

# Vector Floating-point Processing Unit (VFPU) Instruction Manual

---

Please note that the specifications contained in this document are preliminary and subject to change without prior notice.

© 2004 Sony Computer Entertainment Inc.  
All Rights Reserved.  
SCE Confidential

# Table of Contents

VFPU Instructions .....	10
bvf .....	11
bvfl .....	12
bvt .....	13
bvtl .....	14
lv.s .....	15
lv.q .....	16
lvl.q .....	17
lvr.q .....	19
mfv .....	21
mfvc .....	22
mtv .....	23
mtvc .....	24
sv.s .....	25
sv.q .....	26
sv.q .....	27
svl.q .....	29
svr.q .....	31
vabs.s .....	33
vabs.p .....	34
vabs.t .....	35
vabs.q .....	36
vadd.s .....	37
vadd.p .....	38
vadd.t .....	39
vadd.q .....	40
vasin.s .....	41
vasin.p .....	43
vasin.t .....	45
vasin.q .....	47
vavg.p .....	49
vavg.t .....	50

vavg.q.....	51
vbfi1.p .....	52
vbfi1.q .....	53
vbfi2.q .....	54
vcmovf.s .....	55
vcmovf.p.....	57
vcmovf.t .....	59
vcmovf.q.....	61
vcmovt.s.....	63
vcmovt.p .....	65
vcmovt.t .....	67
vcmovt.q .....	69
vcmp.s.....	71
vcmp.p.....	73
vcmp.t .....	75
vcmp.q.....	77
vcos.s.....	79
vcos.p .....	81
vcos.t .....	83
vcos.q .....	85
vers.t .....	87
versp.t .....	88
vest.s .....	89
vest.p.....	91
vest.t .....	93
vest.q.....	95
vdet.p .....	97
vdiv.s .....	98
vdiv.p.....	99
vdiv.t .....	100
vdiv.q.....	101
vdot.p .....	102
vdot.t.....	103
vdot.q .....	104
vexp2.s.....	105
vexp2.p.....	107

vexp2.t .....	109
vexp2.q.....	111
vf2h.p .....	113
vf2h.q .....	114
vf2id.s .....	115
vf2id.p.....	116
vf2id.t.....	117
vf2id.q.....	119
vf2in.s .....	121
vf2in.p.....	122
vf2in.t.....	124
vf2in.q.....	126
vf2iu.s .....	128
vf2iu.p.....	129
vf2iu.t.....	130
vf2iu.q.....	132
vf2iz.s.....	134
vf2iz.p .....	135
vf2iz.t .....	136
vf2iz.q .....	138
vfad.p .....	140
vfad.t.....	141
vfad.q .....	142
vfim.s .....	143
vflush .....	144
vh2f.s .....	145
vh2f.p.....	146
vhdp.p.....	147
vhdp.t.....	148
vhdp.q .....	149
vhtfm2.p .....	150
vhtfm3.t.....	151
vhtfm4.q .....	152
vi2c.q.....	153
vi2f.s .....	154
vi2f.p.....	155

vi2f.t.....	156
vi2f.q.....	157
vi2s.p.....	158
vi2s.q.....	159
vi2uc.q.....	160
vi2us.p.....	162
vi2us.q.....	163
vidt.p.....	165
vidt.q.....	166
viim.s.....	167
vlgb.s.....	168
vlog2.s.....	170
vlog2.p.....	172
vlog2.t.....	174
vlog2.q.....	176
vmax.s.....	178
vmax.p.....	179
vmax.t.....	180
vmax.q.....	181
vmfvc.....	182
vmidt.p.....	183
vmidt.t.....	184
vmidt.q.....	185
vmin.s.....	187
vmin.p.....	188
vmin.t.....	189
vmin.q.....	190
vmmov.p.....	191
vmmov.t.....	192
vmmov.q.....	193
vmmul.p.....	195
vmmul.t.....	197
vmmul.q.....	199
vmone.p.....	201
vmone.t.....	202
vmone.q.....	203

vmov.s .....	205
vmov.p.....	206
vmov.t.....	207
vmov.q.....	208
vmscl.p.....	209
vmscl.t .....	210
vmscl.q.....	211
vmtvc .....	213
vmul.s .....	214
vmul.p.....	215
vmul.t.....	216
vmul.q.....	217
vmzero.p .....	218
vmzero.t.....	219
vmzero.q .....	220
vneg.s.....	222
vneg.p .....	223
vneg.t .....	224
vneg.q.....	225
vnop .....	226
vnrcp.s .....	227
vnrcp.p.....	229
vnrcp.t.....	231
vnrcp.q.....	233
vnsin.s.....	235
vnsin.p .....	236
vnsin.t.....	237
vnsin.q .....	238
vocp.s .....	239
vocp.p.....	240
vocp.t.....	241
vocp.q .....	242
vone.s.....	243
vone.p.....	244
vone.t .....	245
vone.q.....	246

vpfxd .....	247
vpfxs.....	248
vpfxt.....	249
vqmul.q .....	250
vrcp.s.....	251
vrcp.p .....	252
vrcp.t.....	254
vrcp.q .....	256
vrex2.s .....	258
vrex2.p .....	260
vrex2.t.....	262
vrex2.q .....	264
vrndf1.s.....	266
vrndf1.p .....	267
vrndf1.t.....	268
vrndf1.q .....	269
vrndf2.s.....	270
vrndf2.p .....	271
vrndf2.t.....	272
vrndf2.q .....	273
vrndi.s.....	274
vrndi.p .....	275
vrndi.t.....	276
vrndi.q .....	277
vrnds.s .....	278
vrot.p.....	279
vrot.t .....	281
vrot.q.....	283
vrsq.s.....	285
vrsq.p .....	287
vrsq.t.....	289
vrsq.q .....	291
vs2i.s .....	293
vs2i.p.....	294
vsat0.s.....	295
vsat0.p .....	296

vsat0.t.....	297
vsat0.q .....	298
vsat1.s.....	299
vsat1.p .....	300
vsat1.t.....	301
vsat1.q .....	302
vsbn.s.....	303
vsbz.s .....	305
vscl.p.....	306
vscl.t.....	307
vscl.q .....	308
vscmp.s .....	309
vscmp.p.....	310
vscmp.t .....	311
vscmp.q .....	312
vsge.s .....	313
vsge.p.....	314
vsge.t.....	315
vsge.q .....	316
vsgn.s.....	317
vsgn.p.....	318
vsgn.t .....	319
vsgn.q.....	320
vsin.s.....	321
vsin.p .....	323
vsin.t .....	325
vsin.q .....	327
vslt.s.....	329
vslt.p .....	330
vslt.t.....	331
vslt.q .....	332
vsocp.s.....	333
vsocp.p .....	334
vsqrt.s .....	335
vsqrt.p.....	337
vsqrt.t .....	339



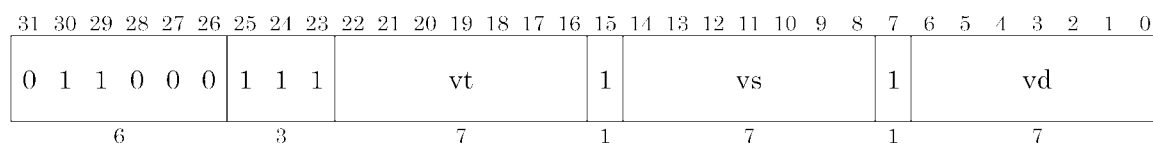
vsqrt.q.....	341
vsrt1.q.....	343
vsrt2.q.....	344
vsrt3.q.....	345
vsrt4.q.....	346
vsub.s.....	347
vsub.p.....	348
vsub.t.....	349
vsub.q.....	350
vsync.....	351
vt4444.q.....	352
vt5551.q.....	354
vt5650.q.....	356
vtfm2.p.....	358
vtfm3.t.....	359
vtfm4.q.....	360
vus2i.s.....	361
vus2i.p.....	362
vwnb.s.....	363
vzero.s.....	364
vzero.p.....	365
vzero.t.....	366
vzero.q.....	367

## VFPU Instructions

---

## vdiv.q

Divide Quad Word



VFPU

### Syntax:

```
vdiv.q vd, vs, vt
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 59      pitch : 56

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

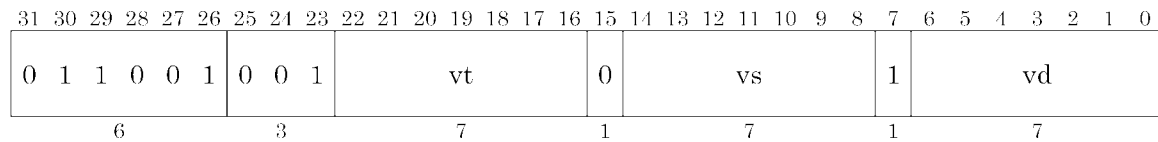
Four elements from the matrix registers indicated by vs are divided by four elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] / t[0];
d[1] <- s[1] / t[1];
d[2] <- s[2] / t[2];
d[3] <- s[3] / t[3];
WriteMatrix( QUADWORD, vd, d );
```

## vdot.p

Dot Product Pair Word



VFPU

### Syntax:

```
vdot.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

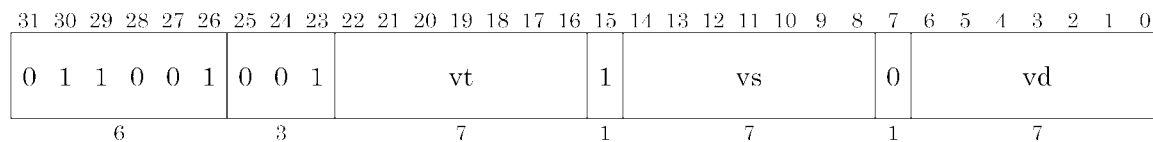
The dot product of two elements from the matrix registers indicated by vs and two elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + s[1] * t[1];
WriteMatrix( SINGLEWORD, vd, d );
```

## vdot.t

### Dot Product Triple Word



VFPU

#### Syntax:

```
vdot.t vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

#### Description:

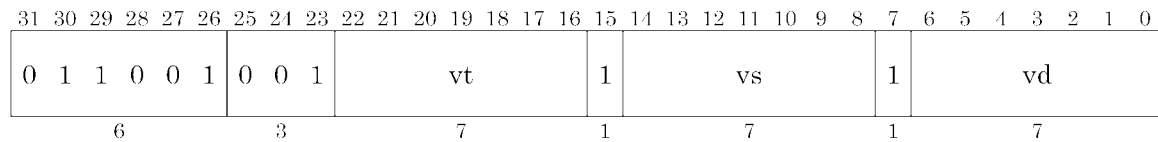
The dot product of three elements from the matrix registers indicated by vs and three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + s[1] * t[1];
d[0] <- d[0] + s[2] * t[2];
WriteMatrix( SINGLEWORD, vd, d );
```

## vdot.q

Dot Product Quad Word



VFPU

### Syntax:

```
vdot.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

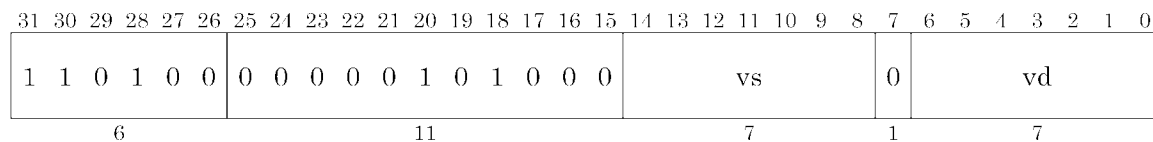
The dot product of four elements from the matrix registers indicated by vs and four elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + s[1] * t[1];
d[0] <- d[0] + s[2] * t[2];
d[0] <- d[0] + s[3] * t[3];
WriteMatrix( SINGLEWORD, vd, d );
```

## vexp2.s

Exponential base 2 Single Word



VFPU

### Syntax:

vexp2.s vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The base 2 exponential of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

$$\text{approx\_exp2}(-0.0) = +1.0$$

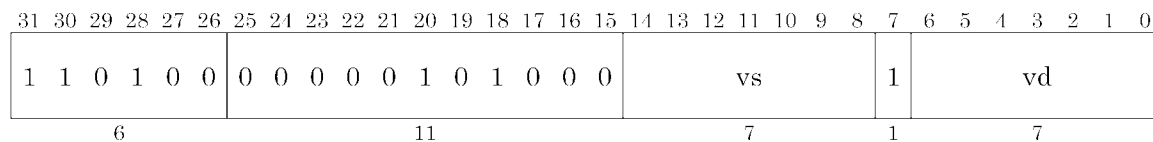
**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_exp2( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vexp2.p

Exponential base 2 Pair Word



VFPU

### Syntax:

vexp2.p vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 exponentials of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

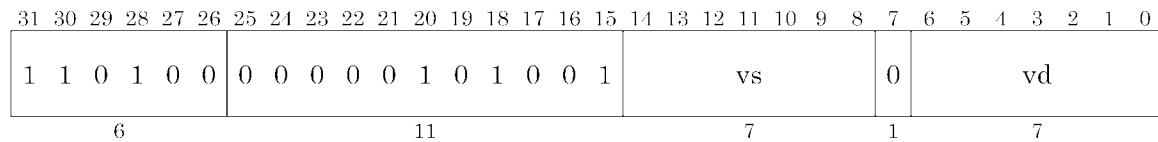
$$\text{approx\_exp2}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_exp2( s[0] );
d[1] <- approx_exp2( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vexp2.t

Exponential base 2 Triple Word



VFPU

### Syntax:

vexp2.t vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 exponentials of the floating-point values of three elements from the matrix registers indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

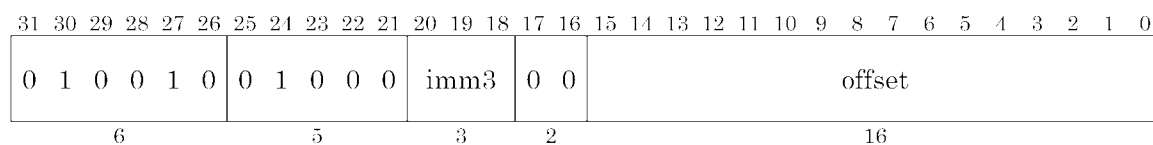
$$\text{approx\_exp2}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_exp2( s[0] );
d[1] <- approx_exp2( s[1] );
d[2] <- approx_exp2( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## bvf

Branch on VFPU False



VFPU

### Syntax:

```
bvf imm3, offset
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

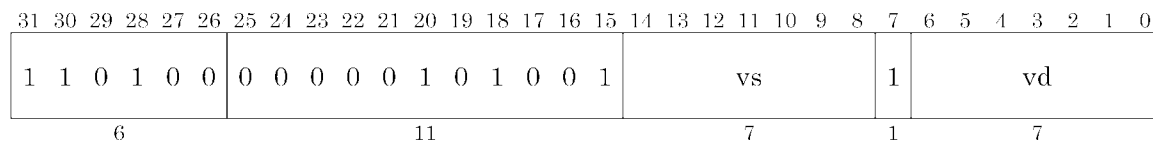
If the value of the VFPU\_CC control register bit indicated by the imm3 field is false (0), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

### Operation:

```
I+0: condition <- (VFPU_CC[imm3] == 0);
      target_offset <- sign_extend(offset<<2)
I+1: if condition then
      PC <- PC + target_offset;
endif
```

## vexp2.q

Exponential base 2 Quad Word



VFPU

### Syntax:

vexp2.q vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 exponentials of the floating-point values of four elements from the matrix registers indicated by vs are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

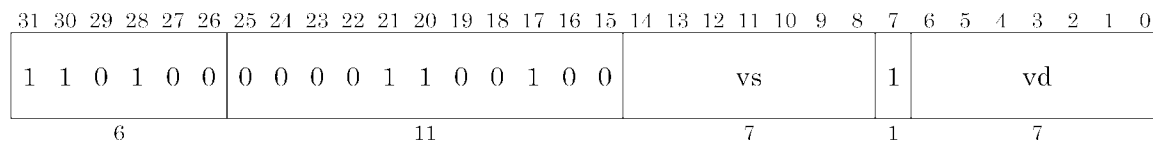
$$\text{approx\_exp2}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_exp2( s[0] );
d[1] <- approx_exp2( s[1] );
d[2] <- approx_exp2( s[2] );
d[3] <- approx_exp2( s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vf2h.p

Convert float to float16 Pair Word



VFPU

### Syntax:

```
vf2h.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The single-precision floating-point values of two elements from the matrix registers indicated by vs are converted to half-precision floating-point numbers and packed into 32 bits. The 32-bit result is stored at the location in the matrix register file indicated by vd.

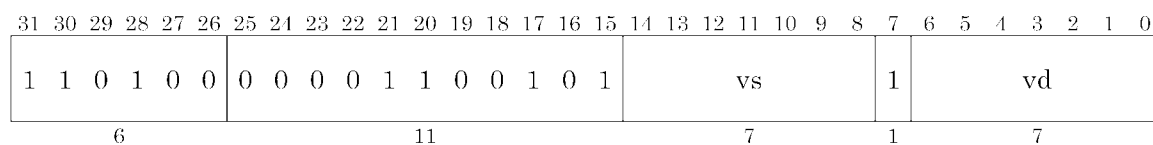
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0].l <- float_to_float16( s[0] );
d[0].u <- float_to_float16( s[1] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vf2h.q

Convert float to float16 Quad Word



VFPU

### Syntax:

```
vf2h.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

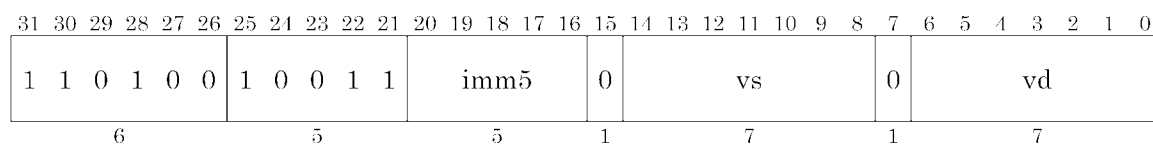
The single-precision floating-point values of four elements from the matrix registers indicated by vs are converted to half-precision floating-point numbers and packed into 64 bits. The 64-bit result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0].l <- float_to_float16( s[0] );
d[0].u <- float_to_float16( s[1] );
d[1].l <- float_to_float16( s[2] );
d[1].u <- float_to_float16( s[3] );
WriteMatrix( PAIRWORD, vd, d );
```

## vf2id.s

Round to smallest integer from float with Scaling Single Word



VFPU

### Syntax:

```
vf2id.s vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point value of one element from the matrix register indicated by vs is multiplied by 2 raised to the imm5 power and rounded to the largest integer less than or equal to the argument. The one-element integer result is stored at the location in the matrix register file indicated by vd. Special solutions are as follows.

*floor(nan)* = 0x7FFFFFFF

*floor(+inf)* = 0x7FFFFFFF

*floor(-inf)* = 0x80000000

*floor(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

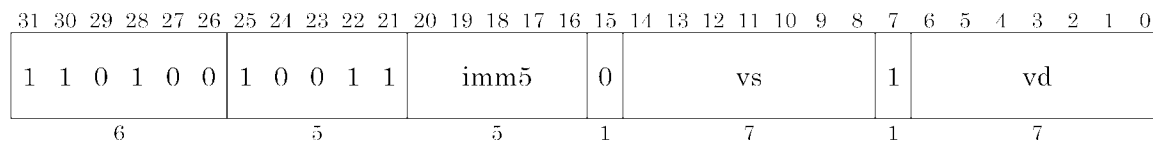
*floor(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- floor( s[0] * (1<<imm5) );
WriteMatrix( SINGLEWORD, vd, d );
```

## vf2id.p

Round to smallest integer from float with Scaling Pair Word



VFPU

### Syntax:

```
vf2id.p vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5                  pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of two elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the largest integer less than or equal to the arguments. The two-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*floor(nan)* = 0x7FFFFFFF

*floor(+inf)* = 0x7FFFFFFF

*floor(-inf)* = 0x80000000

*floor(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

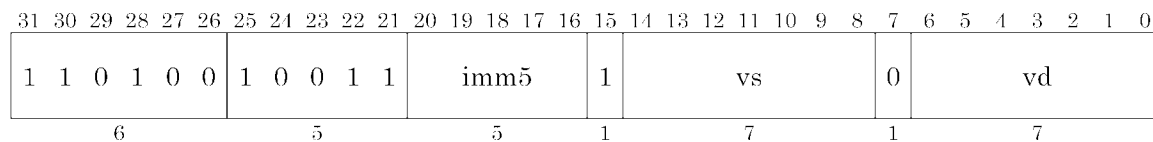
*floor(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- floor( s[0] * (1<<imm5) );
d[1] <- floor( s[1] * (1<<imm5) );
WriteMatrix( PAIRWORD, vd, d );
```

## vf2id.t

Round to smallest integer from float with Scaling Triple Word



VFPU

### Syntax:

```
vf2id.t vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of three elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the largest integer less than or equal to the arguments. The three-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*floor(nan)* = 0x7FFFFFFF

*floor(+inf)* = 0x7FFFFFFF

*floor(-inf)* = 0x80000000

*floor(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*floor(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

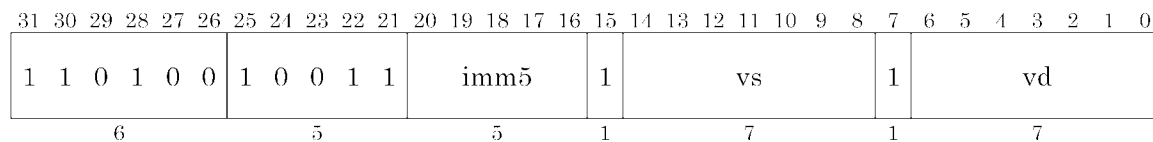
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- floor( s[0] * (1<<imm5) );
d[1] <- floor( s[1] * (1<<imm5) );
d[2] <- floor( s[2] * (1<<imm5) );
```

```
WriteMatrix( TRIPLEWORD, vd, d );
```

## vf2id.q

Round to smallest integer from float with Scaling Quad Word



VFPU

### Syntax:

```
vf2id.q vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfmt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of four elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the largest integer less than or equal to the arguments. The four-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*floor(nan)* = 0x7FFFFFFF

*floor(+inf)* = 0x7FFFFFFF

*floor(-inf)* = 0x80000000

*floor(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*floor(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

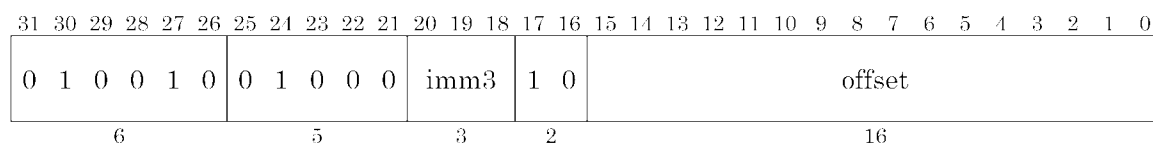
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- floor( s[0] * (1<<imm5) );
d[1] <- floor( s[1] * (1<<imm5) );
d[2] <- floor( s[2] * (1<<imm5) );
```

```
d[3] <- floor( s[3] * (1<<imm5) );
WriteMatrix( QUADWORD, vd, d );
```

## bvfl

Branch on VFPU False Likely



VFPU

### Syntax:

```
bvfl imm3, offset
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

If the value of the VFPU\_CC control register bit indicated by the imm3 field is false (0), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

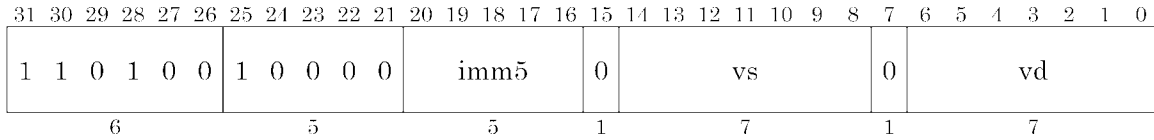
### Operation:

```
I+0: condition <- (VFPU_CC[imm3] == 0);
      target_offset <- sign_extend(offset<<2)
I+1: if condition then
      PC <- PC + target_offset;
      else
      NullifyCurrentInstruction();
      endif
```



## vf2in.s

Round to nearest integer from float with Scaling Single Word



VFPU

### Syntax:

```
vf2in.s vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point value of one element from the matrix register indicated by vs is multiplied by 2 raised to the imm5 power and rounded to the nearest integer. The one-element integer result is stored at the location in the matrix register file indicated by vd. Special solutions are as follows.

