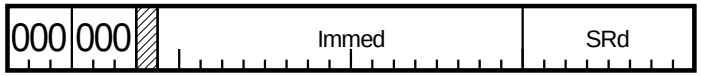
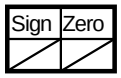
Mnemonic: LDI Immed,SRd

Instruction: Load Immediate

Format:



Flags:

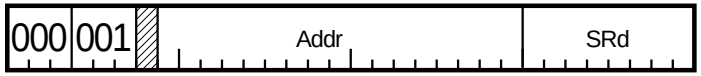


Description: Immed->SRd ; (Immed=17bits)

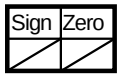
Mnemonic: LD Addr,SRd

Instruction: Load direct

Format:



Flags:



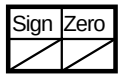
Description: M[Addr]->SRd ; (Addr=17bits)

Mnemonic: LDB SRb,Disp,SRd

Instruction: Load based

Format:

Flags:



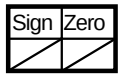
Description: M[SRb+Disp]->SRd ; (Disp=9bits)

Mnemonic: LDBI SRb,SRi,SRd

Instruction: Load base-indexed

Format:

Flags:



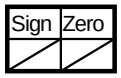
Description: M[SRb+SRi]->SRd

Mnemonic: LDBS SRb,Sft,SRd

Instruction: Load base-shifted

Format:

Flags:

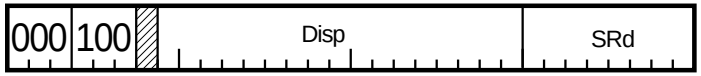


Description: M[SRb>>Sft]->SRd ; (Sft=9bits)

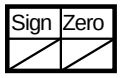
Mnemonic: LDPCB Disp,SRd

Instruction: Load PC-based

Format:



Flags:

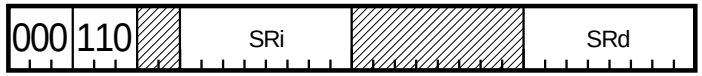


Description: M[PC+Disp]->SRd ; (Disp=17bits)

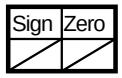
Mnemonic: LDPCBI SRi,SRd

Instruction: Load PC-base-indexed

Format:



Flags:

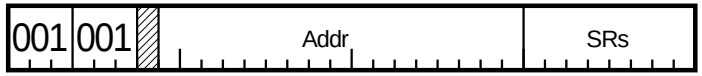


Description: M[PC+SRi]->SRd

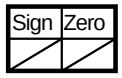
Mnemonic: ST Addr,SRs

Instruction: Store direct

Format:



Flags:



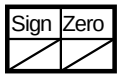
Description: SRs->M[Addr] ; (Addr=17bits)

Mnemonic: STB SRb,Disp,SRs

Instruction: Store based

Format:

Flags:



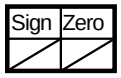
Description: SRs->M[SRb+Disp] ; (Disp=9bits)

Mnemonic: STBI SRb,SRi,SRs

Instruction: Store base-indexed

Format:

Flags:



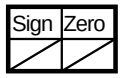
Description: SRs->M[SRb+SRi]

Mnemonic: STBS SRb,Sft,SRs

Instruction: Store base-shifted

Format:

Flags:

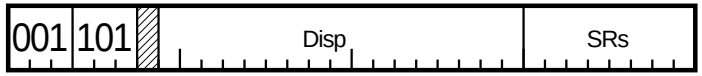


Description: SRs->M[SRb>>Sft] ; Sft=9bits

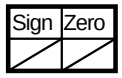
Mnemonic: STPCB Disp,SRs

Instruction: Store PC-based

Format:



Flags:

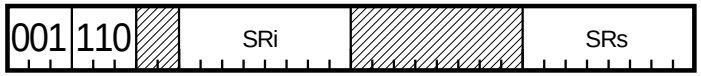


Description: SRs->M[PC+Disp] ; Disp=17bits

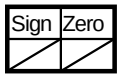
Mnemonic: STPCBI SRi,SRs

Instruction: Store PC-base-indexed

Format:



Flags:

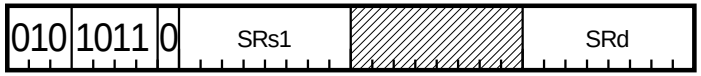


Description: SRs->M[PC+SRi]

