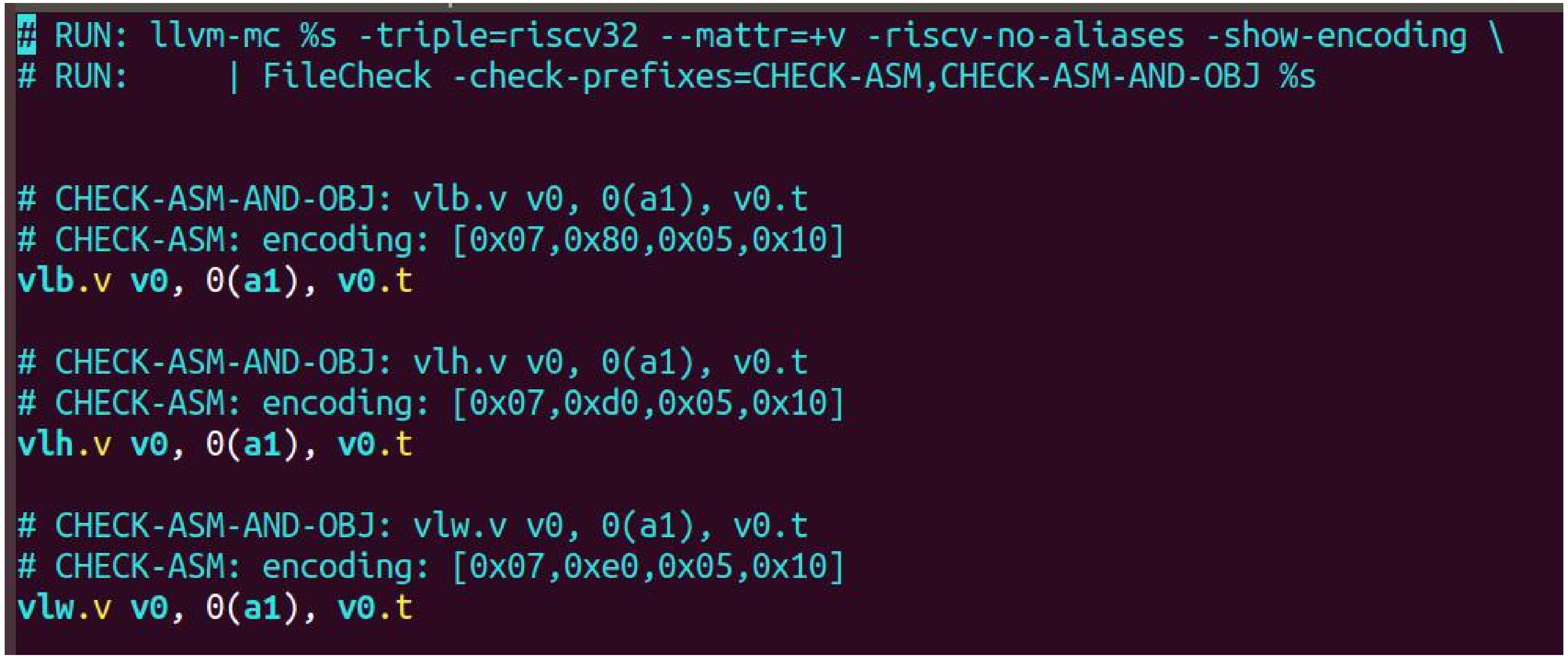
图1是rvv-llvm的定义

图2是Roger Espasa团队写的



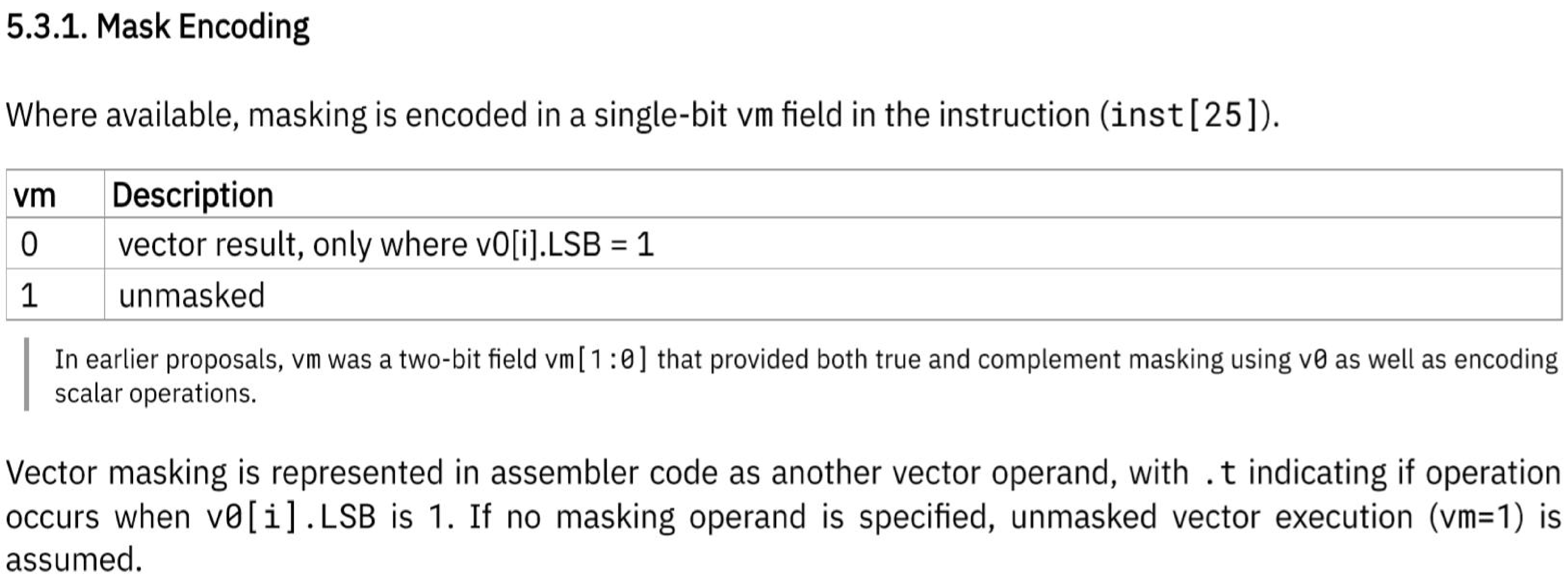
```
# RUN: llvm-mc %s -triple=riscv32 --mattr=+v -riscv-no-aliases -show-encoding \
# RUN:      | FileCheck -check-prefixes=CHECK-ASM,CHECK-ASM-AND-OBJ %s


# CHECK-ASM-AND-OBJ: vsetvl a3, a3, a2
# CHECK-ASM: encoding: [0xd7,0xf6,0xc6,0x80]
vsetvl a3, a3, a2

# CHECK-ASM-AND-OBJ: vsetvli a0, a1, e16,m2,d4
# CHECK-ASM: encoding: [0x57,0xf5,0x55,0x04]
vsetvli a0, a1, e16,m2,d4

# CHECK-ASM-AND-OBJ: vsetvli a0, a1, e1024,m8,d8
# CHECK-ASM: encoding: [0x57,0xf5,0xf5,0x07]
vsetvli a0, a1, e1024, m8, d8
```

rvv-valid.s



```
# RUN: llvm-mc %s -triple=riscv32 --mattr=+v -riscv-no-aliases -show-encoding \
# RUN:       | FileCheck -check-prefixes=CHECK-ASM,CHECK-ASM-AND-OBJ %s


# CHECK-ASM-AND-OBJ: vlb.v v0, 0(a1), v0.t
# CHECK-ASM: encoding: [0x07,0x80,0x05,0x10]
vlb.v v0, 0(a1), v0.t

# CHECK-ASM-AND-OBJ: vlh.v v0, 0(a1), v0.t
# CHECK-ASM: encoding: [0x07,0xd0,0x05,0x10]
vlh.v v0, 0(a1), v0.t

# CHECK-ASM-AND-OBJ: vlw.v v0, 0(a1), v0.t
# CHECK-ASM: encoding: [0x07,0xe0,0x05,0x10]
vlw.v v0, 0(a1), v0.t
```