$\text{rint}(\text{nan}) = 0x7FFFFFFF$

$\text{rint}(+\text{inf}) = 0x7FFFFFFF$

$\text{rint}(-\text{inf}) = 0x80000000$

$\text{rint}(x) = 0x7FFFFFFF ; +2^{31} \leq x < +\text{inf}$

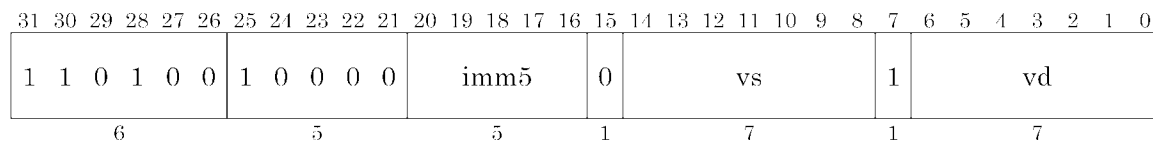
$\text{rint}(x) = 0x80000000 ; -\text{inf} < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- rint( s[0] * (1<<imm5) );
WriteMatrix( SINGLEWORD, vd, d );
```

## vf2in.p

Round to nearest integer from float with Scaling Pair Word



VFPU

### Syntax:

```
vf2in.p vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of two elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the nearest integer. The two-element integer result is stored at locations in the matrix register file indicated by vd.

Special solutions are as follows.

$\text{rint}(\text{nan}) = 0x7FFFFFFF$

$\text{rint}(+\text{inf}) = 0x7FFFFFFF$

$\text{rint}(-\text{inf}) = 0x80000000$

$\text{rint}(x) = 0x7FFFFFFF ; +2^{31} \leq x < +\text{inf}$

$\text{rint}(x) = 0x80000000 ; -\text{inf} < x < -2^{31}$

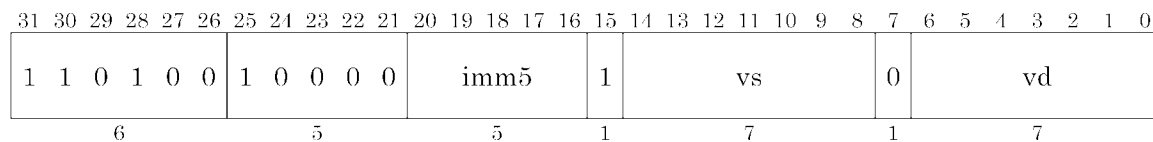
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- rint( s[0] * (1<<imm5) );
d[1] <- rint( s[1] * (1<<imm5) );
```

```
WriteMatrix( PAIRWORD, vd, d );
```

## vf2in.t

Round to nearest integer from float with Scaling Triple Word



VFPU

### Syntax:

```
vf2in.t vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of three elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the nearest integer. The three-element integer result is stored at locations in the matrix register file indicated by vd.

Special solutions are as follows.

$\text{rint}(\text{nan}) = 0x7FFFFFFF$

$\text{rint}(+\text{inf}) = 0x7FFFFFFF$

$\text{rint}(-\text{inf}) = 0x80000000$

$\text{rint}(x) = 0x7FFFFFFF ; +2^{31} \leq x < +\text{inf}$

$\text{rint}(x) = 0x80000000 ; -\text{inf} < x < -2^{31}$

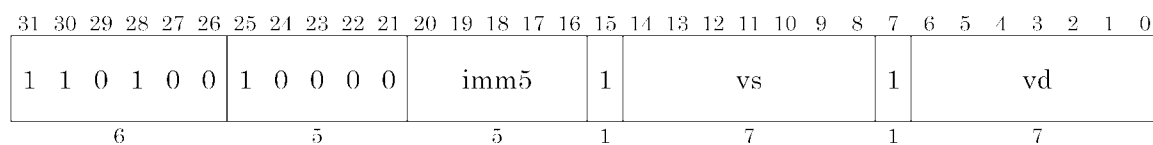
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- rint( s[0] * (1<<imm5) );
d[1] <- rint( s[1] * (1<<imm5) );
```

```
d[2] <- rint( s[2] * (1<<imm5) );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vf2in.q

Round to nearest integer from float with Scaling Quad Word



VFPU

### Syntax:

```
vf2in.q vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of four elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the nearest integer. The four-element integer result is stored at locations in the matrix register file indicated by vd.

Special solutions are as follows.

$\text{rint}(\text{nan}) = 0x7FFFFFFF$

$\text{rint}(+\text{inf}) = 0x7FFFFFFF$

$\text{rint}(-\text{inf}) = 0x80000000$

$\text{rint}(x) = 0x7FFFFFFF ; +2^{31} \leq x < +\text{inf}$

$\text{rint}(x) = 0x80000000 ; -\text{inf} < x < -2^{31}$

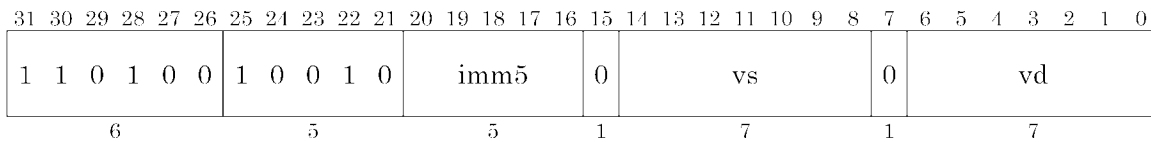
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- rint( s[0] * (1<<imm5) );
d[1] <- rint( s[1] * (1<<imm5) );
```

```
d[2] <- rint( s[2] * (1<<imm5) );
d[3] <- rint( s[3] * (1<<imm5) );
WriteMatrix( QUADWORD, vd, d );
```

## vf2iu.s

Round to largest integer from float with Scaling Single Word



VFPU

### Syntax:

```
vf2iu.s vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point value of one element from the matrix register indicated by vs is multiplied by 2 raised to the imm5 power and rounded to the smallest integer greater than or equal to the argument. The one-element integer result is stored at the location in the matrix register file indicated by vd. Special solutions are as follows.

*ceil(nan)* = 0x7FFFFFFF

*ceil(+inf)* = 0x7FFFFFFF

*ceil(-inf)* = 0x80000000

*ceil(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*ceil(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

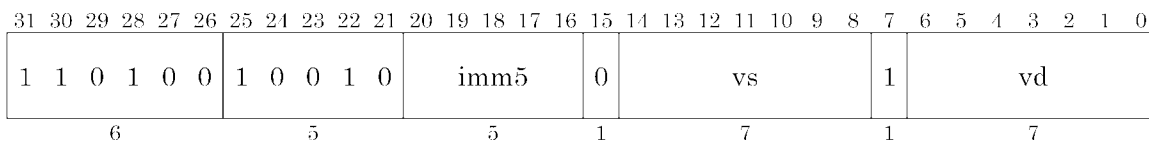
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- ceil( s[0] * (1<<imm5) );
WriteMatrix( SINGLEWORD, vd, d );
```



## vf2iu.p

Round to largest integer from float with Scaling Pair Word



VFPU

### Syntax:

```
vf2iu.p vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of two elements from the matrix registers indicated by *vs* are multiplied by 2 raised to the *imm5* power and rounded to the smallest integers greater than or equal to the arguments. The two-element integer result is stored at locations in the matrix register file indicated by *vd*. Special solutions are as follows.

*ceil(nan)* = 0x7FFFFFFF

*ceil(+inf)* = 0x7FFFFFFF

*ceil(-inf)* = 0x80000000

*ceil(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

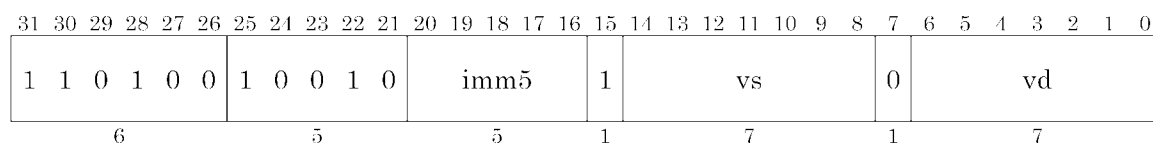
*ceil(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- ceil( s[0] * (1<<imm5) );
d[1] <- ceil( s[1] * (1<<imm5) );
WriteMatrix( PAIRWORD, vd, d );
```

## vf2iu.t

Round to largest integer from float with Scaling Triple Word



VFPU

### Syntax:

```
vf2iu.t vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of three elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the smallest integers greater than or equal to the arguments. The three-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*ceil(nan)* = 0x7FFFFFFF

*ceil(+inf)* = 0x7FFFFFFF

*ceil(-inf)* = 0x80000000

*ceil(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

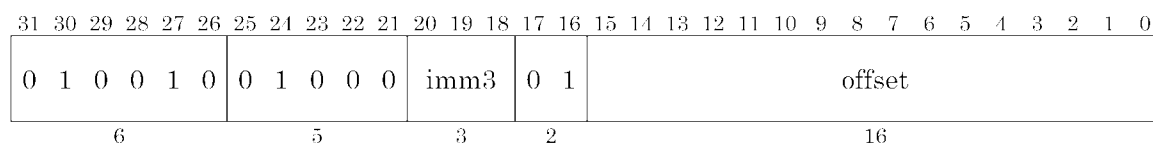
*ceil(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- ceil( s[0] * (1<<imm5) );
d[1] <- ceil( s[1] * (1<<imm5) );
d[2] <- ceil( s[2] * (1<<imm5) );
```

## bvt

Branch on VFPU True



VFPU

### Syntax:

```
bvt imm3, offset
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

If the value of the VFPU\_CC control register bit indicated by the imm3 field is true (1), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value.

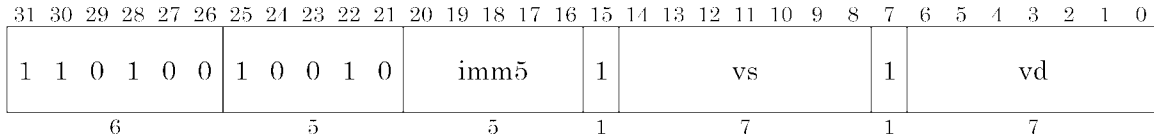
### Operation:

```
I+0: condition <- (VFPU_CC[imm3] == 1);
      target_offset <- sign_extend(offset<<2);
I+1: if condition then
      PC <- PC + target_offset;
      endif
```

```
WriteMatrix( TRIPLEWORD, vd, d );
```

## vf2iu.q

Round to largest integer from float with Scaling Quad Word



VFPU

### Syntax:

```
vf2iu.q vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of four elements from the matrix registers indicated by *vs* are multiplied by 2 raised to the *imm5* power and rounded to the smallest integers greater than or equal to the arguments. The four-element integer result is stored at locations in the matrix register file indicated by *vd*. Special solutions are as follows.

*ceil(nan)* = 0x7FFFFFFF

*ceil(+inf)* = 0x7FFFFFFF

*ceil(-inf)* = 0x80000000

*ceil(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*ceil(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

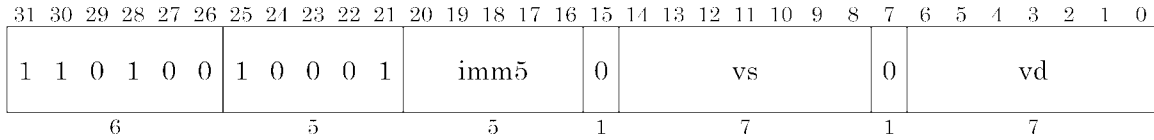
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- ceil( s[0] * (1<<imm5) );
d[1] <- ceil( s[1] * (1<<imm5) );
d[2] <- ceil( s[2] * (1<<imm5) );
```

```
d[3] <- ceil( s[3] * (1<<imm5) );
WriteMatrix( QUADWORD, vd, d );
```

## vf2iz.s

Round to zero integer from float with Scaling Single Word



VFPU

### Syntax:

```
vf2iz.s vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point value of one element from the matrix register indicated by vs is multiplied by 2 raised to the imm5 power and rounded to the integer value closest to zero. The one-element integer result is stored at the location in the matrix register file indicated by vd. Special solutions are as follows.

$\text{trunc}(\text{nan}) = 0x7FFFFFFF$

$\text{trunc}(+\text{inf}) = 0x7FFFFFFF$

$\text{trunc}(-\text{inf}) = 0x80000000$

$\text{trunc}(x) = 0x7FFFFFFF ; +2^{31} \leq x < +\text{inf}$

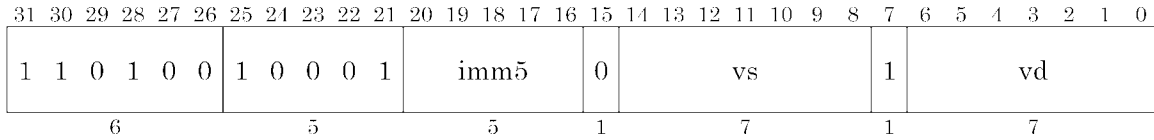
$\text{trunc}(x) = 0x80000000 ; -\text{inf} < x < -2^{31}$

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- trunc( s[0] * (1<<imm5) );
WriteMatrix( SINGLEWORD, vd, d );
```

## vf2iz.p

Round to zero integer from float with Scaling Pair Word



VFPU

### Syntax:

```
vf2iz.p vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5                  pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of two elements from the matrix registers indicated by *vs* are multiplied by 2 raised to the *imm5* power and rounded to the integer values closest to zero. The two-element integer result is stored at locations in the matrix register file indicated by *vd*. Special solutions are as follows.

*trunc(nan)* = 0x7FFFFFFF

*trunc(+inf)* = 0x7FFFFFFF

*trunc(-inf)* = 0x80000000

*trunc(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*trunc(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

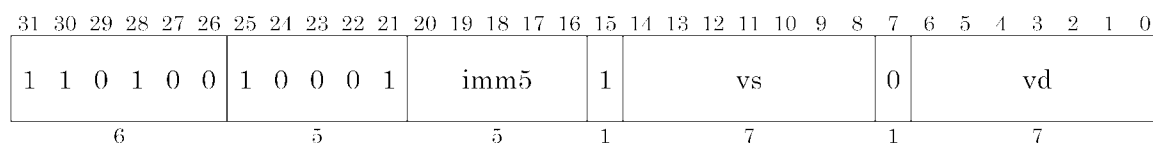
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- trunc( s[0] * (1<<imm5) );
d[1] <- trunc( s[1] * (1<<imm5) );
WriteMatrix( PAIRWORD, vd, d );
```



## vf2iz.t

Round to zero integer from float with Scaling Triple Word



VFPU

### Syntax:

```
vf2iz.t vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of three elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the integer values closest to zero. The three-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*trunc(nan)* = 0x7FFFFFFF

*trunc(+inf)* = 0x7FFFFFFF

*trunc(-inf)* = 0x80000000

*trunc(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*trunc(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

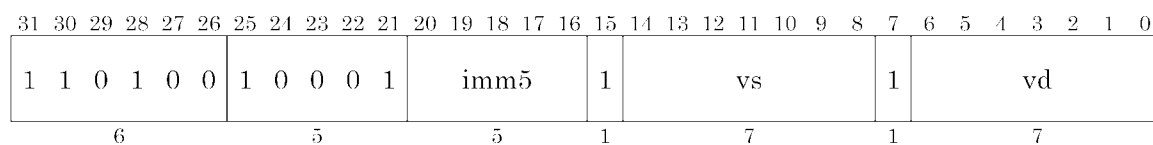
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- trunc( s[0] * (1<<imm5) );
d[1] <- trunc( s[1] * (1<<imm5) );
d[2] <- trunc( s[2] * (1<<imm5) );
```

```
WriteMatrix( TRIPLEWORD, vd, d );
```

## vf2iz.q

Round to zero integer from float with Scaling Quad Word



VFPU

### Syntax:

```
vf2iz.q vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Only write mask is valid

### Description:

The floating-point values of four elements from the matrix registers indicated by vs are multiplied by 2 raised to the imm5 power and rounded to the integer values closest to zero. The four-element integer result is stored at locations in the matrix register file indicated by vd. Special solutions are as follows.

*trunc(nan)* = 0x7FFFFFFF

*trunc(+inf)* = 0x7FFFFFFF

*trunc(-inf)* = 0x80000000

*trunc(x)* = 0x7FFFFFFF ;  $+2^{31} \leq x < +inf$

*trunc(x)* = 0x80000000 ;  $-inf < x < -2^{31}$

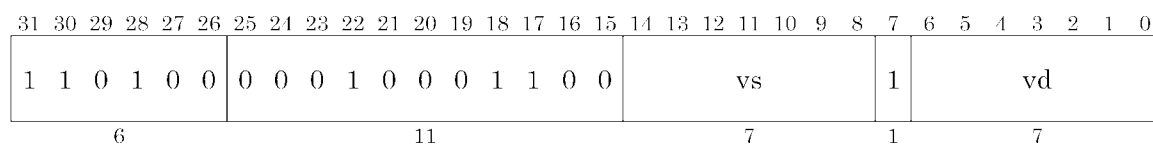
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- trunc( s[0] * (1<<imm5) );
d[1] <- trunc( s[1] * (1<<imm5) );
d[2] <- trunc( s[2] * (1<<imm5) );
```

```
d[3] <- trunc( s[3] * (1<<imm5) );
WriteMatrix( QUADWORD, vd, d );
```

## vfad.p

### Funnel Add Pair Word



VFPU

#### Syntax:

```
vfad.p vd, vs
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

#### Description:

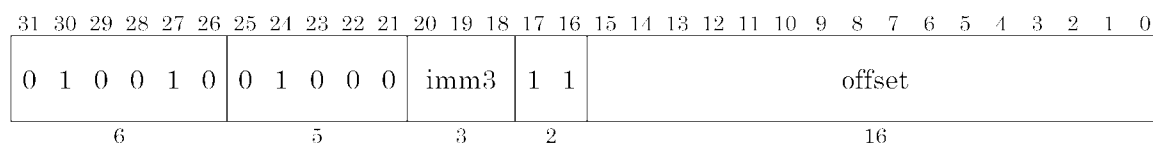
Two elements from the matrix registers indicated by vs are added together as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- s[0];
d[0] <- d[0] + s[1];
WriteMatrix( SINGLEWORD, vd, d );
```

## bvtl

Branch on VFPU True Likely



VFPU

### Syntax:

```
bvtl imm3, offset
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

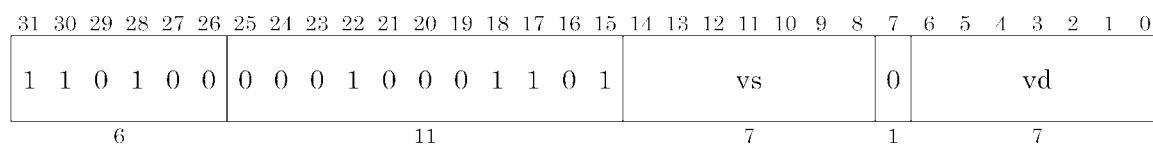
If the value of the VFPU\_CC control register bit indicated by the imm3 field is true (1), the program branches with a one instruction delay to the branch target address. The branch target address is the sum of the PC and the 16-bit offset after it is shifted left two bits and sign-extended to a 32 bit value. If the branch is not taken, the instruction in the branch delay slot is discarded.