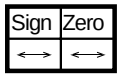
Mnemonic: NOT SRs1,SRd

Instruction: Logical NOT

Format:



Flags:



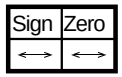
Description: ~SRs1 -> SRd

Mnemonic: ADD SRs1,SRs2,SRd

Instruction: Integer addition

Format:

Flags:



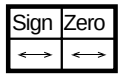
Description: SRs2 + SRs1 -> SRd

Mnemonic: ADDI Immed,SRs2,SRd

Instruction: Integer addition immediate

Format:

Flags:



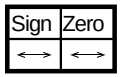
Description: SRs2 + Immed -> SRd ; (Immed=8bits)

Mnemonic: SUB SRs1,SRs2,SRd

Instruction: Integer subtract

Format:

Flags:



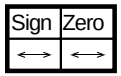
Description: SRs2 - SRs1 -> SRd

Mnemonic: SUBI Immed,SRs2,SRd

Instruction: Integer subtract immediate

Format:

Flags:



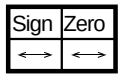
Description: SRs2 - Immed -> SRd ; (Immed=8bits)

Mnemonic: MUL SRs1,SRs2,SRd

Instruction: Integer multiply

Format:

Flags:



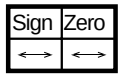
Description: SRs2 × SRs1 -> SRd

Mnemonic: MULI Immed,SRs2,SRd

Instruction: Integer multiply immediate

Format:

Flags:



Description: SRs2 × Immed -> SRd ; (Immed=8bits)

Mnemonic: DIV SRs1,SRs2,SRd

Instruction: Integer division

Format:

Flags:

Sign	Zero
↔	↔

Description: SRs2 ÷ SRs1 -> SRd

Mnemonic: DIVI Immed,SRs2,SRd

Instruction: Integer division immediate

Format:

Flags:

Sign	Zero
↔	↔

Description: SRs2 ÷ Immed -> SRd ; (Immed=8bits)

Mnemonic: MOD SRs1,SRs2,SRd

Instruction: Integer modulo

Format:

Flags:

Sign	Zero
↔	↔

Description: SRs2 mod SRs1 -> SRd

Mnemonic: MODI Immed,SRs2,SRd

Instruction: Integer modulo immediate

Format:

Flags:

Sign	Zero
↔	↔

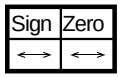
Description: SRs2 mod Immed -> SRd ; (Immed=8bits)

Mnemonic: RSUB SRs1,SRs2,SRd

Instruction: Integer reverse subtract

Format:

Flags:



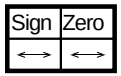
Description: SRs1 - SRs2 -> SRd

Mnemonic: RSUBI Immed,SRs2,SRd

Instruction: Integer reverse subtract immediate

Format:

Flags:



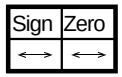
Description: Immed - SRs2 -> SRd ; (Immed=8bits)

Mnemonic: UMUL SRs1,SRs2,SRd

Instruction: Unsigned integer multiply

Format:

Flags:



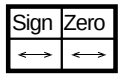
Description: SRs2 × SRs1 -> SRd (unsigned)

Mnemonic: UMULI Immed,SRs2,SRd

Instruction: Unsigned integer multiply immediate

Format:

Flags:



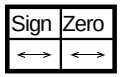
Description: SRs2 × Immed -> SRd ; (Immed=8bits,unsigned)

Mnemonic: UDIV SRs1,SRs2,SRd

Instruction: Unsigned integer division

Format:

Flags:



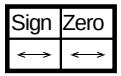
Description: SRs2 ÷ SRs1 -> SRd (unsigned)

Mnemonic: UDIVI Immed,SRs2,SRd

Instruction: Unsigned integer division immediate

Format:

Flags:



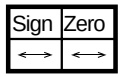
Description: SRs2 ÷ Immed -> SRd ; (Immed=8bits,unsigned)

Mnemonic: AND SRs1,SRs2,SRd

Instruction: Logical AND

Format:

Flags:



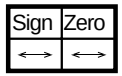
Description: SRs2 & SRs1 -> SRd

Mnemonic: ANDI Immed,SRs2,SRd

Instruction: Logical AND immediate

Format:

Flags:



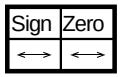
Description: SRs2 & Immed -> SRd ; (Immed=8bits)

Mnemonic: OR SRs1,SRs2,SRd

Instruction: Logical OR

Format:

Flags:



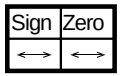
Description: SRs2 | SRs1 -> SRd

Mnemonic: ORI Immed,SRs2,SRd

Instruction: Logical OR immediate

Format:

Flags:



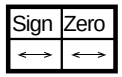
Description: SRs2 | Immed -> SRd ; (Immed=8bits)

Mnemonic: XOR SRs1,SRs2,SRd

Instruction: Logical XOR

Format:

Flags:



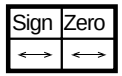
Description: SRs2 ^ SRs1 -> SRd

Mnemonic: XORI Immed,SRs2,SRd

Instruction: Logical XOR immediate

Format:

Flags:



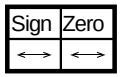
Description: SRs2 ^ Immed -> SRd ; (Immed=8bits)

Mnemonic: SRL SRs1,SRs2,SRd

Instruction: Shift right logical

Format:

Flags:



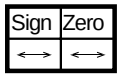
Description: SRs2 >> SRs1 -> SRd

Mnemonic: SRLI Immed,SRs2,SRd

Instruction: Shift right logical immediate

Format:

Flags:



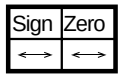
Description: SRs2 >> Immed -> SRd ; (Immed=8bits)

Mnemonic: SLL SRs1,SRs2,SRd

Instruction: Shift left logical

Format:

Flags:



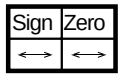
Description: SRs2 << SRs1 -> SRd

Mnemonic: SLLI Immed,SRs2,SRd

Instruction: Shift left logical immediate

Format:

Flags:



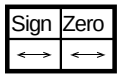
Description: SRs2 << Immed -> SRd ; (Immed=8bits)

Mnemonic: SRA SRs1,SRs2,SRd

Instruction: Shift right arithmetic

Format:

Flags:



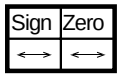
Description: SRs2 >> SRs1 -> SRd

Mnemonic: SRAI Immed,SRs2,SRd

Instruction: Shift right arithmetic immediate

Format:

Flags:



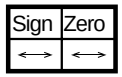
Description: SRs2 >> Immed -> SRd ; (Immed=8bits)

Mnemonic: SLA SRs1,SRs2,SRd

Instruction: Shift left arithmetic

Format:

Flags:



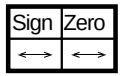
Description: SRs2 << SRs1 -> SRd

Mnemonic: SLAI Immed,SRs2,SRd

Instruction: Shift left arithmetic immediate

Format:

Flags:



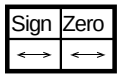
Description: SRs2 << Immed -> SRd ; (Immed=8bits)

Mnemonic: FADD SRs1,SRs2,SRd

Instruction: floating point addition

Format:

Flags:



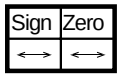
Description: SRs2 + SRs1 -> SRd

Mnemonic: FSUB SRs1,SRs2,SRd

Instruction: floating point subtract

Format:

Flags:



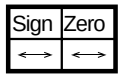
Description: SRs2 - SRs1 -> SRd

Mnemonic: FMUL SRs1,SRs2,SRd

Instruction: floating point multiply

Format:

Flags:



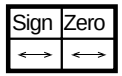
Description: SRs2 × SRs1 -> SRd

Mnemonic: FDIV SRs1,SRs2,SRd

Instruction: floating point division

Format:

Flags:

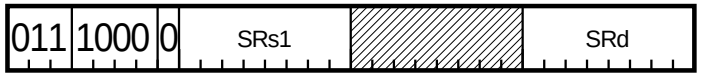


Description: SRs2 ÷ SRs1 -> SRd

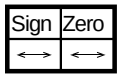
Mnemonic: **INT SRs1,SRd**

Instruction: floating point to integer conversion

Format:



Flags:

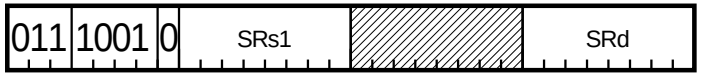


Description: (int)SRs1 -> SRd

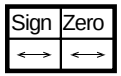
Mnemonic: **FLT SRs1,SRd**

Instruction: integer to floating point conversion

Format:



Flags:

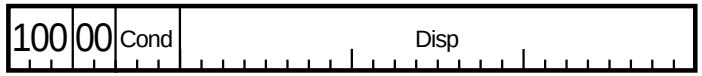


Description: (float)SRs1 -> SRd

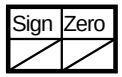
Mnemonic: **JMP Disp ; JR Disp**
JC Cond,Disp ; JRC Cond,Disp