rvv-mask-valid.s



5.3.1. Mask Encoding

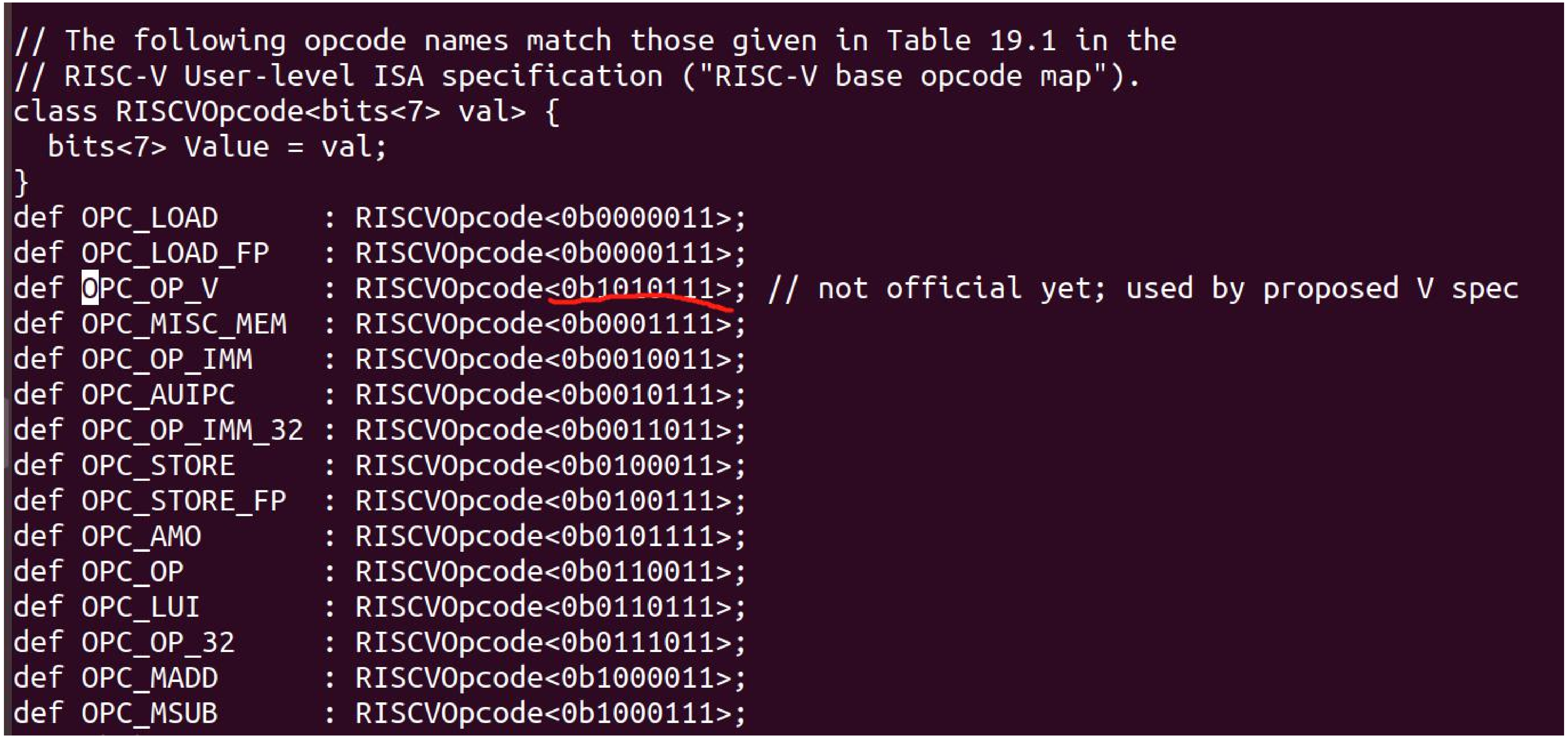
Where available, masking is encoded in a single-bit vm field in the instruction (`inst[25]`).

vm	Description
0	vector result, only where v0[i].LSB = 1
1	unmasked

> In earlier proposals, vm was a two-bit field `vm[1:0]` that provided both true and complement masking using `v0` as well as encoding scalar operations.