### Operation:

```
I+0: condition <- (VFPU_CC[imm3] == 1 );
      target_offset <- sign_extend(offset<<2);
I+1: if condition then
      PC <- PC + target_offset;
      else
      NullifyCurrentInstruction();
      endif
```

## vfad.t

### Funnel Add Triple Word



VFPU

#### Syntax:

```
vfad.t vd, vs
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

#### Description:

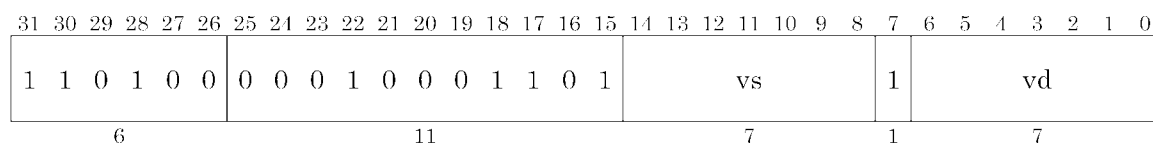
Three elements from the matrix registers indicated by vs are added together as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- s[0];
d[0] <- d[0] + s[1];
d[0] <- d[0] + s[2];
WriteMatrix( SINGLEWORD, vd, d );
```

## vfad.q

### Funnel Add Quad Word



VFPU

#### Syntax:

```
vfad.q vd, vs
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

#### Description:

Four elements from the matrix registers indicated by vs are added together as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

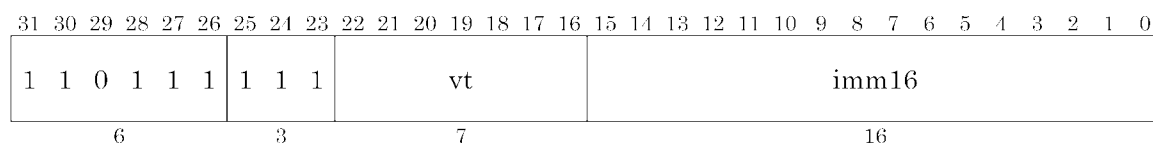
#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- s[0];
d[0] <- d[0] + s[1];
d[0] <- d[0] + s[2];
d[0] <- d[0] + s[3];
WriteMatrix( SINGLEWORD, vd, d );
```



## vfim.s

Convert float16 immediate to float Single Word



VFPU

### Syntax:

```
vfim.s vt, imm16
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

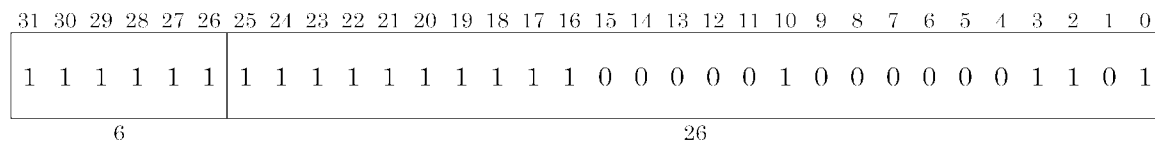
The half-precision floating-point value indicated by imm16 is converted to a single-precision floating-point value and stored at the location in the matrix register file indicated by vt.

### Operation:

```
f <- float16_to_float( imm16 );
WriteMatrix( SINGLEWORD, vt, f );
```

## vflush

Flush Write Buffer



VFPU

### Syntax:

`vflush`

### Instruction Type

Synchronization instruction

### Processing Time:

latency : 0      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

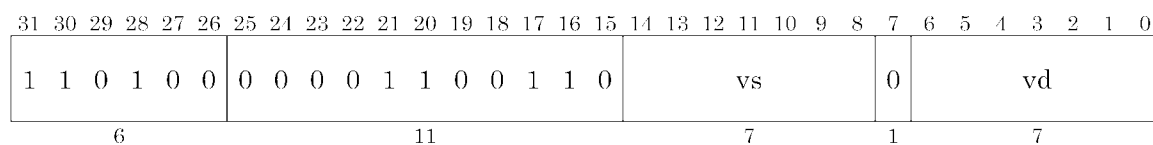
The write buffer is flushed. The VFPU pipeline stalls until the write buffer has emptied.

### Operation:

`Flush();`

## vh2f.s

Convert float16 to float Single Word



VFPU

### Syntax:

```
vh2f.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

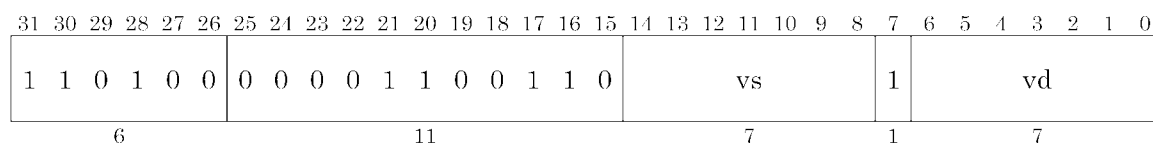
The half-precision floating-point values of two elements from the matrix registers indicated by vs are converted to single-precision floating-point numbers. The two-element floating-point result is stored in locations of the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- float16_to_float( s[0].l );
d[1] <- float16_to_float( s[0].u );
WriteMatrix( PAIRWORD, vd, d );
```

## vh2f.p

Convert float16 to float Pair Word



VFPU

### Syntax:

vh2f.p vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

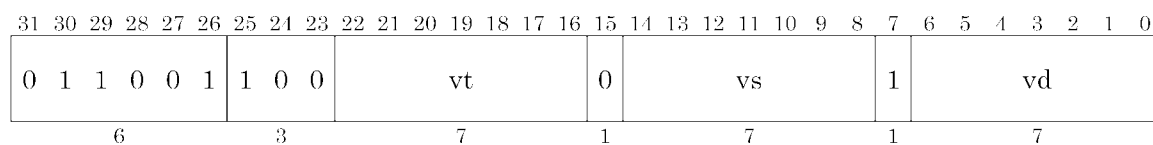
The half-precision floating-point values of four elements from the matrix registers indicated by vs are converted to single-precision floating-point numbers. The four-element floating-point result is stored in locations of the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- float16_to_float( s[0].l );
d[1] <- float16_to_float( s[0].u );
d[2] <- float16_to_float( s[1].l );
d[3] <- float16_to_float( s[1].u );
WriteMatrix( QUADWORD, vd, d );
```

## vhdp.p

### Homogeneous Dot Product Pair Word



VFPU

#### Syntax:

```
vhdp.p vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Valid	Valid

#### Description:

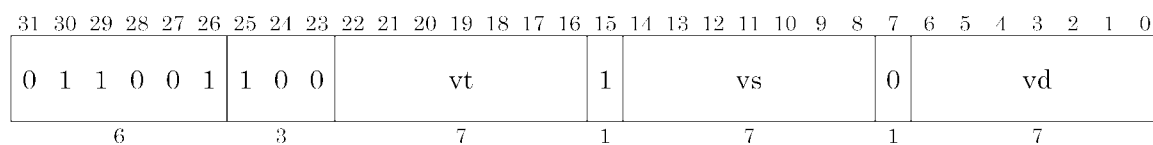
The homogeneous dot product of two elements from the matrix registers indicated by vs and two elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + t[1];
WriteMatrix( SINGLEWORD, vd, d );
```

## vhdp.t

### Homogeneous Dot Product Triple Word



VFPU

#### Syntax:

```
vhdp.t vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Valid	Valid

#### Description:

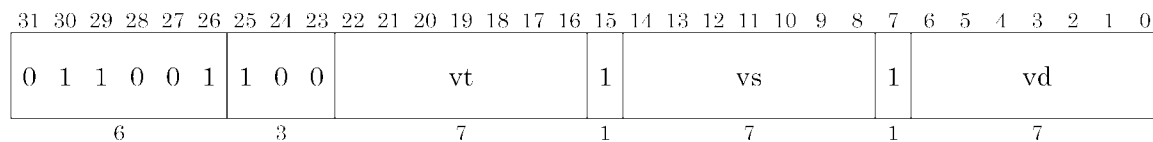
The homogeneous dot product of three elements from the matrix registers indicated by vs and three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + s[1] * t[1];
d[0] <- d[0] + t[2];
WriteMatrix( SINGLEWORD, vd, d );
```

## vhdp.q

### Homogeneous Dot Product Quad Word



VFPU

#### Syntax:

```
vhdp.q vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Valid	Valid

#### Description:

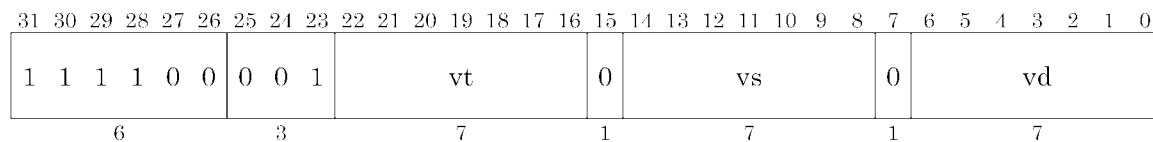
The homogeneous dot product of four elements from the matrix registers indicated by vs and four elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] * t[0];
d[0] <- d[0] + s[1] * t[1];
d[0] <- d[0] + s[2] * t[2];
d[0] <- d[0] + t[3];
WriteMatrix( SINGLEWORD, vd, d );
```

## vhtfm2.p

### Homogeneous Transform 2 Pair Word



VFPU

#### Syntax:

```
vhtfm2.p vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 8      pitch : 2

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

The transform of the elements of the 2x2 matrix from the matrix registers indicated by vs and two elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

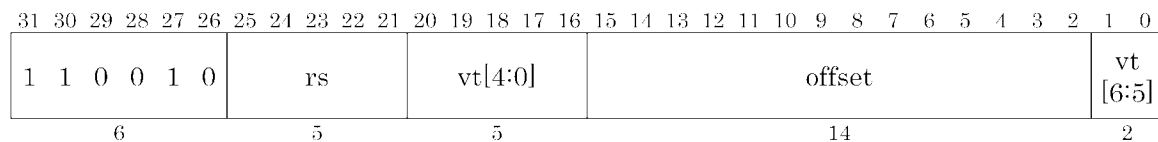
#### Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[0] + s[4];
d[1] <- s[1] * t[0] + s[5];
WriteMatrix( PAIRWORD, vd, d );
```



## lv.s

### Load Single Word to VFPU



VFPU

#### Syntax:

```
lv.s vt, offset(rs)
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. The word in memory at this effective address is stored at the location in the matrix register file indicated by vt. If the address is not word aligned, the CPU generates an address error exception.

#### Operation:

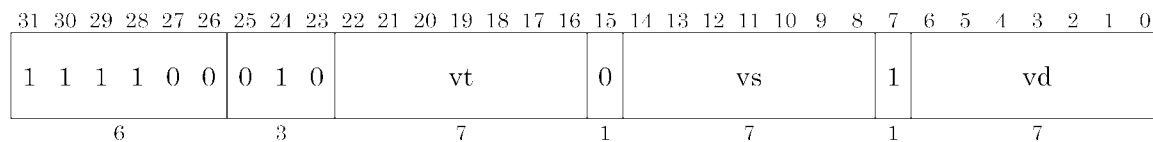
```
vAddr <- sign_extend( {offset[15:2], 2'b0} ) + GPR[rs];
pAddr <- AddressTranslation( vAddr, DATA, LOAD );
memword <- LoadMemory( SINGLEWORD, pAddr, vAddr, DATA );
WriteMatrix( SINGLEWORD, {vt[6:5], vt[4:0]}, memword );
```

#### Exceptions:

Address Error exception  
Bus Error exception

## vhtfm3.t

### Homogeneous Transform 3 Triple Word



VFPU

#### Syntax:

```
vhtfm3.t vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9      pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

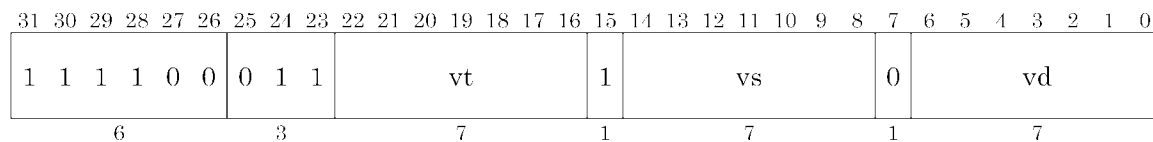
The transform of the elements of the 3x3 matrix from the matrix registers indicated by vs and three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] * t[0] + s[4] * t[1] + s[8];
d[1] <- s[1] * t[0] + s[5] * t[1] + s[9];
d[2] <- s[2] * t[0] + s[6] * t[1] + s[10];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vhtfm4.q

### Homogeneous Transform 4 Quad Word



VFPU

#### Syntax:

```
vhtfm4.q vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 10      pitch : 4

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

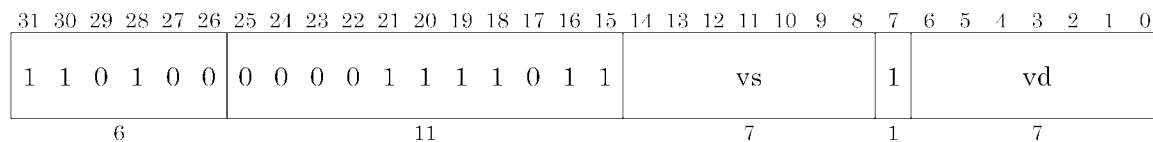
The transform of the elements of the 4x4 matrix from the matrix registers indicated by vs and four elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( QUADXQUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] * t[0] + s[4] * t[1] + s[8] * t[2] + s[12];
d[1] <- s[1] * t[0] + s[5] * t[1] + s[9] * t[2] + s[13];
d[2] <- s[2] * t[0] + s[6] * t[1] + s[10] * t[2] + s[14];
d[3] <- s[3] * t[0] + s[7] * t[1] + s[11] * t[2] + s[15];
WriteMatrix( QUADWORD, vd, d );
```

## vi2c.q

Convert integer to signed char Quad Word



VFPU

### Syntax:

```
vi2c.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

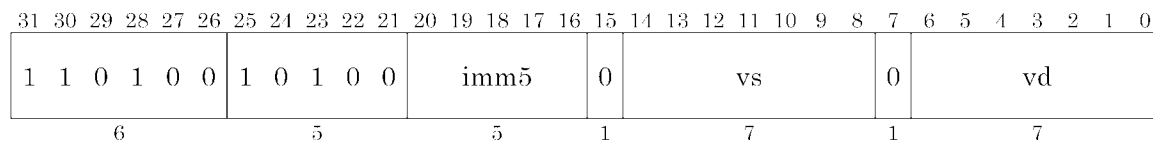
The integer values of four elements from the matrix registers indicated by vs are converted to signed 8-bit integers and packed into 32 bits. The 32-bit result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
tmp[ 7: 0] <- s[0]>>24;
tmp[15: 8] <- s[1]>>24;
tmp[23:16] <- s[2]>>24;
tmp[31:24] <- s[3]>>24;
d[0] <- tmp;
WriteMatrix( SINGLEWORD, vd, d );
```

## vi2f.s

Convert integer to float with Scaling Single Word



VFPU

### Syntax:

```
vi2f.s vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

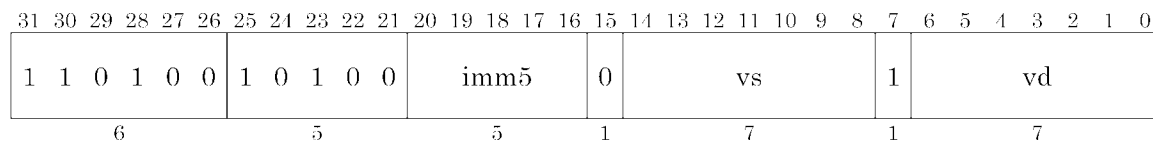
The integer value of one element from the matrix register indicated by vs is converted to a floating-point number and divided by 2 raised to the imm5 power. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- float( s[0] ) / (1<<imm5);
WriteMatrix( SINGLEWORD, vd, d );
```

## vi2f.p

Convert integer to float with Scaling Pair Word



VFPU

### Syntax:

```
vi2f.p vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

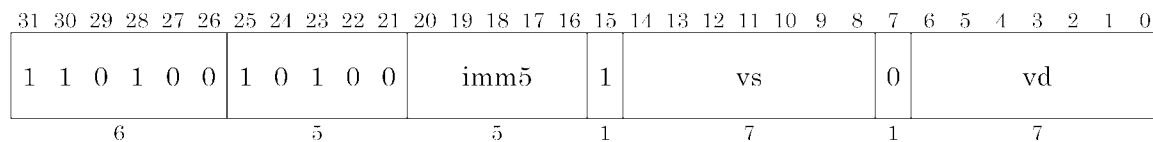
The integer values of two elements from the matrix registers indicated by vs are converted to floating-point numbers and divided by 2 raised to the imm5 power. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- float( s[0] ) / (1<<imm5);
d[1] <- float( s[1] ) / (1<<imm5);
WriteMatrix( PAIRWORD, vd, d );
```

## vi2f.t

Convert integer to float with Scaling Triple Word



VFPU

### Syntax:

```
vi2f.t vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

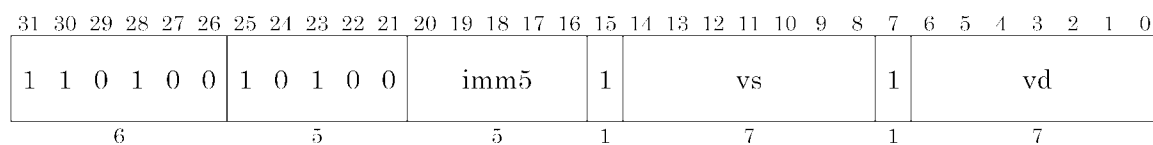
The integer values of three elements from the matrix registers indicated by vs are converted to floating-point numbers and divided by 2 raised to the imm5 power. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- float( s[0] ) / (1<<imm5);
d[1] <- float( s[1] ) / (1<<imm5);
d[2] <- float( s[2] ) / (1<<imm5);
WriteMatrix( TRIPLEWORD, vd, d );
```

## vi2f.q

Convert integer to float with Scaling Quad Word



VFPU

### Syntax:

```
vi2f.q vd, vs, imm5
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

The integer values of four elements from the matrix registers indicated by vs are converted to floating-point numbers and divided by 2 raised to the imm5 power. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

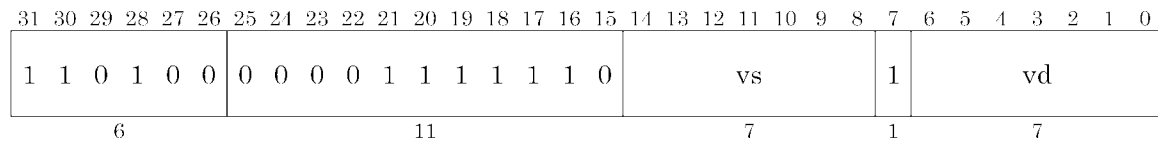
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- float( s[0] ) / (1<<imm5);
d[1] <- float( s[1] ) / (1<<imm5);
d[2] <- float( s[2] ) / (1<<imm5);
d[3] <- float( s[3] ) / (1<<imm5);
WriteMatrix( QUADWORD, vd, d );
```



## vi2s.p

Convert integer to signed short Pair Word



VFPV

### Syntax:

vi2s.p vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

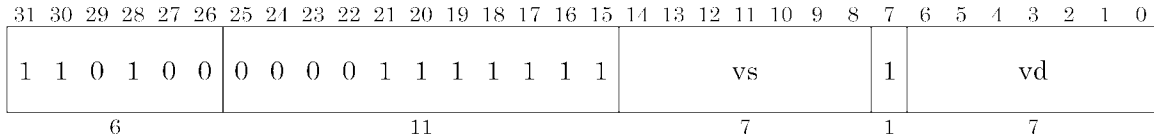
The integer values of two elements from the matrix registers indicated by vs are converted to signed 16-bit integers and packed into 32 bits. The 32-bit result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
tmp0[15: 0] <- s[0]>>16;
tmp0[31:16] <- s[1]>>16;
d[0] <- tmp0;
WriteMatrix( SINGLEWORD, vd, d );
```

## vi2s.q

Convert integer to signed short Quad Word



VFPV

### Syntax:

vi2s.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

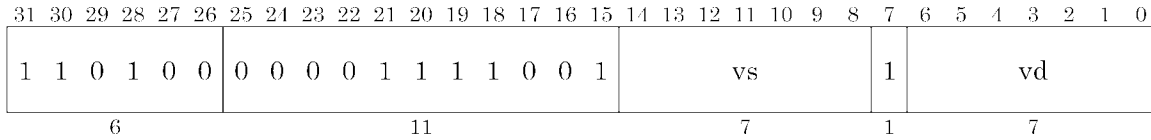
The integer values of four elements from the matrix registers indicated by vs are converted to signed 16-bit integers and packed into 64 bits. The 64-bit result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
tmp0[15: 0] <- s[0]>>16;
tmp0[31:16] <- s[1]>>16;
tmp1[15: 0] <- s[2]>>16;
tmp1[31:16] <- s[3]>>16;
d[0] <- tmp0;
d[1] <- tmp1;
WriteMatrix( PAIRWORD, vd, d );
```

## vi2uc.q

Convert integer to unsigned char Quad Word



VFPU

### Syntax:

```
vi2uc.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

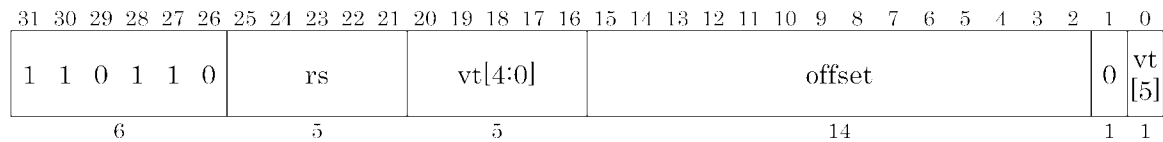
The integer values of four elements from the matrix registers indicated by vs are converted to unsigned 8-bit integers and packed into 32 bits. The 32-bit result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
if( s[0] < 0 )
    tmp[ 7: 0] <- 0;
else
    tmp[ 7: 0] <- s[0]>>23;
if( s[1] < 0 )
    tmp[15: 8] <- 0;
else
    tmp[15: 8] <- s[1]>>23;
if( s[2] < 0 )
    tmp[23:16] <- 0;
else
    tmp[23:16] <- s[2]>>23;
if( s[3] < 0 )
```

## lv.q

### Load Quad Word to VFPU



VFPU

#### Syntax:

```
lv.q vt, offset(rs)
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. The quadword in memory at this effective address is stored at locations in the matrix register file indicated by vt. If the address is not quadword aligned, the CPU generates an address error exception.

#### Operation:

```
vAddr <- sign_extend( {offset[15:2], 2'b0} ) + GPR[rs];
pAddr <- AddressTranslation( vAddr, DATA, LOAD );
memword <- LoadMemory( QUADWORD, pAddr, vAddr, DATA );
WriteMatrix( QUADWORD, {vt[5], vt[4:0]}, memword );
```

#### Exceptions:

Address Error exception  
Bus Error exception

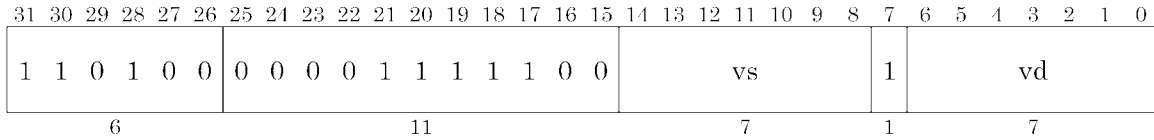
```

        tmp[31:24] <- 0;
    else
        tmp[31:24] <- s[3]>>23;
    d[0] <- tmp;
    WriteMatrix( SINGLEWORD, vd, d );

```

## vi2us.p

Convert integer to unsigned short Pair Word



VFPU

### Syntax:

```
vi2us.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3          pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

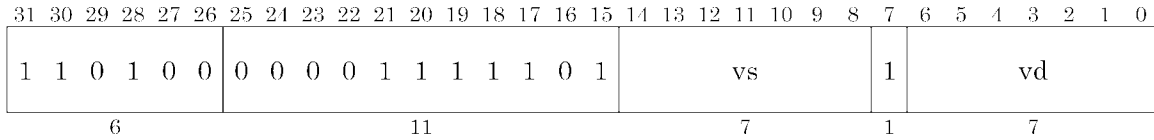
The integer values of two elements from the matrix registers indicated by `vs` are converted to unsigned 16-bit integers and packed into 32 bits. The 32-bit result is stored at the location in the matrix register file indicated by `vd`.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
if( s[0] < 0 )
    tmp0[15: 0] <- 0;
else
    tmp0[15: 0] <- s[0]>>15;
if( s[1] < 0 )
    tmp0[31:16] <- 0;
else
    tmp0[31:16] <- s[1]>>15;
d[0] <- tmp0;
WriteMatrix( SINGLEWORD, vd, d );
```

## vi2us.q

Convert integer to unsigned short Quad Word



VFPU

### Syntax:

```
vi2us.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Only write mask is valid

### Description:

The integer values of four elements from the matrix registers indicated by vs are converted to unsigned 16-bit integers and packed into 64 bits. The 64-bit result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
if( s[0] < 0 )
    tmp0[15: 0] <- 0;
else
    tmp0[15: 0] <- s[0]>>15;
if( s[1] < 0 )
    tmp0[31:16] <- 0;
else
    tmp0[31:16] <- s[1]>>15;
if( s[2] < 0 )
    tmp1[15: 0] <- 0;
else
    tmp1[15: 0] <- s[2]>>15;
if( s[3] < 0 )
```