Instruction: Jump Relative
Jump Relative Conditional

Format:



Flags:



Description: if (Cond) then PC + Disp -> PC (Disp=24bits)

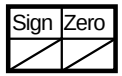
Cond: 000 - jump always
001 - plus
010 - minus
011 - zero
100 - not plus
101 - not minus
110 - not zero
111 - (reserved)

Mnemonic: **JB SRb,Disp**
JBC Cond,SRb,Disp

Instruction: Jump Based
Conditional Jump Based

Format:

Flags:



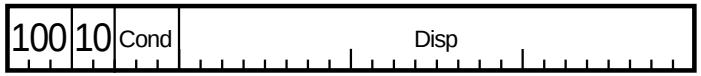
Description: if Cond then SRb + Disp -> PC (Disp=16bits)

Cond: 000 - jump always
001 - plus
010 - minus
011 - zero
100 - not plus
101 - not minus
110 - not zero
111 - (reserved)

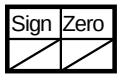
Mnemonic: **JMPL Disp ; JRL Disp**
JCL Cond,Disp ; JRCL Cond,Disp

Instruction: Jump Relative and Link
Jump Relative Conditional and Link

Format:



Flags:



Description: if (Cond) then PC+2 -> LinkReg ; PC + Disp -> PC (Disp=24bits)

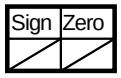
Cond: 000 - jump always
001 - plus
010 - minus
011 - zero
100 - not plus
101 - not minus
110 - not zero
111 - (reserved)

Mnemonic: **JBL SRb,Disp**
JBCL Cond,SRb,Disp

Instruction: Jump Based and Link
Conditional Jump Based and Link

Format:

Flags:



Description: if Cond then PC -> LinkReg ; SRb + Disp -> PC (Disp=16bits)

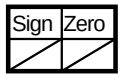
Cond: 000 - jump always
001 - plus
010 - minus
011 - zero
100 - not plus
101 - not minus
110 - not zero
111 - (reserved)

Mnemonic: **CJ(Cond) Opr,SRs1,SRs2,Disp**

Instruction: Compare and jump conditional

Format:

Flags:



Description: if (SRs2 op SRs1 = Cond) then PC+Disp -> PC ; (Disp=8bits)

Cond: 00 - plus (CJP)
01 - minus (CJM)
10 - zero (CJZ)
11 - not zero (CJNZ)

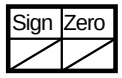
Opr: 00 - SRs2 - SRs1 (SUB)
01 - SRs1 - SRs2 (RSUB)
10 - SRs2 AND SRs1 (AND)
11 - SRs2 OR SRs1 (OR)

Mnemonic: **CJ(Cond)I Opr,Immed,SRs2,Disp**

Instruction: Compare and jump conditional immediate

Format:

Flags:



Description: if (Immed op SRs1 = Cond) then PC+Disp -> PC ; (Disp=8bits)

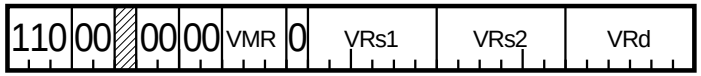
Cond: 00 - plus (CJP)
01 - minus (CJM)
10 - zero (CJZ)
11 - not zero (CJNZ)

Opr: 00 - SRs2 - Immed (SUB)
01 - Immed - SRs2 (RSUB)
10 - SRs2 AND Immed (AND)
11 - SRs2 OR Immed (OR)

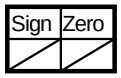
Mnemonic: VFADD VRs1,VRs2,VRd{,VMR}

Instruction: Vector floating point addition

Format:



Flags:

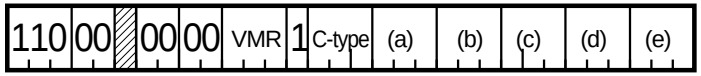


Description: VRs2 + VRs1 -> VRd ; using vector mask register VMR

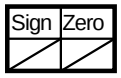
Mnemonic: CVFADD src1,src2,dstn{,VMR}

Instruction: Vector floating point addition with chaining

Format:



Flags:



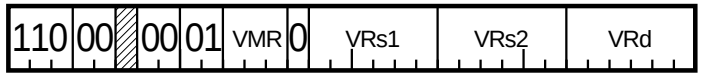
Description: src2 + src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