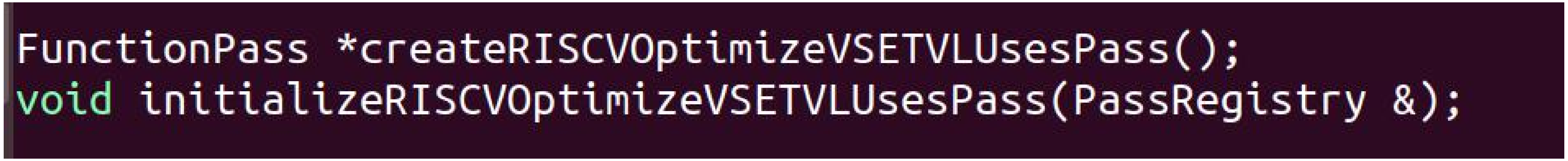
Vector masking is represented in assembler code as another vector operand, with `.t` indicating if operation occurs when `v0[i].LSB` is 1. If no masking operand is specified, unmasked vector execution (`vm=1`) is assumed.

riscv-v-spec 0.7.1



```
// The following opcode names match those given in Table 19.1 in the
// RISC-V User-level ISA specification ("RISC-V base opcode map").
class RISCVOpcode<bits<7> val> {
  bits<7> Value = val;
}
def OPC_LOAD      : RISCVOpcode<0b0000011>;
def OPC_LOAD_FP   : RISCVOpcode<0b0000111>;
def OPC_OP_V      : RISCVOpcode<0b1010111>; // not official yet; used by proposed V spec
def OPC_MISC_MEM  : RISCVOpcode<0b0001111>;
def OPC_OP_IMM    : RISCVOpcode<0b0010011>;
def OPC_AUIPC     : RISCVOpcode<0b0010111>;
def OPC_OP_IMM_32 : RISCVOpcode<0b0011011>;
def OPC_STORE     : RISCVOpcode<0b0100011>;
def OPC_STORE_FP  : RISCVOpcode<0b0100111>;
def OPC_AMO       : RISCVOpcode<0b0101111>;
def OPC_OP        : RISCVOpcode<0b0110011>;
def OPC_LUI       : RISCVOpcode<0b0110111>;
def OPC_OP_32     : RISCVOpcode<0b0111011>;
def OPC_MADD      : RISCVOpcode<0b1000011>;
def OPC_MSUB      : RISCVOpcode<0b1000111>;
```

RISCVInstrFormats.td



```
FunctionPass *createRISCVOptimizeVSETVLUsesPass();
void initializeRISCVOptimizeVSETVLUsesPass(PassRegistry &);
```

RISCV.h

//===--===//

// This file contains the entry points for global functions defined in the LLVM

// RISC-V back-end.

//===--===//

Optimization pass to avoid copies to GPR when using VSETVL

在如下文件中定义了

llvm/lib/CodeGen/SelectionDAG/SelectionDAGBuilder.cpp

void SelectionDAGBuilder::visitShuffleVector(const User &I)

llvm/include/llvm/IR/IntrinsicsRISCV.td

// Vector extension def int_riscv_vadd等

llvm/lib/Target/RISCV/RISCV.h

FunctionPass *createRISCVOptimizeVSETVLUsesPass();

void initializeRISCVOptimizeVSETVLUsesPass(PassRegistry &);

• 04 参考资料

llvm学习手册指导

https://github.com/llvm/llvm-project/blob/master/llvm/docs/tutorial

《玄铁C910指令集手册》

PLCT实验室的rvv-llvm

https://github.com/isrc-cas/rvv-llvm/tree/rvv-iscas/ llvm/

• 05 问题

llvm/lib/Target/RISCV/RISCV.h

```
FunctionPass *createRISCVOptimizeVSETVLUsesPass();
void initializeRISCVOptimizeVSETVLUsesPass(PassRegistry &)
```

这些是根据哪些参考资料写出来的?

谢 谢

欢迎交流合作

2019/02/25