```

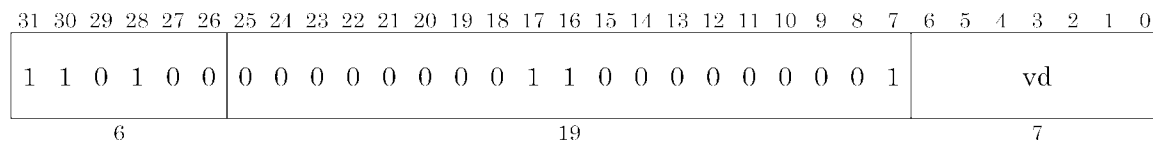
        tmp1[31:16] <- 0;
    else
        tmp1[31:16] <- s[3]>>15;
    d[0] <- tmp0;
    d[1] <- tmp1;
    WriteMatrix( PAIRWORD, vd, d );

```



## vidt.p

Identity Pair Word



VFPU

### Syntax:

```
vidt.p vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

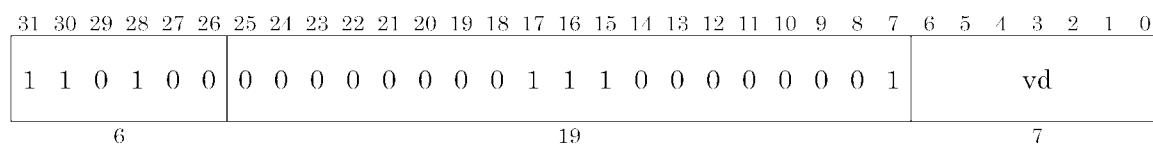
One vector from the identity matrix is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- (vd[0]==0) ? 1.0 : 0.0;
d[1] <- (vd[0]==1) ? 1.0 : 0.0;
WriteMatrix( PAIRWORD, vd, d );
```

## vidt.q

Identity Quad Word



VFPU

### Syntax:

```
vidt.q vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

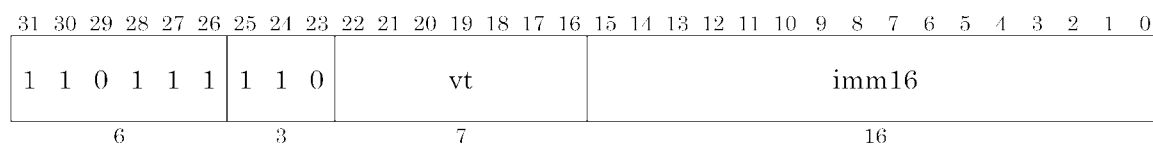
One vector from the identity matrix is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- (vd[1:0]==0) ? 1.0 : 0.0;
d[1] <- (vd[1:0]==1) ? 1.0 : 0.0;
d[2] <- (vd[1:0]==2) ? 1.0 : 0.0;
d[3] <- (vd[1:0]==3) ? 1.0 : 0.0;
WriteMatrix( QUADWORD, vd, d );
```

## viim.s

Convert integer immediate to float Single Word



VFPU

### Syntax:

```
viim.s vt, imm16
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

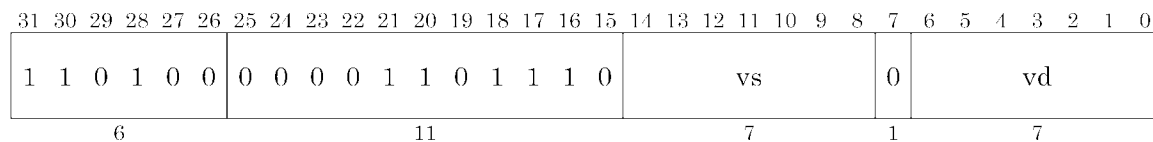
The integer indicated by imm16 is converted to a floating-point number and stored at the location in the matrix register file indicated by vt.

### Operation:

```
f <- float( imm16 );
WriteMatrix( SINGLEWORD, vt, f );
```

## vlgb.s

LogB Single Word



VFPU

### Syntax:

vlgb.s vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The logB of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

logB is defined by the following expression.

$$x = \text{scaleBZ}(x) * 2^{\log B(x)} ; 1 \leq \text{scaleBZ}(x) < 2.$$

Special solutions are as follows.

$$\log B(\text{nan}) = \text{nan}$$

$$\log B(+\text{inf}) = +\text{inf}$$

$$\log B(-\text{inf}) = -\text{inf}$$

$$\log B(+0.0) = -\text{inf}$$

$$\log B(-0.0) = -\text{inf}$$

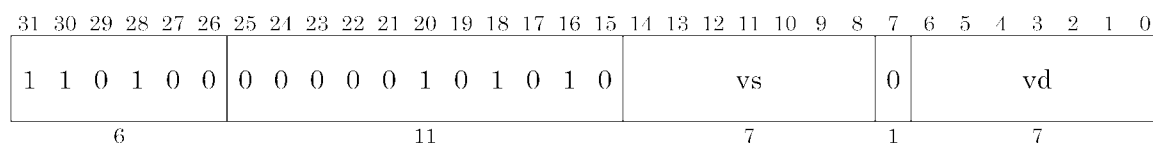
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
```

```
d[0] <- logB( |s[0]| );
WriteMatrix( SINGLEWORD, vd, d );
```

# vlog2.s

Logarithm base 2 Single Word



VFPU

## Syntax:

vlog2.s vd, vs

## Instruction Type

Pipeline instruction

## Processing Time:

latency : 7      pitch : 1

## Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

## Description:

The base 2 logarithm of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_log2}(x) - \log_2(x) | < 2^{-20}$$

Special solutions are as follows.

$$\text{approx\_log2}(\text{nan}) = \text{nan}$$

$$\text{approx\_log2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_log2}(-\text{inf}) = \text{nan}$$

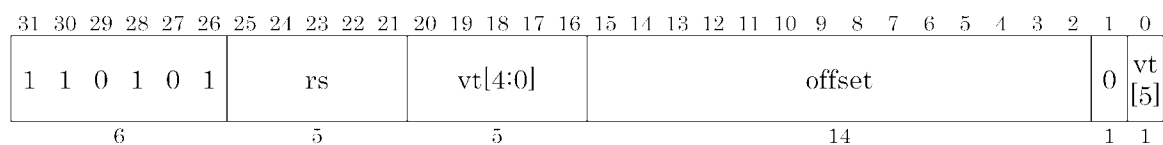
$$\text{approx\_log2}(+0.0) = -\text{inf}$$

$$\text{approx\_log2}(-0.0) = -\text{inf}$$

$$\text{approx\_log2}(x) = \text{nan} ; -\text{inf} < x < -0.0$$

## lvl.q

Load Quad Word Left to VFPU



VFPU

### Syntax:

```
lvl.q vt, offset(rs)
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. From one to four words are read from memory beginning at this effective address such that the high-order word is located at this address, and the low-order word ends on a quadword boundary. The resultant words are stored left-justified within the quadword at locations in the matrix register file indicated by vt. Any remaining words to the right in the quadword are unaffected by the instruction and will not be changed. If the effective address is not word aligned, the CPU generates an address error exception.

### Operation:

```
vAddr  <- sign_extend( {offset[15:2],2'b0} ) + GPR[rs];
pAddr  <- AddressTranslation( vAddr, DATA, LOAD );
offset  <- pAddr[3:2];
dataword <- LoadMemory( QUADWORD, pAddr, vAddr, DATA );
```

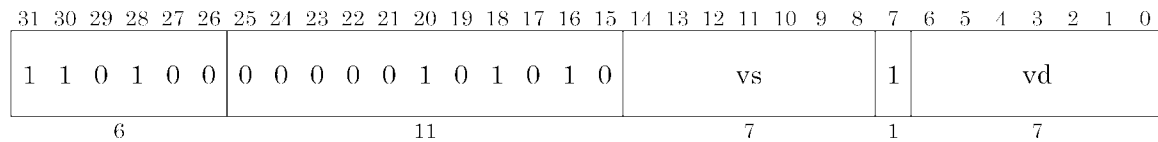
**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_log2( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vlog2.p

Logarithm base 2 Pair Word



VFPU

### Syntax:

vlog2.p vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 logarithms of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_log2}(x) - \log_2(x) | < 2^{-20}$$

Special solutions are as follows.

$$\text{approx\_log2}(\text{nan}) = \text{nan}$$

$$\text{approx\_log2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_log2}(-\text{inf}) = \text{nan}$$

$$\text{approx\_log2}(+0.0) = -\text{inf}$$

$$\text{approx\_log2}(-0.0) = -\text{inf}$$

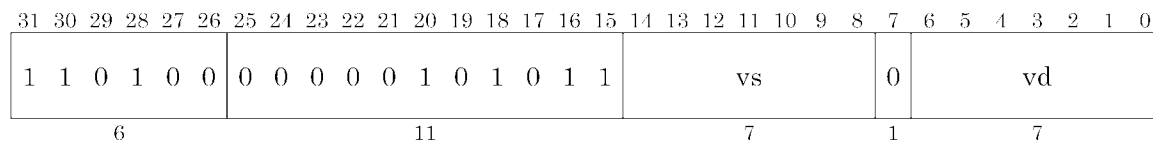
$$\text{approx\_log2}(x) = \text{nan} ; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_log2( s[0] );
d[1] <- approx_log2( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vlog2.t

Logarithm base 2 Triple Word



VFPU

### Syntax:

vlog2.t vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 logarithms of the floating-point values of three elements from the matrix registers indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_log2}(x) - \log_2(x) | < 2^{-20}$$

Special solutions are as follows.

$$\text{approx\_log2}(\text{nan}) = \text{nan}$$

$$\text{approx\_log2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_log2}(-\text{inf}) = \text{nan}$$

$$\text{approx\_log2}(+0.0) = -\text{inf}$$

$$\text{approx\_log2}(-0.0) = -\text{inf}$$

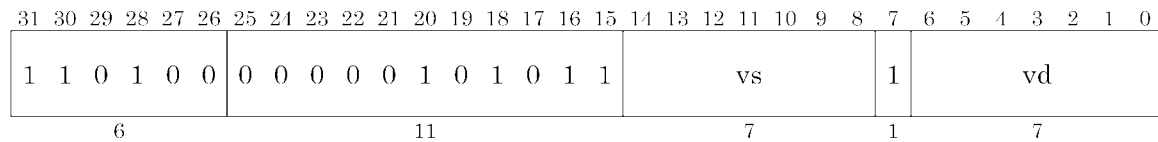
$$\text{approx\_log2}(x) = \text{nan} ; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_log2( s[0] );
d[1] <- approx_log2( s[1] );
d[2] <- approx_log2( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vlog2.q

Logarithm base 2 Quad Word



VFPU

### Syntax:

vlog2.q vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The base 2 logarithms of the floating-point values of four elements from the matrix registers indicated by vs are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_log2}(x) - \log_2(x) | < 2^{-20}$$

Special solutions are as follows.

$$\text{approx\_log2}(\text{nan}) = \text{nan}$$

$$\text{approx\_log2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_log2}(-\text{inf}) = \text{nan}$$

$$\text{approx\_log2}(+0.0) = -\text{inf}$$

$$\text{approx\_log2}(-0.0) = -\text{inf}$$

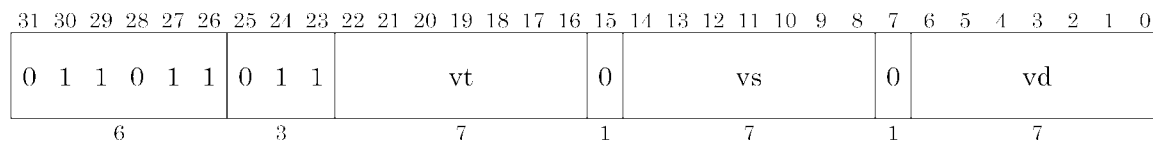
$$\text{approx\_log2}(x) = \text{nan} ; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_log2( s[0] );
d[1] <- approx_log2( s[1] );
d[2] <- approx_log2( s[2] );
d[3] <- approx_log2( s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vmax.s

Maximum Single Word



VFPU

### Syntax:

```
vmax.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

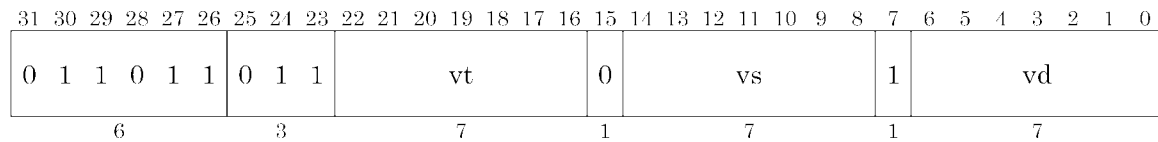
One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt and the largest element is selected. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- max( s[0] , t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vmax.p

Maximum Pair Word



VFPU

### Syntax:

```
vmax.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Two elements from the matrix registers indicated by vs are compared with the corresponding two elements from the matrix registers indicated by vt and the largest two elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

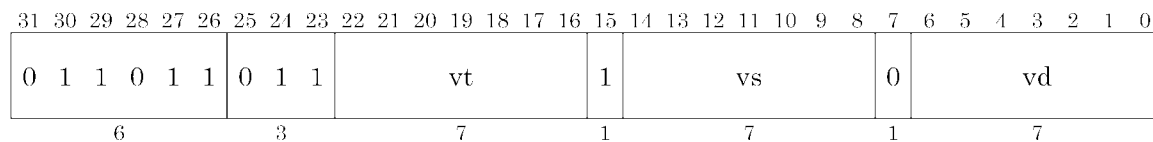
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- max( s[0] , t[0] );
d[1] <- max( s[1] , t[1] );
WriteMatrix( PAIRWORD, vd, d );
```



## vmax.t

Maximum Triple Word



VFPU

### Syntax:

```
vmax.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Three elements from the matrix registers indicated by vs are compared with the corresponding three elements from the matrix registers indicated by vt and the largest three elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- max( s[0] , t[0] );
d[1] <- max( s[1] , t[1] );
d[2] <- max( s[2] , t[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

```

d      <- ReadMatrix( QUADWORD, vt );
switch( offset )
{
  case 0 : d[3] <- dataword[0]; break;
  case 1 : d[3] <- dataword[1];
           d[2] <- dataword[0]; break;
  case 2 : d[3] <- dataword[2];
           d[2] <- dataword[1];
           d[1] <- dataword[0]; break;
  case 3 : d[3] <- dataword[3];
           d[2] <- dataword[2];
           d[1] <- dataword[1];
           d[0] <- dataword[0]; break;
}
WriteMatrix( QUADWORD, vt, d)

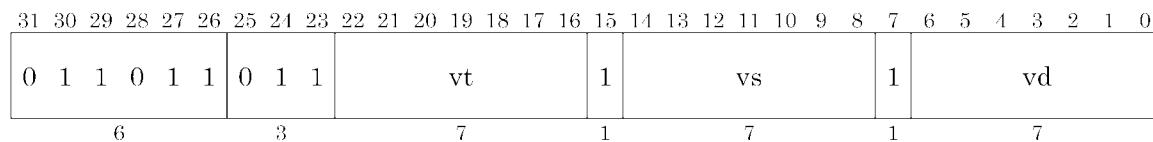
```

### Exceptions:

Address Error exception  
 Bus Error exception

## vmax.q

Maximum Quad Word



VFPU

### Syntax:

```
vmax.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

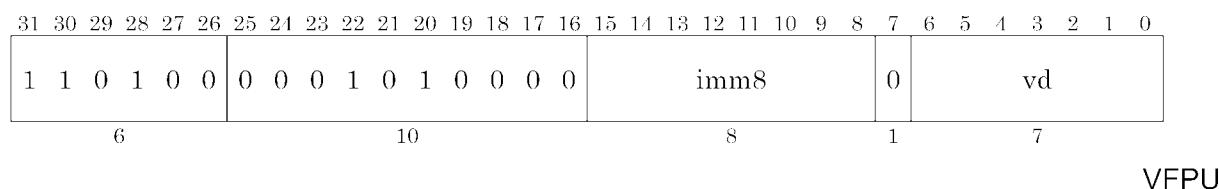
Four elements from the matrix registers indicated by vs are compared with the corresponding four elements from the matrix registers indicated by vt and the largest four elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- max( s[0] , t[0] );
d[1] <- max( s[1] , t[1] );
d[2] <- max( s[2] , t[2] );
d[3] <- max( s[3] , t[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vmfvc

### VFPU Move Word from VFPU Control



#### Syntax:

```
vmfvc vd, imm8
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

The contents of the VFPU control register indicated by the imm8 field are stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

#### Operation:

```
dataword <- ReadControl( imm8 );
WriteMatrix( SINGLEWORD, vd, dataword );
```

## vmidt.p

### Identity Pair x Pair Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	vd						
6						19																	7								

VFPU

### Syntax:

```
vmidt.p vd
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 4      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The 2x2 identity matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

### Operation:

```
d[0] <- 1.0;
d[1] <- 0.0;
d[4] <- 0.0;
d[5] <- 1.0;
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

## vmidt.t

Identity Triple x Triple Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	vd						
6						19																			7						

VFPU

### Syntax:

```
vmidt.t vd
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

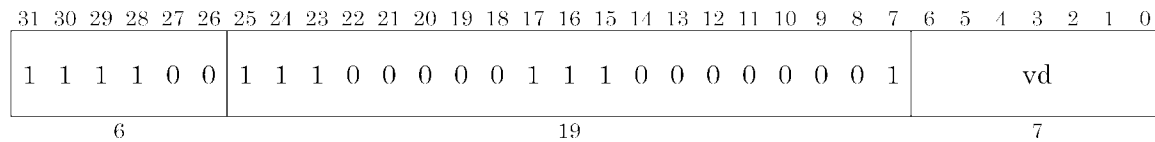
The 3x3 identity matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

### Operation:

```
d[0]  <- 1.0;
d[1]  <- 0.0;
d[2]  <- 0.0;
d[4]  <- 0.0;
d[5]  <- 1.0;
d[6]  <- 0.0;
d[8]  <- 0.0;
d[9]  <- 0.0;
d[10] <- 1.0;
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

## vmidt.q

Identity Quad x Quad Matrix



VFPU

### Syntax:

vmidt.q vd

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 6      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The 4x4 identity matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

### Operation:

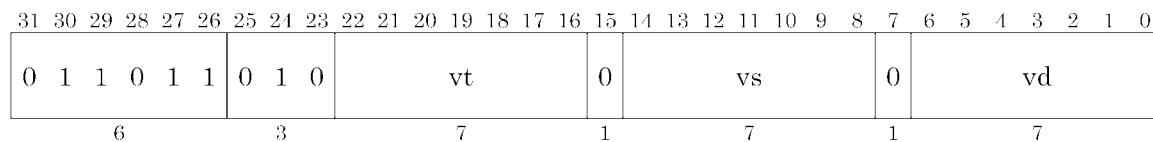
```
d[0]  <- 1.0;
d[1]  <- 0.0;
d[2]  <- 0.0;
d[3]  <- 0.0;
d[4]  <- 0.0;
d[5]  <- 1.0;
d[6]  <- 0.0;
d[7]  <- 0.0;
d[8]  <- 0.0;
d[9]  <- 0.0;
d[10] <- 1.0;
d[11] <- 0.0;
d[12] <- 0.0;
d[13] <- 0.0;
d[14] <- 0.0;
d[15] <- 1.0;
```

```
WriteMatrix( QUADXQUADWORD, vd, d );
```



## vmin.s

Minimum Single Word



VFPU

### Syntax:

```
vmin.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

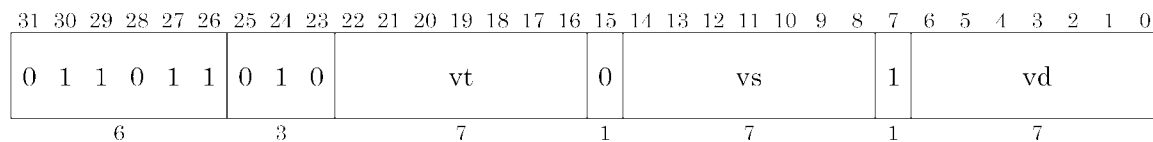
One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt and the smallest element is selected. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- min( s[0] , t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vmin.p

Minimum Pair Word



VFPU

### Syntax:

```
vmin.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

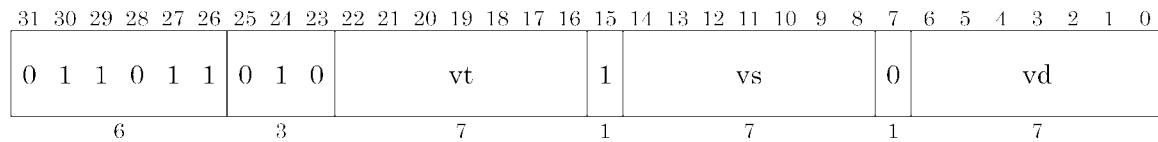
Two elements from the matrix registers indicated by vs are compared with the corresponding two elements from the matrix registers indicated by vt and the smallest elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- min( s[0] , t[0] );
d[1] <- min( s[1] , t[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vmin.t

Minimum Triple Word



VFPU

### Syntax:

```
vmin.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

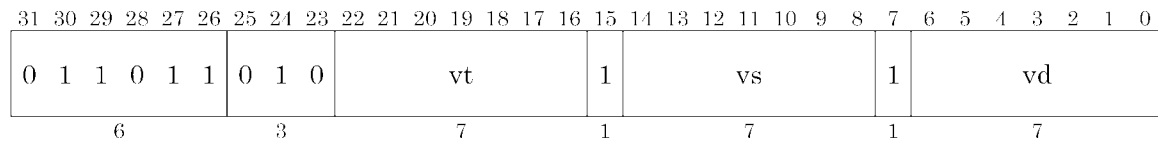
Three elements from the matrix registers indicated by vs are compared with the corresponding three elements from the matrix registers indicated by vt and the smallest elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- min( s[0] , t[0] );
d[1] <- min( s[1] , t[1] );
d[2] <- min( s[2] , t[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vmin.q

Minimum Quad Word



VFPU

### Syntax:

```
vmin.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

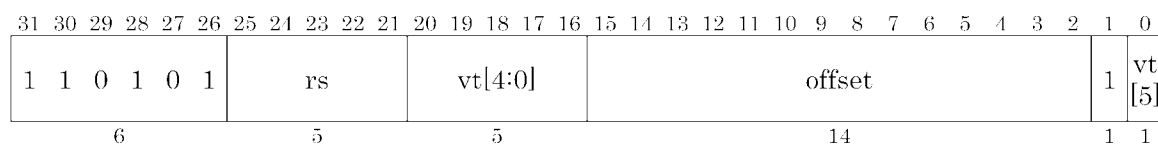
Four elements from the matrix registers indicated by vs are compared with the corresponding four elements from the matrix registers indicated by vt and the smallest elements from the individual comparisons are selected. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- min( s[0] , t[0] );
d[1] <- min( s[1] , t[1] );
d[2] <- min( s[2] , t[2] );
d[3] <- min( s[3] , t[3] );
WriteMatrix( QUADWORD, vd, d );
```

## lvr.q

Load Quad Word Right to VFPU



VFPU

### Syntax:

```
lvr.q vt, offset(rs)
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. From one to four words are read from memory ending at this effective address such that the high-order word starts on a quadword boundary and the low-order word is located at this address. The resultant words are stored right-justified within the quadword at locations in the matrix register file indicated by vt. Any remaining words to the left in the quadword are unaffected by the instruction and will not be changed. If the effective address is not word aligned, the CPU generates an address error exception.