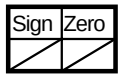
Mnemonic: VFSUB VRs1,VRs2,VRd{,VMR}

Instruction: Vector floating point subtract

Format:



Flags:

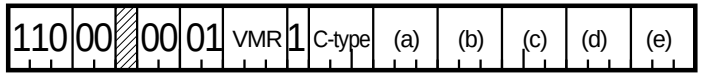


Description: VRs2 - VRs1 -> VRd ; using vector mask register VMR

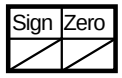
Mnemonic: CVFSUB src1,src2,dstn{,VMR}

Instruction: Vector floating point subtract with chaining

Format:



Flags:



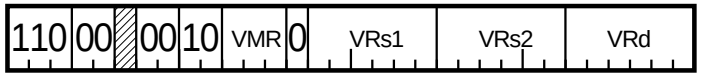
Description: src2 - src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

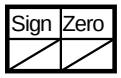
Mnemonic: VFMUL VRs1,VRs2,VRd{,VMR}

Instruction: Vector floating point multiply

Format:



Flags:



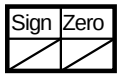
Description: VRs2 × VRs1 -> VRd ; using vector mask register VMR

Mnemonic: CVFMUL src1,src2,dstn{,VMR}

Instruction: Vector floating point multiply with chaining

Format:

Flags:



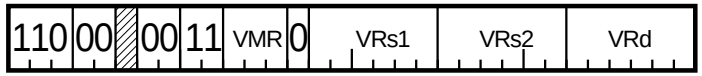
Description: src2 × src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

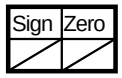
Mnemonic: VFDIV VRs1,VRs2,VRd{,VMR}

Instruction: Vector floating point division

Format:



Flags:

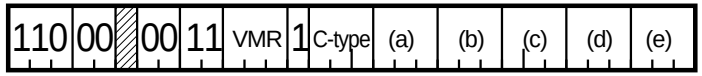


Description: VRs2 ÷ VRs1 -> VRd ; using vector mask register VMR

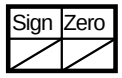
Mnemonic: CVFDIV src1,src2,dstn{,VMR}

Instruction: Vector floating point division with chaining

Format:



Flags:



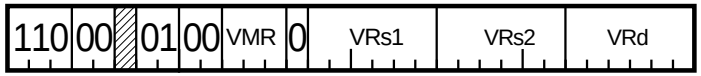
Description: src2 ÷ src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

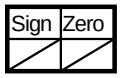
Mnemonic: VADD VRs1,VRs2,VRd{,VMR}

Instruction: Vector integer addition

Format:



Flags:



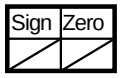
Description: VRs2 + VRs1 -> VRd ; using vector mask register VMR

Mnemonic: CVADD src1,src2,dstn{,VMR}

Instruction: Vector integer addition with chaining

Format:

Flags:



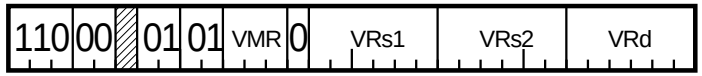
Description: src2 + src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

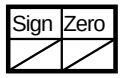
Mnemonic: VSUB VRs1,VRs2,VRd{,VMR}

Instruction: Vector integer subtract

Format:



Flags:



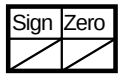
Description: VRs2 - VRs1 -> VRd ; using vector mask register VMR

Mnemonic: CVSUB src1,src2,dstn{,VMR}

Instruction: Vector integer subtract with chaining

Format:

Flags:



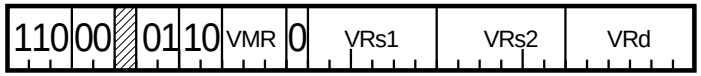
Description: src2 - src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

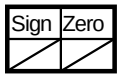
Mnemonic: VMUL VRs1,VRs2,VRd{,VMR}

Instruction: Vector integer multiply

Format:



Flags:



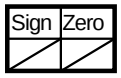
Description: VRs2 × VRs1 -> VRd ; using vector mask register VMR

Mnemonic: CVMUL src1,src2,dstn{,VMR}

Instruction: Vector integer multiply with chaining

Format:

Flags:



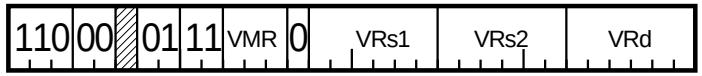
Description: src2 × src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

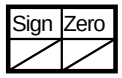
Mnemonic: VDIV VRs1,VRs2,VRd{,VMR}

Instruction: Vector integer division

Format:



Flags:

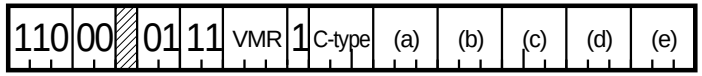


Description: VRs2 ÷ VRs1 -> VRd ; using vector mask register VMR

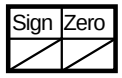
Mnemonic: CVDIV src1,src2,dstn{,VMR}

Instruction: Vector integer division with chaining

Format:



Flags:



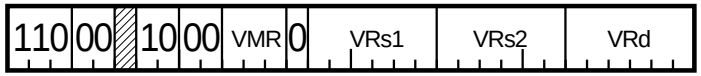
Description: src2 ÷ src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

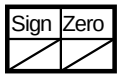
Mnemonic: VAND VRs1,VRs2,VRd{,VMR}

Instruction: Vector logical AND

Format:



Flags:

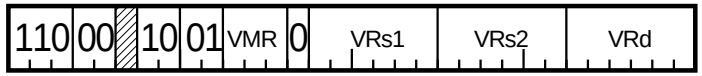


Description: VRs2 & VRs1 -> VRd ; using vector mask register VMR

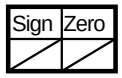
Mnemonic: VOR VRs1,VRs2,VRd{,VMR}

Instruction: Vector logical OR

Format:



Flags:

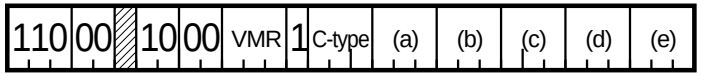


Description: VRs2 | VRs1 -> VRd ; using vector mask register VMR

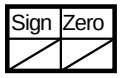
Mnemonic: CVAND src1,src2,dstn{,VMR}

Instruction: Vector logical AND with chaining

Format:



Flags:



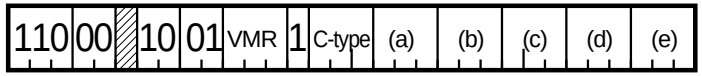
Description: src2 & src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

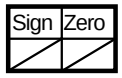
Mnemonic: CVOR src1,src2,dstn{,VMR}

Instruction: Vector logical OR with chaining

Format:



Flags:



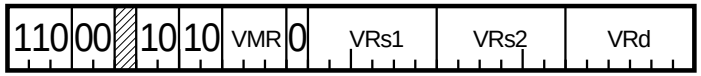
Description: src2 | src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

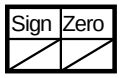
Mnemonic: VXOR VRs1,VRs2,VRd{,VMR}

Instruction: Vector logical XOR

Format:



Flags:

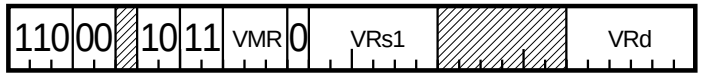


Description: VRs2 ^ VRs1 -> VRd ; using vector mask register VMR

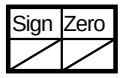
Mnemonic: VNOT VRs1,VRd{,VMR}

Instruction: Vector logical NOT

Format:



Flags:

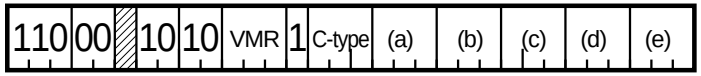


Description: ~VRs1 -> VRd ; using vector mask register VMR

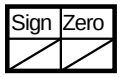
Mnemonic: CVXOR src1,src2,dstn{,VMR}

Instruction: Vector logical XOR with chaining

Format:



Flags:



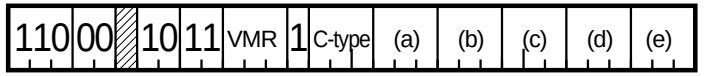
Description: src2 ^ src1 -> dstn ; using vector mask register VMR

src1	src2	dstn	
VRs1	VRs2	CNd	Ctype=001,(a)(b)=VRs1,(c)(d)=VRs2,(e)=CNd
VRs1	CNs2	VRd	Ctype=010,(a)(b)=VRs1,(c)=CNs2,(d)(e)=VRd
VRs1	CNs2	CNd	Ctype=011,(a)(b)=VRs1,(c)=CNs2,(e)=CNd
CNs1	VRs2	VRd	Ctype=100,(a)=CNs1,(b)(c)=VRs2,(d)(e)=VRd
CNs1	VRs2	CNd	Ctype=101,(a)=CNs1,(b)(c)=VRs2,(e)=CNd
CNs1	CNs2	VRd	Ctype=110,(a)=CNs1,(c)=CNs2,(d)(e)=VRd
CNs1	CNs2	CNd	Ctype=111,(a)=CNs1,(c)=CNs2,(e)=CNd

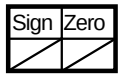
Mnemonic: CVNOT src1,dstn{,VMR}

Instruction: Vector logical NOT with chaining

Format:



Flags:



Description: ~src1 -> dstn ; using vector mask register VMR

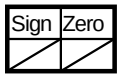
src1	dstn	
VRs1	CNd	Ctype=001,(a)(b)=VRs1,(e)=CNd
CNs1	VRd	Ctype=100,(a)=CNs1,(d)(e)=VRd
CNs1	CNd	Ctype=101,(a)=CNs1,(e)=CNd

Mnemonic: VBCAST SRs',VRd{,VMR}

Instruction: Vector broadcast

Format:

Flags:

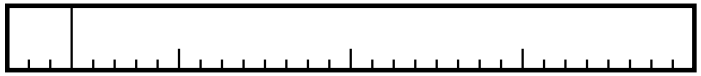


Description: SRs' -> VRd (broadcast) ; using vector mask register VMR

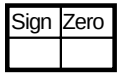
Mnemonic:

Instruction:

Format:



Flags:



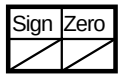
Description:

Mnemonic: VSRL SRs',VRs,VRd{,VMR}

Instruction: Vector shift right logical

Format:

Flags:



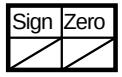
Description: VRs >> SRs' -> VRd ; using vector mask register VMR

Mnemonic: VSLL SRs',VRs,VRd{,VMR}

Instruction: Vector shift left logical

Format:

Flags:



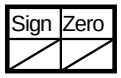
Description: VRs << SRs' -> VRd ; using vector mask register VMR

Mnemonic: VLIR VRd,SRb,SRi{,VMR}

Instruction: Vector load, increment is register

Format:

Flags:



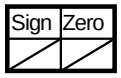
Description: for(i=0;i<128;i++) M[SRb+SRi×i] -> VRd[i]
using vector mask register VMR

Mnemonic: VLII VRd,SRb,Immed{,VMR}

Instruction: Vector load, increment is immediate

Format:

Flags:



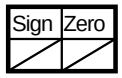
Description: for(i=0;i<128;i++) M[SRb+Immed×i] -> VRd[i]
using vector mask register VMR

Mnemonic: VSIR VRs,SRb,SRi{,VMR}

Instruction: Vector store, increment is register

Format:

Flags:



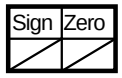
Description: for(i=0;i<128;i++) VRs[i] -> M[SRb+SRi×i]
using vector mask register VMR

Mnemonic: VSII VRs,SRb,Immed{,VMR}

Instruction: Vector store, increment is immediate

Format:

Flags:



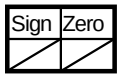
Description: for(i=0;i<128;i++) VRs[i] -> M[SRb+Immed×i]
using vector mask register VMR

Mnemonic: LVLD VRl,VRd{,VMR}

Instruction: List vector vector load

Format:

Flags:



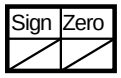
Description: for(i=0;i<128;i++) M[VRl[i]] -> VRd[i]
using vector mask register VMR

Mnemonic: LVST VRl,VRs{,VMR}

Instruction: List vector vector store

Format:

Flags:

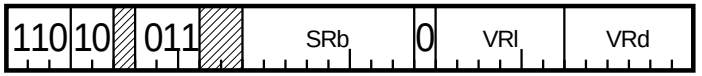


Description: for(i=0;i<128;i++) VRs[i] -> M[VRl[i]]
using vector mask register VMR

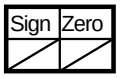
Mnemonic: LVBLD VRl,VRd,SRb

Instruction: List vector-based vector load

Format:



Flags:

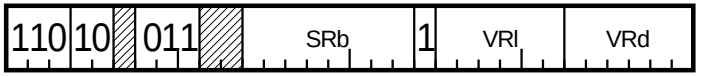


Description: for(i=0;i<128;i++) M[VRl[i]+SRb] -> VRd[i]

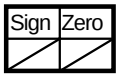
Mnemonic: LVBST VRl,VRs,SRb

Instruction: List vector-based vector store

Format:



Flags:

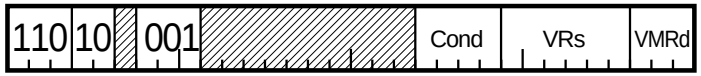


Description: for(i=0;i<128;i++) VRs[i] -> M[VRl[i]+SRb]

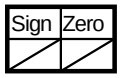
Mnemonic: VMRSET Cond,VRs,VMRd

Instruction: Vector mask register set

Format:



Flags:



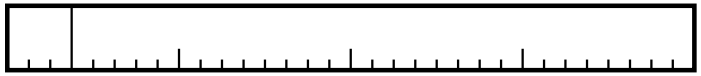
Description: for(i=0;i<128;i++) if cond(VRs[i]) then TRUE -> VMRd[i]
else FALSE -> VMRd[i]

Cond:
0000 - Floating zero (FZ) 1000 - Integer zero (IZ)
0001 - Floating nonzero (FNZ) 1001 - Integer nonzero (INZ)
0010 - Floating plus (FP) 1010 - Integer plus (IP)
0011 - Floating nonplus (FNP) 1011 - Integer nonplus (INP)
0100 - Floating minus (FM) 1100 - Integer minus (IM)
0100 - Floating nonminus (FNM) 1100 - Integer nonminus (INM)

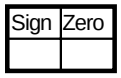
Mnemonic:

Instruction:

Format:



Flags:

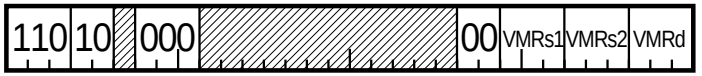


Description:

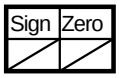
Mnemonic: VMRAND VMRs1,VMRs2,VMRd

Instruction: Vector mask register AND

Format:



Flags:

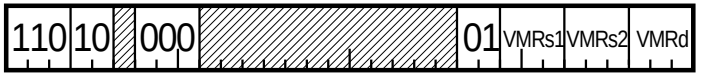


Description: VMRs1 & VMRs2 -> VMRd

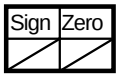
Mnemonic: VMROR VMRs1,VMRs2,VMRd

Instruction: Vector mask register OR

Format:



Flags:

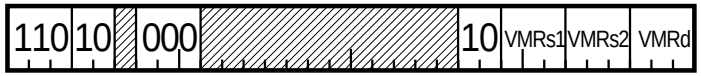


Description: VMRs1 | VMRs2 -> VMRd

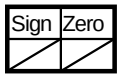
Mnemonic: VMRXOR VMRs1,VMRs2,VMRd

Instruction: Vector mask register XOR

Format:



Flags:

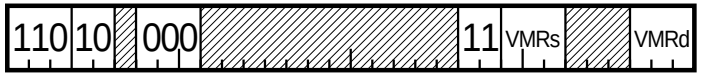


Description: VMRs1 ^ VMRs2 -> VMRd

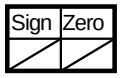
Mnemonic: VMRNOT VMRs,VMRd

Instruction: Vector mask register NOT

Format:



Flags:



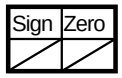
Description: ~VMRs -> VMRd

Mnemonic: VINT VRs,VRd{,VMR}

Instruction: Vector floating point to integer conversion

Format:

Flags:



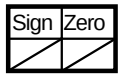
Description: (int)VRs -> VRd ; using vector mask register VMR

Mnemonic: VFLT VRs,VRd{,VMR}

Instruction: Vector integer to floating point conversion

Format:

Flags:



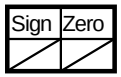
Description: (float)VRs -> VRd ; using vector mask register VMR

Mnemonic: MOVVS VRs,SRp,SRd

Instruction: Move vector to scalar register

Format:

Flags:



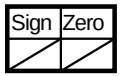
Description: VRs[SRp] -> SRd

Mnemonic: MOVSV VRd,SRp,SRs

Instruction: Move scalar to vector register

Format:

Flags:



Description: SRs -> VRd[SRp]