### Operation:

```
vAddr  <- sign_extend( {offset[15:2],2'b0} ) + GPR[rs];
pAddr  <- AddressTranslation( vAddr, DATA, LOAD );
offset <- pAddr[3:2];
dataword <- LoadMemory( QUADWORD, pAddr, vAddr, DATA );
```

## vmmov.p

### Move Pair x Pair Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0	0	vs							1	vd						
6						11											7							1	7						

VFPU

### Syntax:

```
vmmov.p vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 4      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

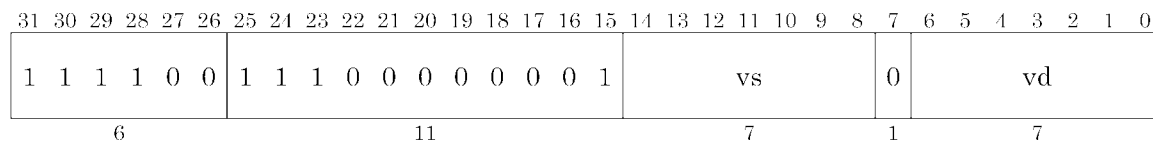
The floating-point values of the elements of the 2x2 matrix from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vs );
d[0] <- s[0];
d[1] <- s[1];
d[4] <- s[4];
d[5] <- s[5];
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

## vmmov.t

Move Triple x Triple Matrix



VFPU

### Syntax:

```
vmmov.t vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

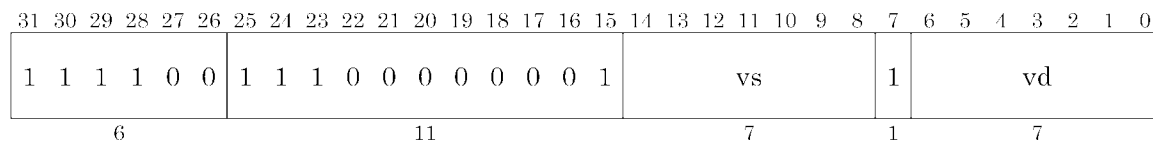
The floating-point values of the elements of the 3x3 matrix from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vs );
d[0] <- s[0];
d[1] <- s[1];
d[2] <- s[2];
d[4] <- s[4];
d[5] <- s[5];
d[6] <- s[6];
d[8] <- s[8];
d[9] <- s[9];
d[10] <- s[10];
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

## vmmov.q

Move Quad x Quad Matrix



VFPU

### Syntax:

```
vmmov.q vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 6      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The floating-point values of the elements of the 4x4 matrix from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

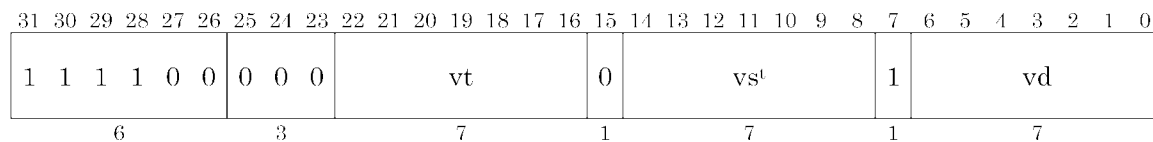
```
s <- ReadMatrix( QUADXQUADWORD, vs );
d[0] <- s[0];
d[1] <- s[1];
d[2] <- s[2];
d[3] <- s[3];
d[4] <- s[4];
d[5] <- s[5];
d[6] <- s[6];
d[7] <- s[7];
d[8] <- s[8];
d[9] <- s[9];
d[10] <- s[10];
d[11] <- s[11];
d[12] <- s[12];
d[13] <- s[13];
d[14] <- s[14];
```



```
d[15] <- s[15];
WriteMatrix( QUADXQUADWORD, vd, d );
```

## vmmul.p

Multiply Pair x Pair Matrix



VFPU

### Syntax:

```
vmmul.p vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

The 2x2 matrix from the inverted matrix registers indicated by vs<sup>t</sup> is multiplied by the 2x2 matrix from the matrix registers indicated by vt. The elements of the matrices are treated as floating-point values. The 2x2 matrix floating-point result is stored in locations of the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vst );
t <- ReadMatrix( PAIRXPAIRWORD, vt );
d[ 0] <- s[ 0]*t[ 0] + s[ 1]*t[ 1];
d[ 4] <- s[ 0]*t[ 4] + s[ 1]*t[ 5];
d[ 1] <- s[ 4]*t[ 0] + s[ 5]*t[ 1];
d[ 5] <- s[ 4]*t[ 4] + s[ 5]*t[ 5];
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

### Equivalence Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vs );
t <- ReadMatrix( PAIRXPAIRWORD, vt );
```

```
d[ 0] <- s[ 0]*t[ 0] + s[ 4]*t[ 1];
d[ 4] <- s[ 0]*t[ 4] + s[ 4]*t[ 5];
d[ 1] <- s[ 1]*t[ 0] + s[ 5]*t[ 1];
d[ 5] <- s[ 1]*t[ 4] + s[ 5]*t[ 5];
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

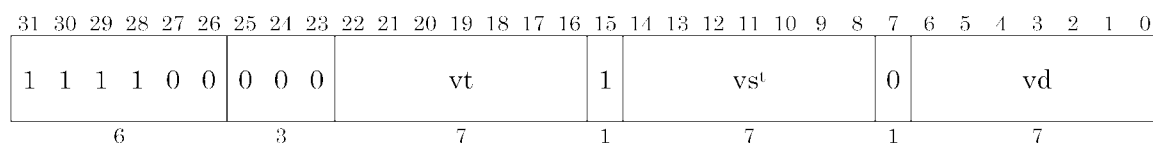
**Note:**

In the opcode of the instruction vmmul.p, please note that the field which corresponds to the operand vs is written as vs<sup>t</sup> which is an inverted expression of the matrix registers. The assembler exceptionally inverts the RXC bit (bit 13 of the opcode) of the operand vs of vmmul instructions. Accordingly, the operation of multiplying the 2x2 matrix m100 by m200 from the right can be written as follows:

```
vmmul.p      m000, m100, m200
```

## vmmul.t

### Multiply Triple x Triple Matrix



VFPU

#### Syntax:

```
vmmul.t vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 15      pitch : 9

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

The 3x3 matrix from the inverted matrix registers indicated by vs<sup>t</sup> is multiplied by the 3x3 matrix from the matrix registers indicated by vt. The elements of the matrices are treated as floating-point values. The 3x3 matrix floating-point result is stored in locations of the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vst );
t <- ReadMatrix( TRIPLEXTRIPLEWORD, vt );
d[ 0] <- s[ 0]*t[ 0] + s[ 1]*t[ 1] + s[ 2]*t[ 2];
d[ 4] <- s[ 0]*t[ 4] + s[ 1]*t[ 5] + s[ 2]*t[ 6];
d[ 8] <- s[ 0]*t[ 8] + s[ 1]*t[ 9] + s[ 2]*t[10];
d[ 1] <- s[ 4]*t[ 0] + s[ 5]*t[ 1] + s[ 6]*t[ 2];
d[ 5] <- s[ 4]*t[ 4] + s[ 5]*t[ 5] + s[ 6]*t[ 6];
d[ 9] <- s[ 4]*t[ 8] + s[ 5]*t[ 9] + s[ 6]*t[10];
d[ 2] <- s[ 8]*t[ 0] + s[ 9]*t[ 1] + s[10]*t[ 2];
d[ 6] <- s[ 8]*t[ 4] + s[ 9]*t[ 5] + s[10]*t[ 6];
d[10] <- s[ 8]*t[ 8] + s[ 9]*t[ 9] + s[10]*t[10];
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

### Equivalence Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEXTRIPLEWORD, vt );
d[ 0] <- s[ 0]*t[ 0] + s[ 4]*t[ 1] + s[ 8]*t[ 2];
d[ 4] <- s[ 0]*t[ 4] + s[ 4]*t[ 5] + s[ 8]*t[ 6];
d[ 8] <- s[ 0]*t[ 8] + s[ 4]*t[ 9] + s[ 8]*t[10];
d[ 1] <- s[ 1]*t[ 0] + s[ 5]*t[ 1] + s[ 9]*t[ 2];
d[ 5] <- s[ 1]*t[ 4] + s[ 5]*t[ 5] + s[ 9]*t[ 6];
d[ 9] <- s[ 1]*t[ 8] + s[ 5]*t[ 9] + s[ 9]*t[10];
d[ 2] <- s[ 2]*t[ 0] + s[ 6]*t[ 1] + s[10]*t[ 2];
d[ 6] <- s[ 2]*t[ 4] + s[ 6]*t[ 5] + s[10]*t[ 6];
d[10] <- s[ 2]*t[ 8] + s[ 6]*t[ 9] + s[10]*t[10];
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

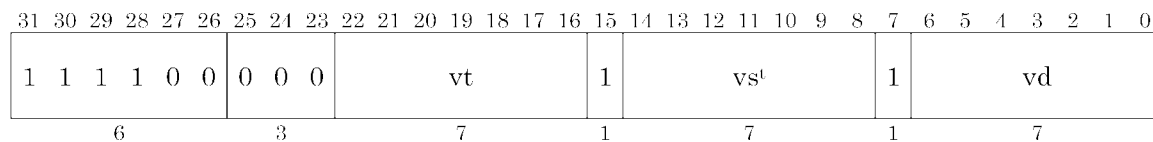
### Note:

In the opcode of the instruction vmmul.t, please note that the field which corresponds to the operand vs is written as vs<sup>t</sup> which is an inverted expression of the matrix registers. The assembler exceptionally inverts the RXC bit (bit 13 of the opcode) of the operand vs of vmmul instructions. Accordingly, the operation of multiplying the 3x3 matrix m100 by m200 from the right can be written as follows:

vmmul.t                      m000, m100, m200

## vmmul.q

Multiply Quad x Quad Matrix



VFPU

### Syntax:

```
vmmul.q vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 22      pitch : 16

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

The 4x4 matrix from the inverted matrix registers indicated by vs<sup>t</sup> is multiplied by the 4x4 matrix from the matrix registers indicated by vt. The elements of the matrices are treated as floating-point values. The 4x4 matrix floating-point result is stored in locations of the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADXQUADWORD, vst );
t <- ReadMatrix( QUADXQUADWORD, vt );
d[ 0] <- s[ 0]*t[ 0] + s[ 1]*t[ 1] + s[ 2]*t[ 2] + s[ 3]*t[ 3];
d[ 4] <- s[ 0]*t[ 4] + s[ 1]*t[ 5] + s[ 2]*t[ 6] + s[ 3]*t[ 7];
d[ 8] <- s[ 0]*t[ 8] + s[ 1]*t[ 9] + s[ 2]*t[10] + s[ 3]*t[11];
d[12] <- s[ 0]*t[12] + s[ 1]*t[13] + s[ 2]*t[14] + s[ 3]*t[15];
d[ 1] <- s[ 4]*t[ 0] + s[ 5]*t[ 1] + s[ 6]*t[ 2] + s[ 7]*t[ 3];
d[ 5] <- s[ 4]*t[ 4] + s[ 5]*t[ 5] + s[ 6]*t[ 6] + s[ 7]*t[ 7];
d[ 9] <- s[ 4]*t[ 8] + s[ 5]*t[ 9] + s[ 6]*t[10] + s[ 7]*t[11];
d[13] <- s[ 4]*t[12] + s[ 5]*t[13] + s[ 6]*t[14] + s[ 7]*t[15];
d[ 2] <- s[ 8]*t[ 0] + s[ 9]*t[ 1] + s[10]*t[ 2] + s[11]*t[ 3];
d[ 6] <- s[ 8]*t[ 4] + s[ 9]*t[ 5] + s[10]*t[ 6] + s[11]*t[ 7];
```

```
d[10] <- s[ 8]*t[ 8] + s[ 9]*t[ 9] + s[10]*t[10] + s[11]*t[11];
d[14] <- s[ 8]*t[12] + s[ 9]*t[13] + s[10]*t[14] + s[11]*t[15];
d[ 3] <- s[12]*t[ 0] + s[13]*t[ 1] + s[14]*t[ 2] + s[15]*t[ 3];
d[ 7] <- s[12]*t[ 4] + s[13]*t[ 5] + s[14]*t[ 6] + s[15]*t[ 7];
d[11] <- s[12]*t[ 8] + s[13]*t[ 9] + s[14]*t[10] + s[15]*t[11];
d[15] <- s[12]*t[12] + s[13]*t[13] + s[14]*t[14] + s[15]*t[15];
WriteMatrix( QUADXQUADWORD, vd, d );
```

### Equivalence Operation:

```
s <- ReadMatrix( QUADXQUADWORD, vs );
t <- ReadMatrix( QUADXQUADWORD, vt );
d[ 0] <- s[ 0]*t[ 0] + s[ 4]*t[ 1] + s[ 8]*t[ 2] + s[12]*t[ 3];
d[ 4] <- s[ 0]*t[ 4] + s[ 4]*t[ 5] + s[ 8]*t[ 6] + s[12]*t[ 7];
d[ 8] <- s[ 0]*t[ 8] + s[ 4]*t[ 9] + s[ 8]*t[10] + s[12]*t[11];
d[12] <- s[ 0]*t[12] + s[ 4]*t[13] + s[ 8]*t[14] + s[12]*t[15];
d[ 1] <- s[ 1]*t[ 0] + s[ 5]*t[ 1] + s[ 9]*t[ 2] + s[13]*t[ 3];
d[ 5] <- s[ 1]*t[ 4] + s[ 5]*t[ 5] + s[ 9]*t[ 6] + s[13]*t[ 7];
d[ 9] <- s[ 1]*t[ 8] + s[ 5]*t[ 9] + s[ 9]*t[10] + s[13]*t[11];
d[13] <- s[ 1]*t[12] + s[ 5]*t[13] + s[ 9]*t[14] + s[13]*t[15];
d[ 2] <- s[ 2]*t[ 0] + s[ 6]*t[ 1] + s[10]*t[ 2] + s[14]*t[ 3];
d[ 6] <- s[ 2]*t[ 4] + s[ 6]*t[ 5] + s[10]*t[ 6] + s[14]*t[ 7];
d[10] <- s[ 2]*t[ 8] + s[ 6]*t[ 9] + s[10]*t[10] + s[14]*t[11];
d[14] <- s[ 2]*t[12] + s[ 6]*t[13] + s[10]*t[14] + s[14]*t[15];
d[ 3] <- s[ 3]*t[ 0] + s[ 7]*t[ 1] + s[11]*t[ 2] + s[15]*t[ 3];
d[ 7] <- s[ 3]*t[ 4] + s[ 7]*t[ 5] + s[11]*t[ 6] + s[15]*t[ 7];
d[11] <- s[ 3]*t[ 8] + s[ 7]*t[ 9] + s[11]*t[10] + s[15]*t[11];
d[15] <- s[ 3]*t[12] + s[ 7]*t[13] + s[11]*t[14] + s[15]*t[15];
WriteMatrix( QUADXQUADWORD, vd, d );
```

### Note:

In the opcode of the instruction `vmmul.q`, please note that the field which corresponds to the operand `vs` is written as `vst` which is an inverted expression of the matrix registers. The assembler exceptionally inverts the `RXC` bit (bit 13 of the opcode) of the operand `vs` of `vmmul` instructions. Accordingly, the operation of multiplying the 4x4 matrix `m100` by `m200` from the right can be written as follows:

`vmmul.q`                      `m000, m100, m200`

```

d      <- ReadMatrix( QUADWORD, vt );
switch( offset )
{
  case 0 : d[3] <- dataword[3];
           d[2] <- dataword[2];
           d[1] <- dataword[1];
           d[0] <- dataword[0]; break;
  case 1 : d[2] <- dataword[3];
           d[1] <- dataword[2];
           d[0] <- dataword[1]; break;
  case 2 : d[1] <- dataword[3];
           d[0] <- dataword[2]; break;
  case 3 : d[0] <- dataword[3]; break;
}
WriteMatrix( QUADWORD, vt, d );

```

### Exceptions:

Address Error exception  
 Bus Error exception



## vmone.p

One Pair x Pair Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	vd						
6						19																			7						

VFPU

### Syntax:

```
vmone.p vd
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 4      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

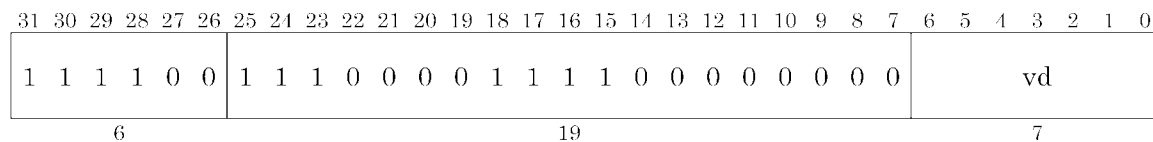
A 2x2 matrix is generated in which all elements have a floating-point value of 1.0. The generated 2x2 matrix is stored at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 1.0;
d[1] <- 1.0;
d[4] <- 1.0;
d[5] <- 1.0;
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

## vmone.t

One Triple x Triple Matrix



VFPU

### Syntax:

`vmone.t vd`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

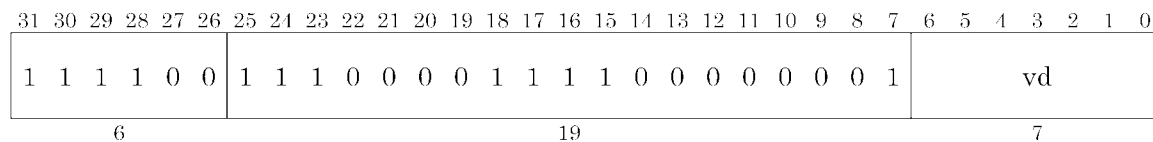
A 3x3 matrix is generated in which all elements have a floating-point value of 1.0. The generated 3x3 matrix is stored at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 1.0;
d[1] <- 1.0;
d[2] <- 1.0;
d[4] <- 1.0;
d[5] <- 1.0;
d[6] <- 1.0;
d[8] <- 1.0;
d[9] <- 1.0;
d[10] <- 1.0;
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

## vmone.q

One Quad x Quad Matrix



VFPU

### Syntax:

`vmone.q vd`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 6      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

A 4x4 matrix is generated in which all elements have a floating-point value of 1.0. The generated 4x4 matrix is stored at locations in the matrix register file indicated by vd.

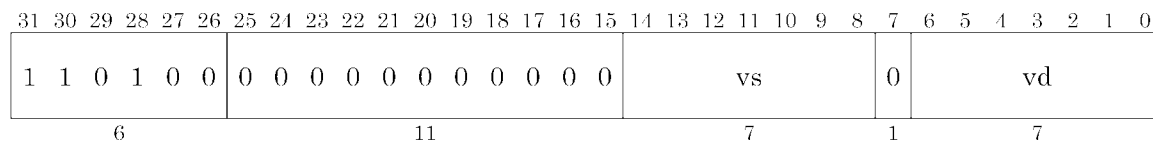
### Operation:

```
d[0]  <- 1.0;
d[1]  <- 1.0;
d[2]  <- 1.0;
d[3]  <- 1.0;
d[4]  <- 1.0;
d[5]  <- 1.0;
d[6]  <- 1.0;
d[7]  <- 1.0;
d[8]  <- 1.0;
d[9]  <- 1.0;
d[10] <- 1.0;
d[11] <- 1.0;
d[12] <- 1.0;
d[13] <- 1.0;
d[14] <- 1.0;
d[15] <- 1.0;
```

```
WriteMatrix( QUADXQUADWORD, vd, d );
```

## vmov.s

Move Single Word



VFPU

### Syntax:

```
vmov.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

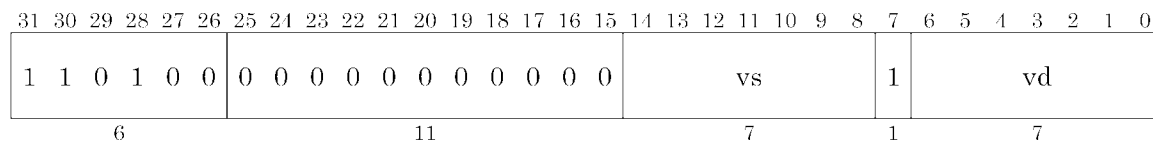
The floating-point value of one element from the matrix register indicated by vs is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d <- s;
WriteMatrix( SINGLEWORD, vd, d );
```

## vmov.p

Move Pair Word



VFPV

### Syntax:

```
vmov.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

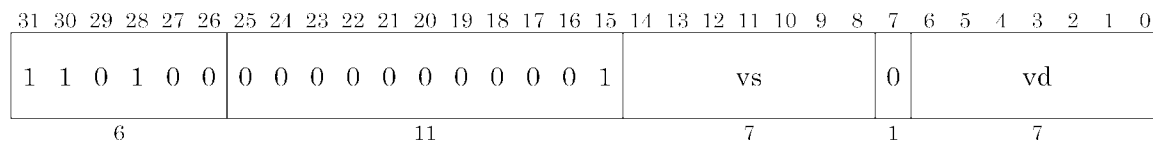
The floating-point values of two elements from the matrix registers indicated by vs are stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d <- s;
WriteMatrix( PAIRWORD, vd, d );
```

## vmov.t

Move Triple Word



VFPV

### Syntax:

```
vmov.t vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

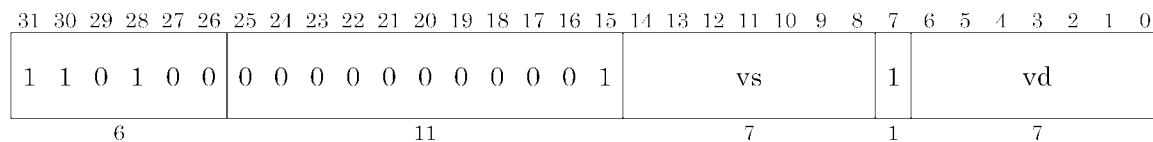
The floating-point values of three elements from the matrix registers indicated by vs are stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d <- s;
WriteMatrix( TRIPLEWORD, vd, d );
```

## vmov.q

Move Quad Word



VFPV

### Syntax:

```
vmov.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The floating-point values of four elements from the matrix registers indicated by vs are stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d <- s;
WriteMatrix( QUADWORD, vd, d );
```



## vmscl.p

### Scale Pair x Pair Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	vt							0	vs							1	vd						
6						3			7							1	7							1	7						

VFPU

### Syntax:

```
vmscl.p vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

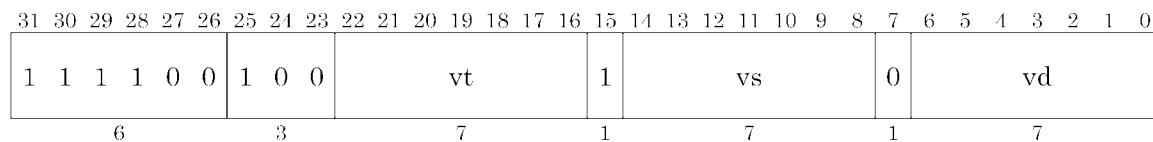
The elements of the 2x2 matrix from the matrix registers indicated by vs are multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point values. The 2x2 matrix floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
d[4] <- s[4] * t[0];
d[5] <- s[5] * t[0];
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

## vmscl.t

Scale Triple x Triple Matrix



VFPU

### Syntax:

```
vmscl.t vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

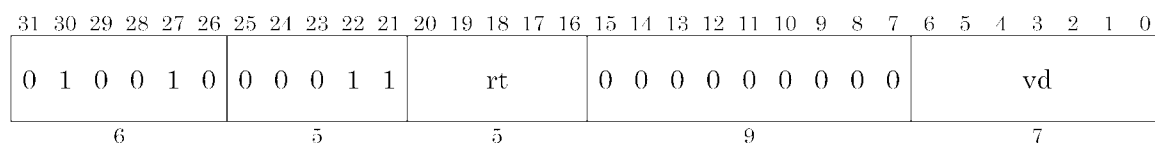
The elements of the 3x3 matrix from the matrix registers indicated by vs are multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point values. The 3x3 matrix floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
d[2] <- s[2] * t[0];
d[4] <- s[4] * t[0];
d[5] <- s[5] * t[0];
d[6] <- s[6] * t[0];
d[8] <- s[8] * t[0];
d[9] <- s[9] * t[0];
d[10] <- s[10] * t[0];
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

## mfv

Move Word from VFPU



VFPU

### Syntax:

```
mfv rt, vd
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 6

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

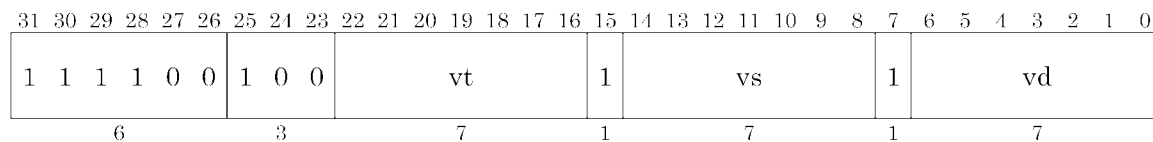
The contents of the matrix register indicated by vd are copied to CPU general-purpose register rt.

### Operation:

```
dataword <- ReadMatrix( SINGLEWORD, vd );
GPR[rt] <- dataword;
```

## vmscl.q

Scale Quad x Quad Matrix



VFPU

### Syntax:

```
vmscl.q vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

The elements of the 4x4 matrix from the matrix registers indicated by vs are multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point values. The 4x4 matrix floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADXQUADWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
d[2] <- s[2] * t[0];
d[3] <- s[3] * t[0];
d[4] <- s[4] * t[0];
d[5] <- s[5] * t[0];
d[6] <- s[6] * t[0];
d[7] <- s[7] * t[0];
d[8] <- s[8] * t[0];
d[9] <- s[9] * t[0];
```

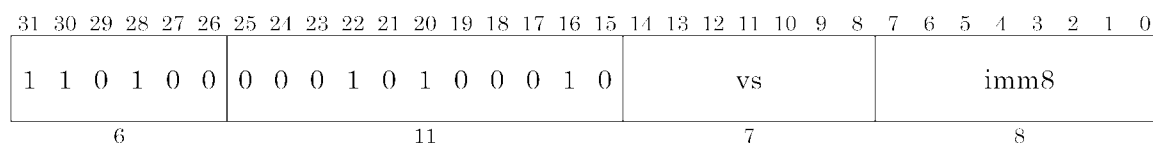
```

d[10] <- s[10] * t[0];
d[11] <- s[11] * t[0];
d[12] <- s[12] * t[0];
d[13] <- s[13] * t[0];
d[14] <- s[14] * t[0];
d[15] <- s[15] * t[0];
WriteMatrix( QUADXQUADWORD, vd, d );

```

## vmtvc

### VFPU Move Word to VFPU Control



VFPU

#### Syntax:

```
vmtvc imm8, vs
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

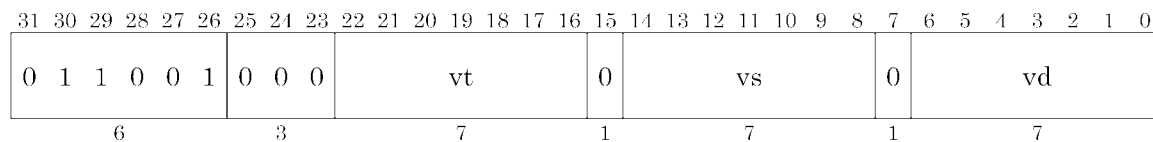
The floating-point value of one element from the matrix register indicated by vs is stored in the VFPU control register indicated by the imm8 field.

#### Operation:

```
dataword <- ReadMatrix( SINGLEWORD, vs );
WriteControl( imm8, dataword );
```

## vmul.s

Multiply Single Word



VFPU

### Syntax:

```
vmul.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

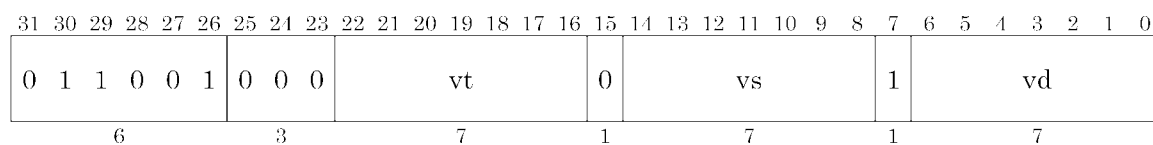
One element from the matrix register indicated by vs is multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
WriteMatrix( SINGLEWORD, vd, d );
```

## vmul.p

Multiply Pair Word



VFPU

### Syntax:

```
vmul.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Two elements from the matrix registers indicated by vs are multiplied by two elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

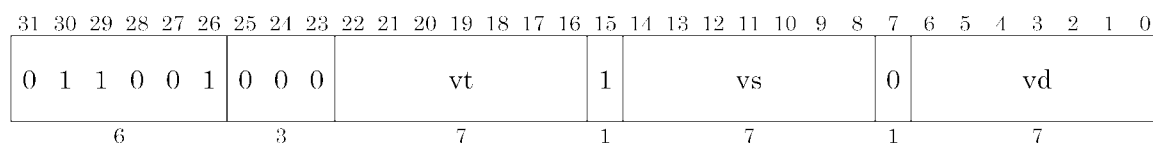
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[1];
WriteMatrix( PAIRWORD, vd, d );
```



## vmul.t

Multiply Triple Word



VFPU

### Syntax:

```
vmul.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

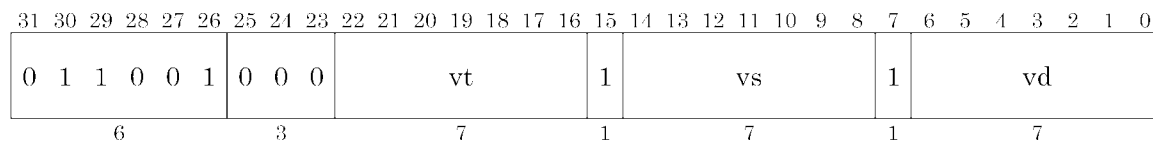
Three elements from the matrix registers indicated by vs are multiplied by three elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[1];
d[2] <- s[2] * t[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vmul.q

Multiply Quad Word



VFPU

### Syntax:

```
vmul.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Four elements from the matrix registers indicated by vs are multiplied by four elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[1];
d[2] <- s[2] * t[2];
d[3] <- s[3] * t[3];
WriteMatrix( QUADWORD, vd, d );
```

## vmzero.p

### Zero Pair x Pair Matrix

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	vd						
6						19																			7						

VFPU

### Syntax:

```
vmzero.p vd
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 4      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

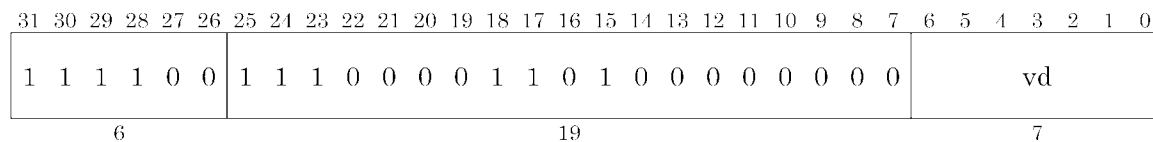
The 2x2 zero matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

### Operation:

```
d[0] <- 0.0;
d[1] <- 0.0;
d[4] <- 0.0;
d[5] <- 0.0;
WriteMatrix( PAIRXPAIRWORD, vd, d );
```

## vmzero.t

### Zero Triple x Triple Matrix



VFPU

#### Syntax:

```
vmzero.t vd
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 5      pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

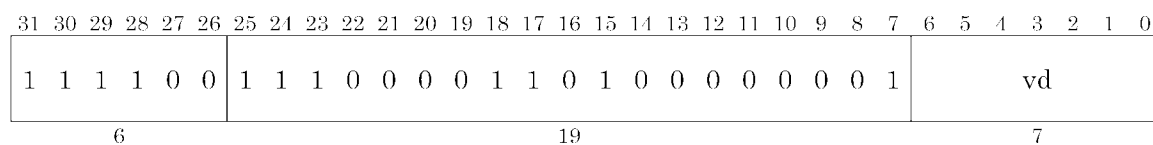
The 3x3 zero matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

#### Operation:

```
d[0] <- 0.0;
d[1] <- 0.0;
d[2] <- 0.0;
d[4] <- 0.0;
d[5] <- 0.0;
d[6] <- 0.0;
d[8] <- 0.0;
d[9] <- 0.0;
d[10] <- 0.0;
WriteMatrix( TRIPLEXTRIPLEWORD, vd, d );
```

## vmzero.q

Zero Quad x Quad Matrix



VFPU

### Syntax:

```
vmzero.q vd
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 6      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

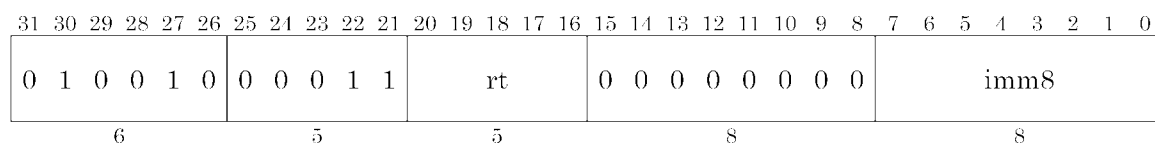
The 4x4 zero matrix is generated and stored at locations in the matrix register file indicated by vd. Elements are stored as floating-point values.

### Operation:

```
d[0]  <- 0.0;
d[1]  <- 0.0;
d[2]  <- 0.0;
d[3]  <- 0.0;
d[4]  <- 0.0;
d[5]  <- 0.0;
d[6]  <- 0.0;
d[7]  <- 0.0;
d[8]  <- 0.0;
d[9]  <- 0.0;
d[10] <- 0.0;
d[11] <- 0.0;
d[12] <- 0.0;
d[13] <- 0.0;
d[14] <- 0.0;
d[15] <- 0.0;
```

## mfvc

Move Word from VFPU Control



VFPU

### Syntax:

```
mfvc rt, imm8
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 6

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

The contents of the VFPU control register indicated by the imm8 field are copied to CPU general-purpose register rt.

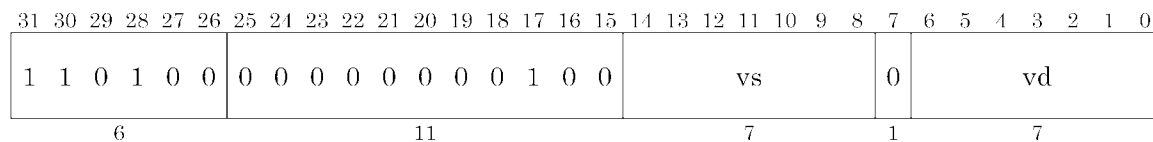
### Operation:

```
dataword <- ReadControl( imm8 );
GPR[rt] <- dataword;
```

```
WriteMatrix( QUADXQUADWORD, vd, d );
```

## vneg.s

Negate Single Word



VFPU

### Syntax:

```
vneg.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

The sign of the floating-point value of one element from the matrix register indicated by vs is inverted. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

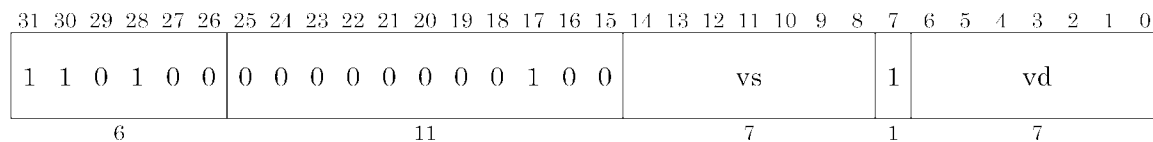
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- -s[0];
WriteMatrix( SINGLEWORD, vd, d );
```



## vneg.p

Negate Pair Word



VFPU

### Syntax:

```
vneg.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

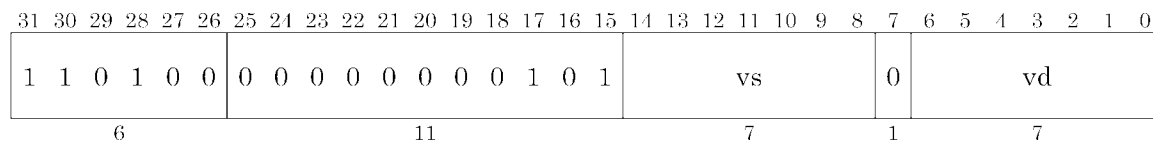
The signs of the floating-point values of two elements from the matrix registers indicated by vs are inverted. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- -s[0];
d[1] <- -s[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vneg.t

Negate Triple Word



VFPU

### Syntax:

vneg.t vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

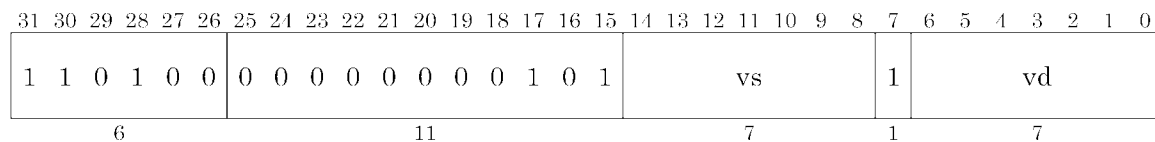
The signs of the floating-point values of three elements from the matrix registers indicated by vs are inverted. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- -s[0];
d[1] <- -s[1];
d[2] <- -s[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vneg.q

Negate Quad Word



VFPU

### Syntax:

vneg.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

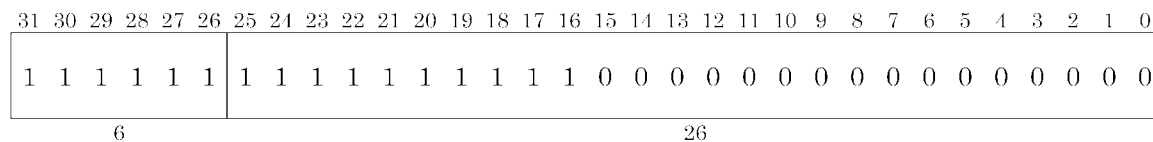
The signs of the floating-point values of four elements from the matrix registers indicated by vs are inverted. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- -s[0];
d[1] <- -s[1];
d[2] <- -s[2];
d[3] <- -s[3];
WriteMatrix( QUADWORD, vd, d );
```

## vnop

Nop



VFPU

### Syntax:

`vnop`

### Instruction Type

Synchronization instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

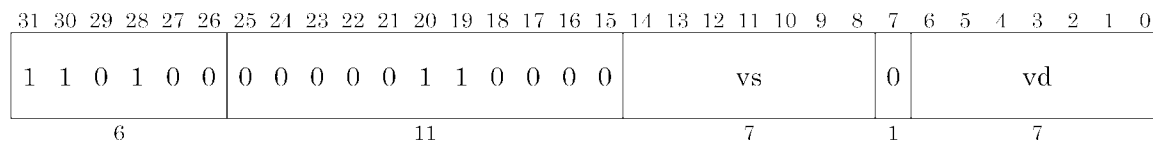
No operation is performed.

### Operation:

`Nop ( ) ;`

## vnrcp.s

Negative Reciprocal Single Word



VFPU

### Syntax:

```
vnrcp.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

The negative reciprocal of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal}(-\text{inf}) = -0.0$$

$$\text{approx\_reciprocal}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal}(-0.0) = -\text{inf}$$

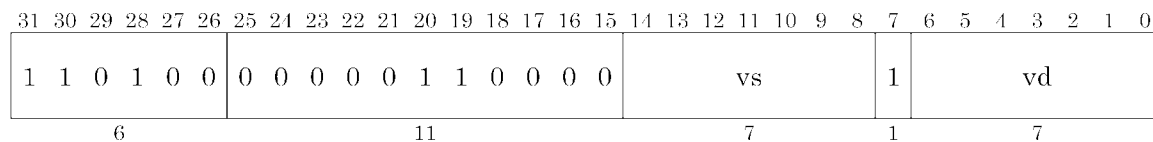
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
```

```
d[0] <- -1 * approx_reciprocal( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vnrcp.p

### Negative Reciprocal Pair Word



VFPU

#### Syntax:

```
vnrcp.p vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 8          pitch : 2

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative reciprocal of the floating-point values of two elements from the matrix registers indicated by vs is calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal}(-\text{inf}) = -0.0$$

$$\text{approx\_reciprocal}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal}(-0.0) = -\text{inf}$$

#### Operation:

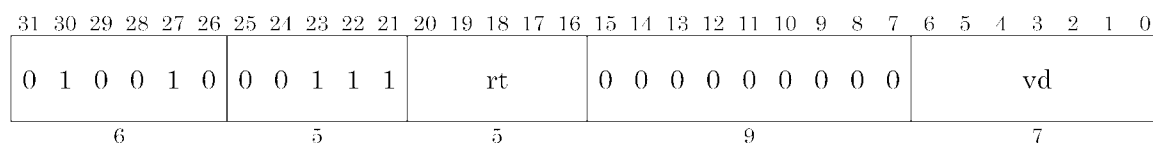
```
s <- ReadMatrix( PAIRWORD, vs );
```

```
d[0] <- -1 * approx_reciprocal( s[0] );
d[1] <- -1 * approx_reciprocal( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```



## mtv

Move Word to VFPU



VFPU

### Syntax:

```
mtv rt, vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

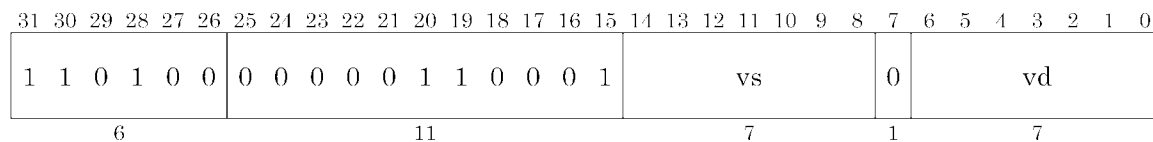
The contents of CPU general-purpose register `rt` are copied to the location in the matrix register file indicated by `vd`.

### Operation:

```
dataword <- GPR[rt];
WriteMatrix( SINGLEWORD, vd, dataword );
```

## vnrcp.t

### Negative Reciprocal Triple Word



VFPU

#### Syntax:

```
vnrcp.t vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9                  pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative reciprocal of the floating-point values of three elements from the matrix registers indicated by vs is calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal}(-\text{inf}) = -0.0$$

$$\text{approx\_reciprocal}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal}(-0.0) = -\text{inf}$$

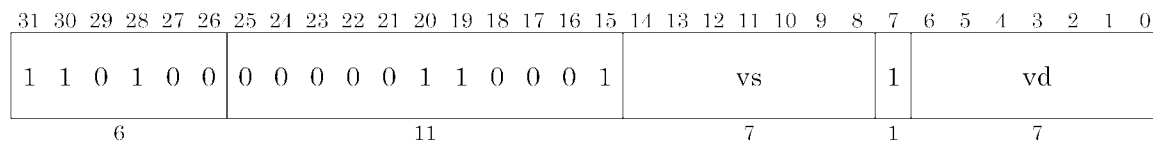
#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
```

```
d[0] <- -1 * approx_reciprocal( s[0] );
d[1] <- -1 * approx_reciprocal( s[1] );
d[2] <- -1 * approx_reciprocal( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vnrcp.q

### Negative Reciprocal Quad Word



VFPU

#### Syntax:

```
vnrcp.q vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 10      pitch : 4

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative reciprocal of the floating-point values of four elements from the matrix registers indicated by vs is calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal}(-\text{inf}) = -0.0$$

$$\text{approx\_reciprocal}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal}(-0.0) = -\text{inf}$$

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
```

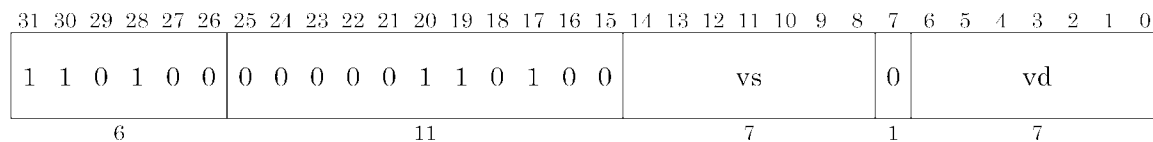
```

d[0] <- -1 * approx_reciprocal( s[0] );
d[1] <- -1 * approx_reciprocal( s[1] );
d[2] <- -1 * approx_reciprocal( s[2] );
d[3] <- -1 * approx_reciprocal( s[3] );
WriteMatrix( QUADWORD, vd, d );

```

## vnsin.s

### Negative Sine Single Word



VFPU

#### Syntax:

```
vnsin.s vd, vs
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

#### Description:

The negative sine of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

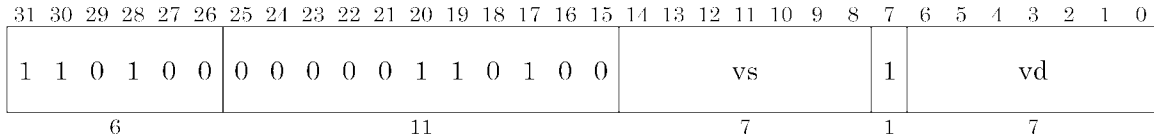
$$|(\text{approx\_sin}(x) - \sin(x)) / \sin(x)| < 2^{-20};$$

#### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- -1 * approx_sin( M_PI_2 * s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vnsin.p

### Negative Sine Pair Word



VFPU

#### Syntax:

```
vnsin.p vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 8          pitch : 2

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative sines of the floating-point values of two elements from the matrix register indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

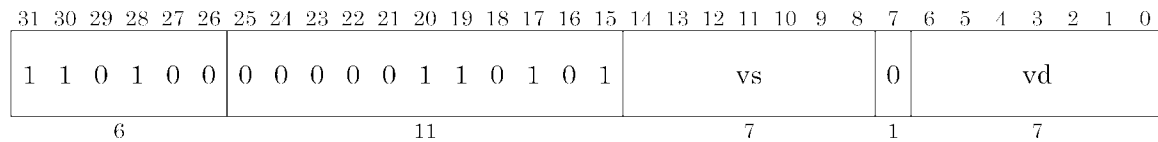
$$|(\text{approx\_sin}(x) - \sin(x)) / \sin(x)| < 2^{-20};$$

#### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- -1 * approx_sin( M_PI_2 * s[0] );
d[1] <- -1 * approx_sin( M_PI_2 * s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vnsin.t

### Negative Sine Triple Word



VFPU

#### Syntax:

```
vnsin.t vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9      pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative sines of the floating-point values of three elements from the matrix register indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$|(\text{approx\_sin}(x) - \sin(x)) / \sin(x)| < 2^{-20};$$

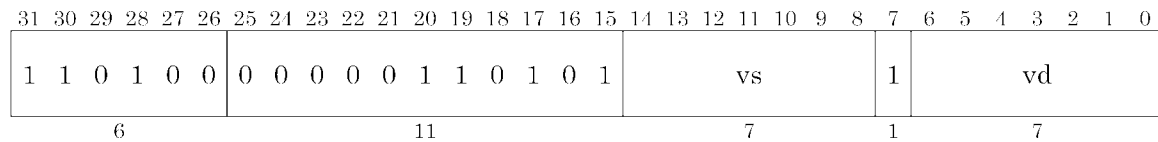
#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- -1 * approx_sin( M_PI_2 * s[0] );
d[1] <- -1 * approx_sin( M_PI_2 * s[1] );
d[2] <- -1 * approx_sin( M_PI_2 * s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```



## vnsin.q

### Negative Sine Quad Word



VFPU

#### Syntax:

```
vnsin.q vd, vs
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 10      pitch : 4

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The negative sines of the floating-point values of four elements from the matrix register indicated by vs are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

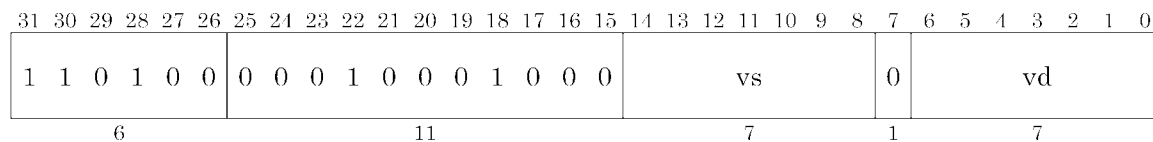
$$|(\text{approx\_sin}(x) - \sin(x)) / \sin(x)| < 2^{-20};$$

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- -1 * approx_sin( M_PI_2 * s[0] );
d[1] <- -1 * approx_sin( M_PI_2 * s[1] );
d[2] <- -1 * approx_sin( M_PI_2 * s[2] );
d[3] <- -1 * approx_sin( M_PI_2 * s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vocp.s

One's Complement Single Word



VFPU

### Syntax:

`vocp.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Valid

### Description:

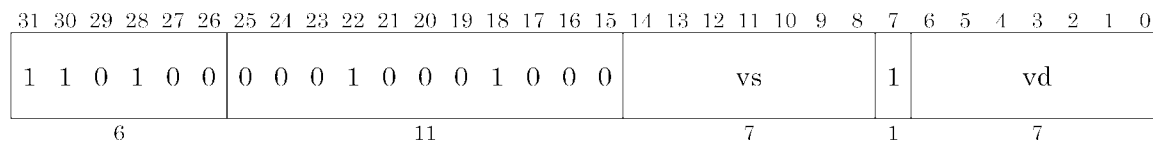
The one's complement of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- 1.0 - s[0];
WriteMatrix( SINGLEWORD, vd, d );
```

## vocp.p

One's Complement Pair Word



VFPU

### Syntax:

`vocp.p vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Valid

### Description:

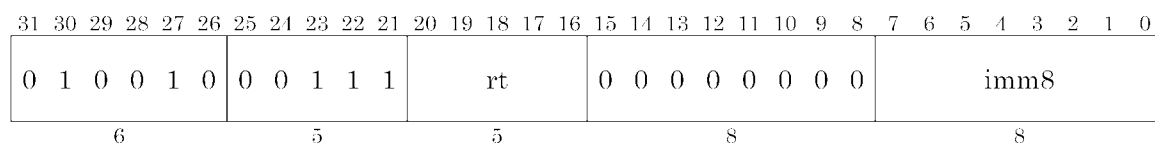
The one's complements of the floating-point values of two elements from the matrix registers indicated by `vs` are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- 1.0 - s[0];
d[1] <- 1.0 - s[1];
WriteMatrix( PAIRWORD, vd, d );
```

## mtvc

Move Word to VFPU Control



VFPU

### Syntax:

```
mtvc rt, imm8
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

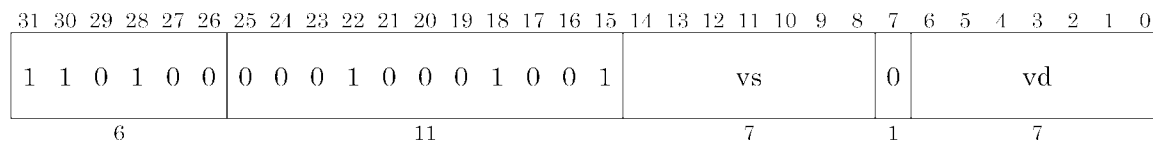
The contents of CPU general-purpose register `rt` are copied to the VFPU control register indicated by the `imm8` field.

### Operation:

```
dataword <- GPR[rt];
WriteControl( imm8, dataword );
```

## vocp.t

One's Complement Triple Word



VFPU

### Syntax:

`vocp.t vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Valid

### Description:

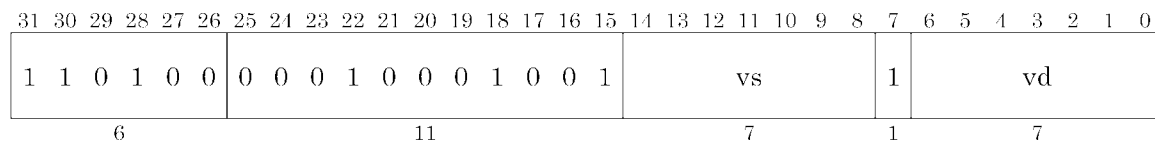
The one's complements of the floating-point values of three elements from the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- 1.0 - s[0];
d[1] <- 1.0 - s[1];
d[2] <- 1.0 - s[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vocp.q

One's Complement Quad Word



VFPU

### Syntax:

`vocp.q vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Valid

### Description:

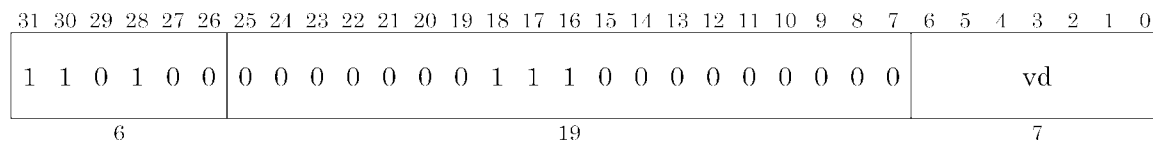
The one's complements of the floating-point values of four elements from the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- 1.0 - s[0];
d[1] <- 1.0 - s[1];
d[2] <- 1.0 - s[2];
d[3] <- 1.0 - s[3];
WriteMatrix( QUADWORD, vd, d );
```

## vone.s

Set One Single Word



VFPU

### Syntax:

`vone.s vd`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

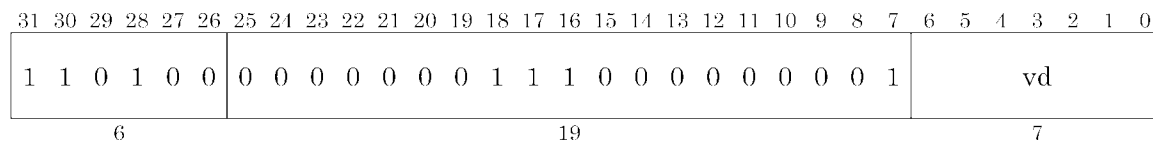
The value 1.0 is stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 1.0;
WriteMatrix( SINGLEWORD, vd, d );
```

## vone.p

Set One Pair Word



VFPU

### Syntax:

`vone.p vd`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

The value 1.0 is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

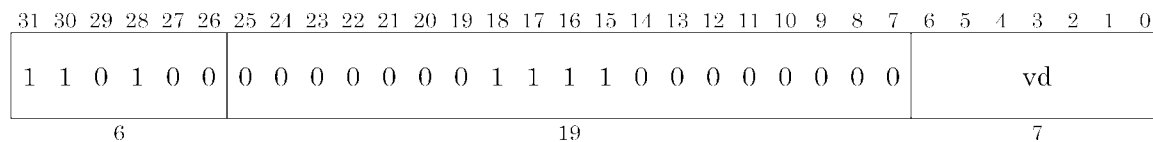
### Operation:

```
d[0] <- 1.0;
d[1] <- 1.0;
WriteMatrix( PAIRWORD, vd, d );
```



## vone.t

Set One Triple Word



VFPU

### Syntax:

`vone.t vd`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

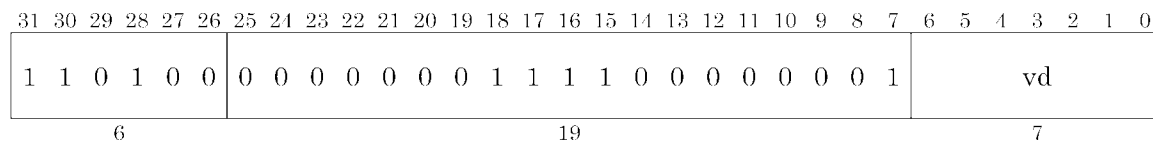
The value 1.0 is stored as a three-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 1.0;
d[1] <- 1.0;
d[2] <- 1.0;
WriteMatrix( TRIPLEWORD, vd, d );
```

## vone.q

Set One Quad Word



VFPU

### Syntax:

vone.q vd

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

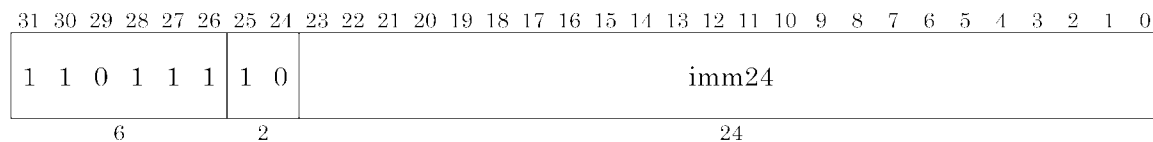
The value 1.0 is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 1.0;
d[1] <- 1.0;
d[2] <- 1.0;
d[3] <- 1.0;
WriteMatrix( QUADWORD, vd, d );
```

# vpfxd

Destination Prefix



VFPU

## Syntax:

vpfxd wpx, wpy, wpz, wpw

## Instruction Type

Prefix instruction

## Processing Time:

latency : 0      pitch : 1

## Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Overwrite

## Description:

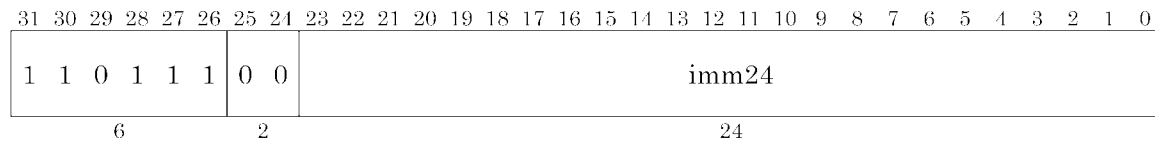
A saturation or write mask operation is applied to the destination of the following instruction, as indicated by the arguments. The prefixing operation specified by this instruction is only valid for the next VFPU instruction, excluding the b\*, mf\*, mt\*, lv\*, sv\*, vpfx\*, vsync, vnop, and vflush instructions which are not affected by prefixing.

## Operation:

```
WriteControl( VFPU_PFXD, imm24[11:0] );
```

## vpfxs

Source Prefix



VFPU

### Syntax:

```
vpfxs rpx, rpy, rpz, rpw
```

### Instruction Type

Prefix instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Overwrite	No effect	No effect

### Description:

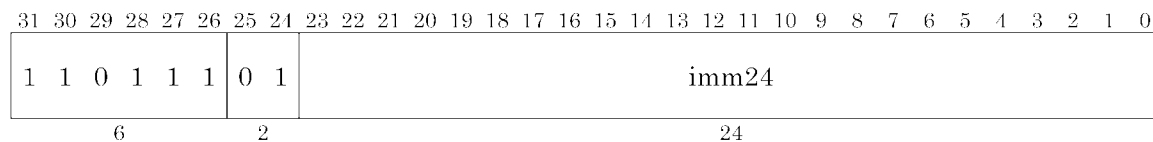
A swizzle, absolute value, constant insertion or negation operation is applied to the source of the following instruction, as indicated by the arguments. The prefixing operation specified by this instruction is only valid for the next VFPU instruction, excluding the b\*, mf\*, mt\*, lv\*, sv\*, vpfx\*, vsync, vnop, and vflush instructions which are not affected by prefixing.

### Operation:

```
WriteControl( VFPU_PFXS, imm24[19:0] );
```

## vpfxt

Target Prefix



VFPU

### Syntax:

```
vpfxt rpx, rpy, rpz, rpw
```

### Instruction Type

Prefix instruction

### Processing Time:

latency : 0      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	Overwrite	No effect

### Description:

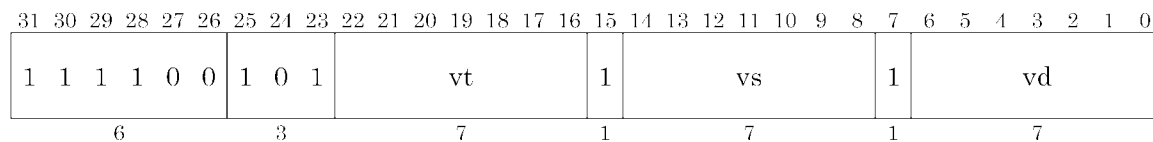
A swizzle, absolute value, constant insertion or negation operation is applied to the target of the following instruction, as indicated by the arguments. The prefixing operation specified by this instruction is only valid for the next VFPU instruction, excluding the b\*, mf\*, mt\*, lv\*, sv\*, vpfx\*, vsync, vnop, and vflush instructions which are not affected by prefixing.

### Operation:

```
WriteControl( VFPU_PFXT, imm24[19:0] );
```

## vqmul.q

### Quaternion Multiply Quad Word



VFPU

#### Syntax:

```
vqmul.q vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 10      pitch : 4

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

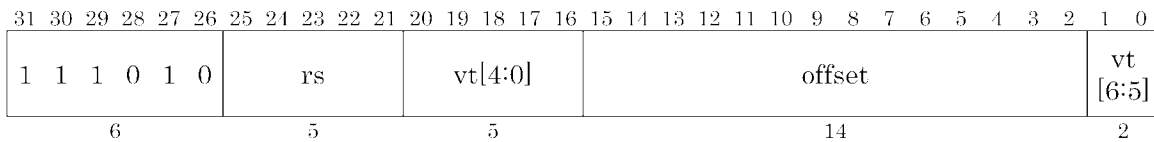
The quaternion multiplication of the four elements from the matrix registers indicated by vs and the four elements from the matrix registers indicated by vt is performed. The elements are treated as floating-point values. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- + s[0]*t[3] + s[1]*t[2] - s[2]*t[1] + s[3]*t[0];
d[1] <- - s[0]*t[2] + s[1]*t[3] + s[2]*t[0] + s[3]*t[1];
d[2] <- + s[0]*t[1] - s[1]*t[0] + s[2]*t[3] + s[3]*t[2];
d[3] <- - s[0]*t[0] - s[1]*t[1] - s[2]*t[2] + s[3]*t[3];
WriteMatrix( QUADWORD, vd, d );
```

## SV.S

### Store Single Word from VFPU



VFPU

#### Syntax:

```
sv.s vt, offset(rs)
```

#### Instruction Type

CPU interlock instruction

#### Processing Time:

latency : 0      pitch : 7

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. The single word from the location in the matrix register file indicated by vt is written to memory at this effective address.

If the effective address is not word aligned, the CPU generates an address error exception.

#### Operation:

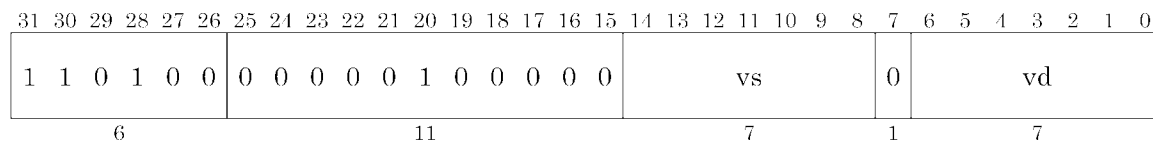
```
vAddr <- sign_extend({offset[15:2], 2'b0}) + GPR[rs];  
pAddr <- AddressTranslation(vAddr, DATA, STORE);  
dataword <- ReadMatrix( SINGLEWORD, {vt[6:5], vt[4:0]} );  
StoreMemory( SINGLEWORD, dataword, pAddr, vAddr, DATA);
```

#### Exceptions:

Address Error exception

## vrcp.s

Reciprocal Single Word



VFPU

### Syntax:

```
vrcp.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The reciprocal of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

*approx\_reciprocal(nan) = nan*

*approx\_reciprocal(+inf) = +0.0*

*approx\_reciprocal(-inf) = -0.0*

*approx\_reciprocal(+0.0) = +inf*

*approx\_reciprocal(-0.0) = -inf*

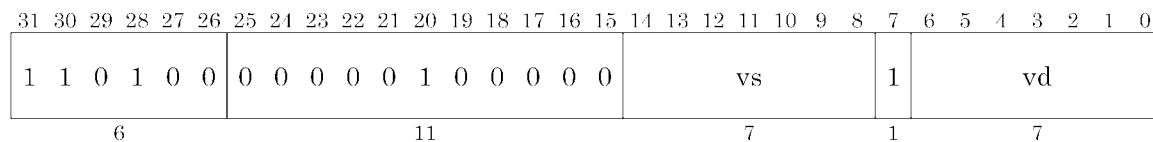
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_reciprocal( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vrcp.p

Reciprocal Pair Word



VFPU

### Syntax:

vrcp.p vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

*approx\_reciprocal(nan) = nan*

*approx\_reciprocal(+inf) = +0.0*

*approx\_reciprocal(-inf) = -0.0*

*approx\_reciprocal(+0.0) = +inf*

*approx\_reciprocal(-0.0) = -inf*

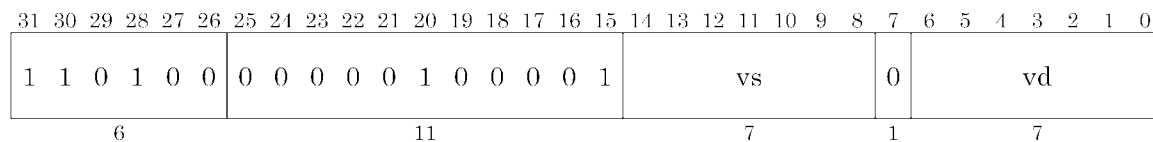
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
```

```
d[0] <- approx_reciprocal( s[0] );
d[1] <- approx_reciprocal( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vrcp.t

Reciprocal Triple Word



VFPU

### Syntax:

`vrcp.t vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the floating-point values of three elements from the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

*approx\_reciprocal(nan) = nan*

*approx\_reciprocal(+inf) = +0.0*

*approx\_reciprocal(-inf) = -0.0*

*approx\_reciprocal(+0.0) = +inf*

*approx\_reciprocal(-0.0) = -inf*

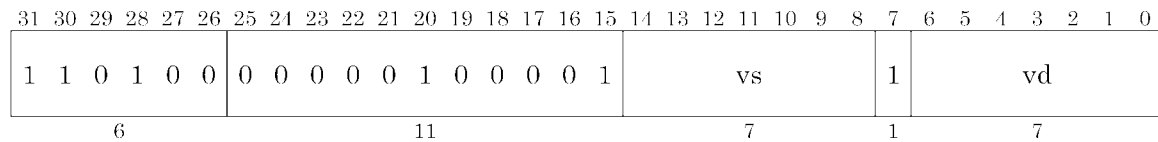
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
```

```
d[0] <- approx_reciprocal( s[0] );
d[1] <- approx_reciprocal( s[1] );
d[2] <- approx_reciprocal( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vrcp.q

Reciprocal Quad Word



VFPU

### Syntax:

`vrcp.q vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the floating-point values of four elements from the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal}(x) - (1/x) | < 2^{-20}, 1.0 \leq x < 2.0$$

Special solutions are as follows.

*approx\_reciprocal(nan) = nan*

*approx\_reciprocal(+inf) = +0.0*

*approx\_reciprocal(-inf) = -0.0*

*approx\_reciprocal(+0.0) = +inf*

*approx\_reciprocal(-0.0) = -inf*

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
```

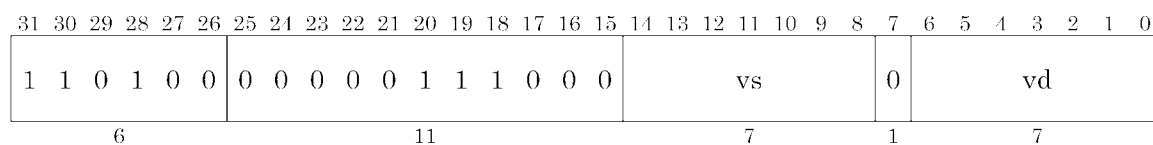
```

d[0] <- approx_reciprocal( s[0] );
d[1] <- approx_reciprocal( s[1] );
d[2] <- approx_reciprocal( s[2] );
d[3] <- approx_reciprocal( s[3] );
WriteMatrix( QUADWORD, vd, d );

```

## vrex2.s

Reciprocal Exponential base 2 Single Word



VFPU

### Syntax:

`vrex2.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

The reciprocal of the base 2 exponential of the floating-point value of one element from the matrix register indicated by *vs* is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by *vd*.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

$$\text{approx\_exp2}(-0.0) = +1.0$$

### Operation:

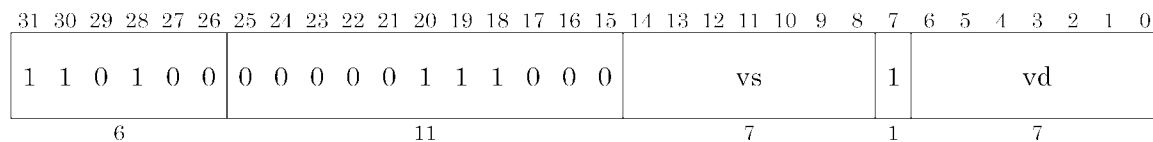
```
s <- ReadMatrix( SINGLEWORD, vs );
```

```
d[0] <- 1 / approx_exp2( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vrexp2.p

Reciprocal Exponential base 2 Pair Word



VFPU

### Syntax:

```
vrexp2.p vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the base 2 exponentials of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

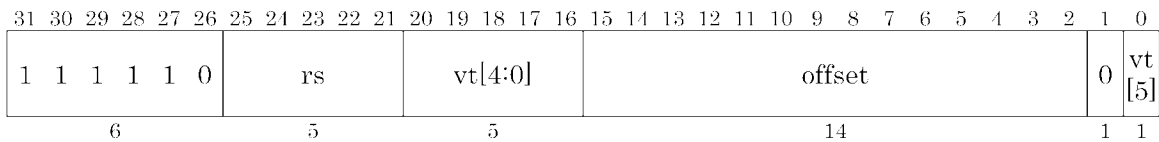
$$\text{approx\_exp2}(-0.0) = +1.0$$

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
```

## sv.q

Store Quad Word from VFPU



VFPU

### Syntax:

```
sv.q vt, offset(rs)
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 7

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. The quadword from locations in the matrix register file indicated by vt is written to memory at this effective address.

If the effective address is not quadword aligned, the CPU generates an address error exception.

### Operation:

```
vAddr <- sign_extend({offset[15:2], 2'b0}) + GPR[rs];
pAddr <- AddressTranslation(vAddr, DATA, STORE);
dataword <- ReadMatrix( QUADWORD, {vt[5], vt[4:0]} );
StoreMemory( QUADWORD, dataword, pAddr, vAddr, DATA);
```

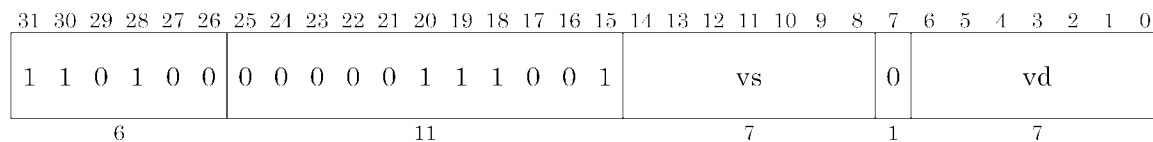
### Exceptions:

Address Error exception

```
d[0] <- 1 / approx_exp2( s[0] );
d[1] <- 1 / approx_exp2( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vrex2.t

Reciprocal Exponential base 2 Triple Word



VFPU

### Syntax:

```
vrex2.t vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the base 2 exponentials of the floating-point values of three elements from the matrix registers indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd. The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

$$\text{approx\_exp2}(-0.0) = +1.0$$

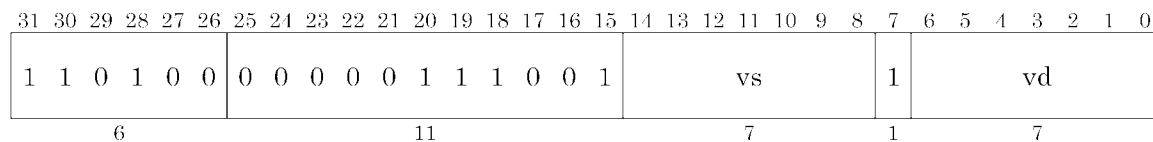
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
```

```
d[0] <- 1 / approx_exp2( s[0] );
d[1] <- 1 / approx_exp2( s[1] );
d[2] <- 1 / approx_exp2( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vrexp2.q

Reciprocal Exponential base 2 Quad Word



VFPU

### Syntax:

```
vrexp2.q vd, vs
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the base 2 exponentials of the floating-point values of four elements from the matrix registers indicated by vs are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_exp2}(x) - 2^x | < 2^{-20} ; 0.0 \leq x < 1.0$$

Special solutions are as follows.

$$\text{approx\_exp2}(\text{nan}) = \text{nan}$$

$$\text{approx\_exp2}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_exp2}(-\text{inf}) = +0.0$$

$$\text{approx\_exp2}(+0.0) = +1.0$$

$$\text{approx\_exp2}(-0.0) = +1.0$$

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
```

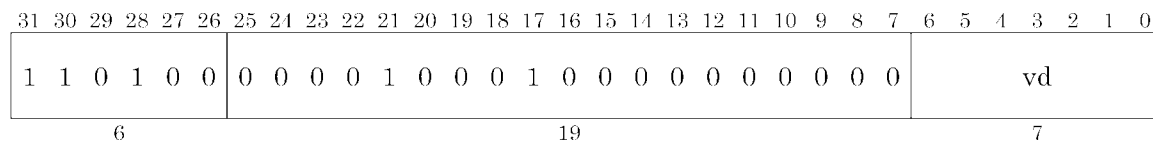
```

d[0] <- 1 / approx_exp2( s[0] );
d[1] <- 1 / approx_exp2( s[1] );
d[2] <- 1 / approx_exp2( s[2] );
d[3] <- 1 / approx_exp2( s[3] );
WriteMatrix( QUADWORD, vd, d );

```

## vrndf1.s

Random Floating Single Word



VFPU

### Syntax:

```
vrndf1.s vd
```

### Instruction Type

Multi-cycle instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

One pseudorandom number is generated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$1.0 \leq \text{random}() < 2.0$$

The period is  $\geq 10^{38}$ .

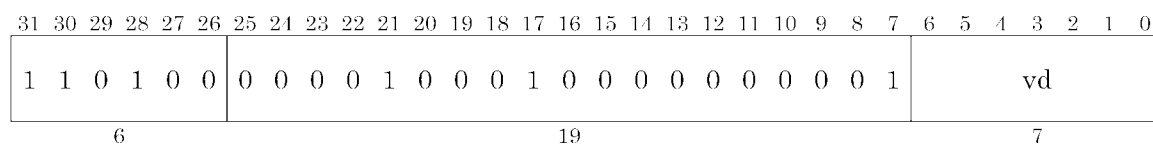
### Operation:

```
d[0] <- random();
WriteMatrix( SINGLEWORD, vd, d );
```



## vrndf1.p

Random Floating Pair Word



VFPU

### Syntax:

```
vrndf1.p vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 8      pitch : 6

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Two pseudorandom numbers are generated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$1.0 \leq \text{random}() < 2.0$$

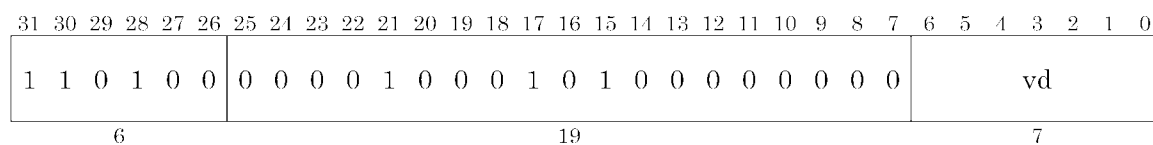
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
WriteMatrix( PAIRWORD, vd, d );
```

## vrndf1.t

Random Floating Triple Word



VFPU

### Syntax:

```
vrndf1.t vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 11      pitch : 9

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Three pseudorandom numbers are generated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$1.0 \leq \text{random}() < 2.0$$

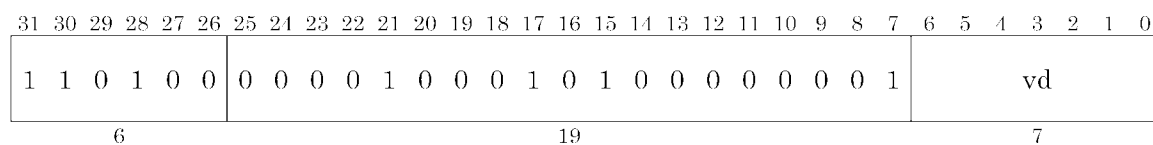
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
WriteMatrix( TRIPLEWORD, vd, d );
```

## vrndf1.q

Random Floating Quad Word



VFPU

### Syntax:

```
vrndf1.q vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 14      pitch : 12

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Four pseudorandom numbers are generated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$1.0 \leq \text{random}() < 2.0$$

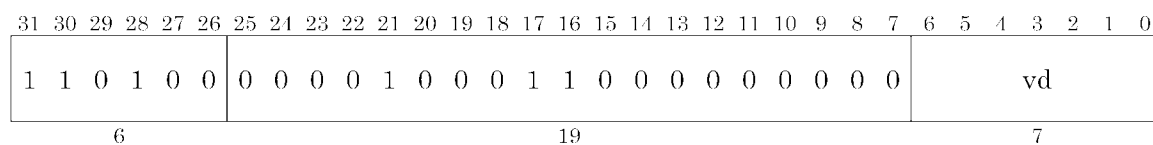
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
d[3] <- random();
WriteMatrix( QUADWORD, vd, d );
```

## vrndf2.s

Random Floating Single Word



VFPU

### Syntax:

```
vrndf2.s vd
```

### Instruction Type

Multi-cycle instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

One pseudorandom number is generated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$2.0 \leq \text{random}() < 4.0$$

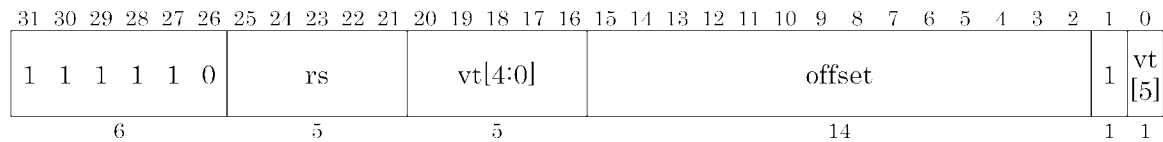
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
WriteMatrix( SINGLEWORD, vd, d );
```

## sv.q

### Store Quad Word to Write Buffer



VFPU

#### Syntax:

```
sv.q vt, offset(rs), wb
```

#### Instruction Type

Pipeline (non-cached) / CPU interlock (cached) instruction

#### Processing Time:

latency : 0      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. If this effective address is in the cached space, the quadword from locations in the matrix register file indicated by vt is written to memory at that address. At the same time, the address is converted to a physical address and the quadword is also written to physical memory via the write buffer. If the effective address is in the non-cached space, it is converted to a physical address, and the quadword is only written to physical memory via the write buffer.

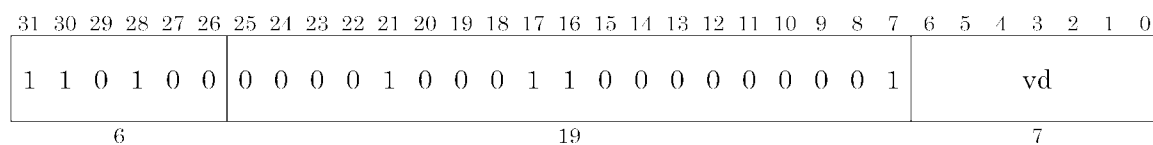
If the effective address is not quadword aligned, the CPU generates an address error exception.

#### Operation:

```
vAddr <- sign_extend({offset[15:2], 2'b0}) + GPR[rs];
pAddr <- AddressTranslation(vAddr, DATA, STORE);
dataword <- ReadMatrix( QUADWORD, {vt[5], vt[4:0]} );
```

## vrndf2.p

Random Floating Pair Word



VFPU

### Syntax:

```
vrndf2.p vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 8      pitch : 6

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Two pseudorandom numbers are generated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$2.0 \leq \text{random}() < 4.0$$

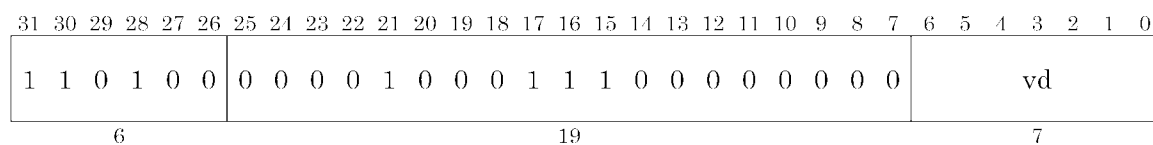
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
WriteMatrix( PAIRWORD, vd, d );
```

## vrndf2.t

Random Floating Triple Word



VFPU

### Syntax:

```
vrndf2.t vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 11      pitch : 9

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Three pseudorandom numbers are generated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$2.0 \leq \text{random}() < 4.0$$

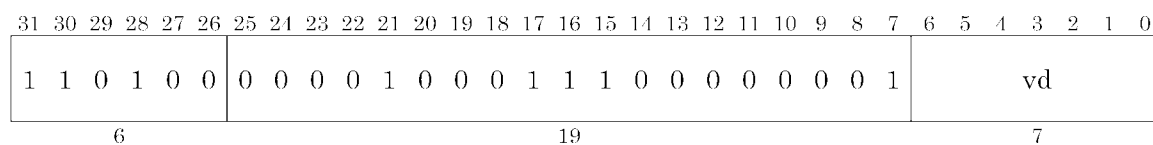
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
WriteMatrix( TRIPLEWORD, vd, d );
```

## vrndf2.q

Random Floating Quad Word



VFPU

### Syntax:

```
vrndf2.q vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 14      pitch : 12

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Four pseudorandom numbers are generated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$$2.0 \leq \text{random}() < 4.0$$

The period is  $\geq 10^{38}$ .

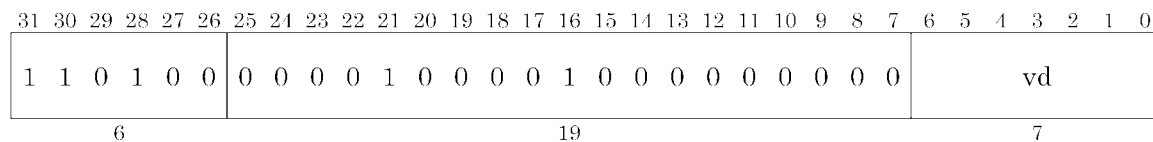
### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
d[3] <- random();
WriteMatrix( QUADWORD, vd, d );
```



## vrndi.s

Random Integer Single Word



VFPU

### Syntax:

```
vrndi.s vd
```

### Instruction Type

Multi-cycle instruction

### Processing Time:

latency : 5      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

One pseudorandom number is generated. The one-element integer result is stored at the location in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$0x00000000 \leq \text{random}() \leq 0xFFFFFFFF$

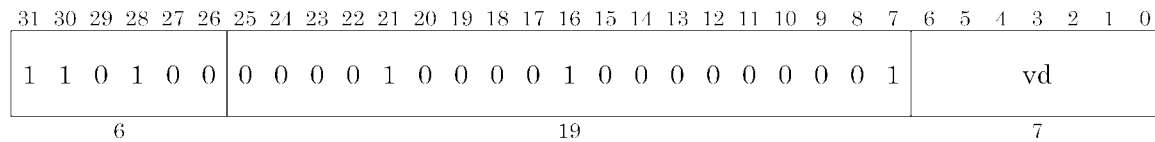
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
WriteMatrix( SINGLEWORD, vd, d );
```

## vrndi.p

Random Integer Pair Word



VFPU

### Syntax:

```
vrndi.p vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 8      pitch : 6

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Two pseudorandom numbers are generated. The two-element integer result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$0x00000000 \leq \text{random}() \leq 0xFFFFFFFF$

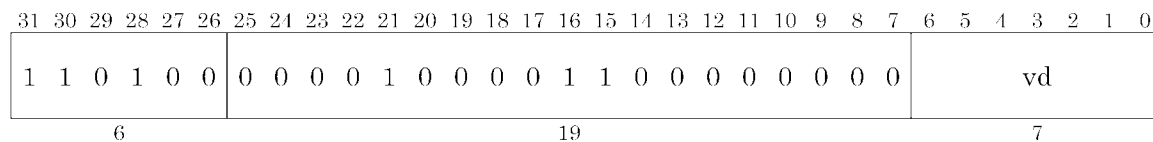
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
WriteMatrix( PAIRWORD, vd, d );
```

## vrndi.t

Random Integer Triple Word



VFPU

### Syntax:

```
vrndi.t vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 11      pitch : 9

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Three pseudorandom numbers are generated. The three-element integer result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$0x00000000 \leq \text{random}() \leq 0xFFFFFFFF$

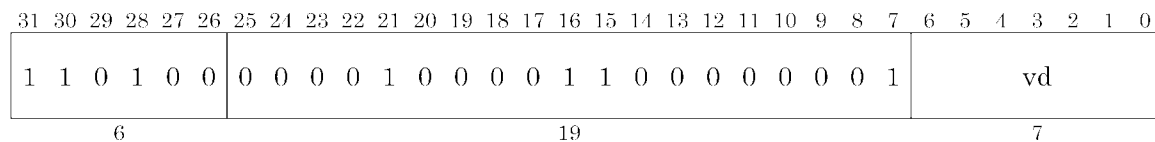
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
WriteMatrix( TRIPLEWORD, vd, d );
```

## vrndi.q

Random Integer Quad Word



VFPU

### Syntax:

```
vrndi.q vd
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 14      pitch : 12

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Use prohibited

### Description:

Four pseudorandom numbers are generated. The four-element integer result is stored at locations in the matrix register file indicated by vd.

The range of random numbers is given by the following expression.

$0x00000000 \leq \text{random}() \leq 0xFFFFFFFF$

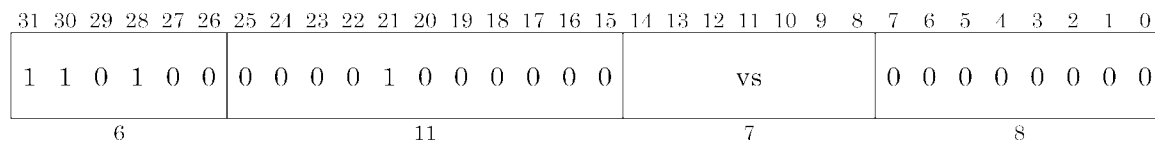
The period is  $\geq 10^{38}$ .

### Operation:

```
d[0] <- random();
d[1] <- random();
d[2] <- random();
d[3] <- random();
WriteMatrix( QUADWORD, vd, d );
```

## vrnds.s

Random Seed Single Word



VFPU

### Syntax:

```
vrnds.s  vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

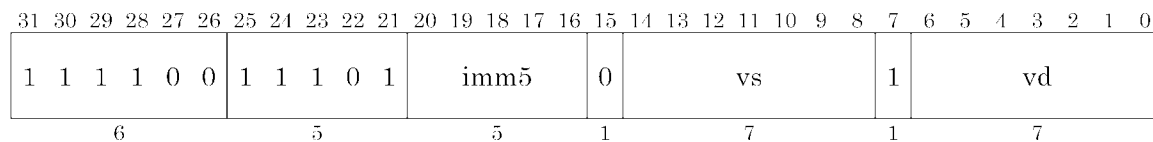
The seed of the pseudorandom number generator is set with the integer value of one element from the matrix register indicated by vs. This value must be an integer other than zero.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
srand( s[0] );
```

## vrot.p

Rotator Pair Word



VFPU

### Syntax:

vrot.p vd, vs, imm5

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The rotators indicated by the imm5 field are calculated for the floating-point values of two elements from the matrix registers indicated by vs. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

If the write positions overlap, the cos values are overwritten.

The following mnemonics can be used for imm5.

Code (imm5)	Mnemonic
0	[C,S]
1	[S,C]
2	[S,0]
3	[S,0]
4	[C,S]
5	[S,C]
6	[0,S]
7	[0,S]
8	[C,0]
9	[0,C]

Code (imm5)	Mnemonic
10	[S,S]
11	[0,0]
12	[C,0]
13	[0,C]
14	[0,0]
15	[S,S]
16	[C,-S]
17	[-S,C]
18	[-S,0]
19	[-S,0]
20	[C,-S]
21	[-S,C]
22	[0,-S]
23	[0,-S]
24	[C,0]
25	[0,C]
26	[-S,-S]
27	[0,0]
28	[C,0]
29	[0,C]
30	[0,0]
31	[-S,-S]

**Operation:**

```

s <- ReadMatrix( SINGLEWORD, vs );
ts <- approx_sin( M_PI_2 * s[0] );
tc <- approx_cos( M_PI_2 * s[0] );
d[0] <- 0;
d[1] <- 0;
if( imm5[4] )
    ts <- -ts;
if( imm5[3:2]==imm5[1:0] )
    begin
        d[0] <- ts;
        d[1] <- ts;
    end
else if( imm5[3:2] <2)
    d[imm5[3:2]] <- ts;
if( imm5[1:0] <2)
    d[(imm5[1:0]] <- tc;
WriteMatrix( PAIRWORD, vd, d );

```

```

if( isCacheSpace( pAddr ) )
    StoreMemory( QUADWORD, dataword, pAddr, vAddr, DATA);
StoreMemory_WriteBuffer( QUADWORD, dataword, pAddr, vAddr, DATA);

```

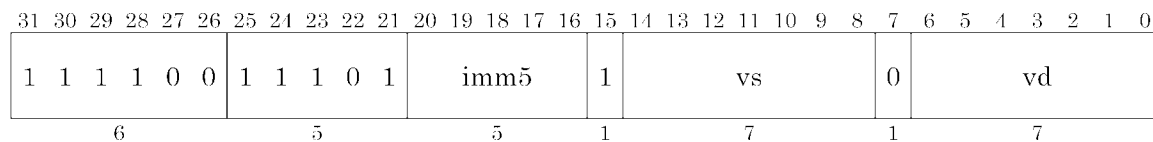
### Exceptions:

Address Error exception



## vrot.t

Rotator Triple Word



VFPU

### Syntax:

```
vrot.t vd, vs, imm5
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The rotators indicated by the imm5 field are calculated for the floating-point values of three elements from the matrix registers indicated by vs. The three-element floating-point result is stored at locations in the matrix register file indicated by vd. If the write positions overlap, the cos values are overwritten. The following mnemonics can be used for imm5.

Code (imm5)	Mnemonic
0	[C,S,S]
1	[S,C,0]
2	[S,0,C]
3	[S,0,0]
4	[C,S,0]
5	[S,C,S]
6	[0,S,C]
7	[0,S,0]
8	[C,0,S]
9	[0,C,S]

Code (imm5)	Mnemonic
10	[S,S,C]
11	[0,0,S]
12	[C,0,0]
13	[0,C,0]
14	[0,0,C]
15	[S,S,S]
16	[C,-S,-S]
17	[-S,C,0]
18	[-S,0,C]
19	[-S,0,0]
20	[C,-S,0]
21	[-S,C,-S]
22	[0,-S,C]
23	[0,-S,0]
24	[C,0,-S]
25	[0,C,-S]
26	[-S,-S,C]
27	[0,0,-S]
28	[C,0,0]
29	[0,C,0]
30	[0,0,C]
31	[-S,-S,-S]

**Operation:**

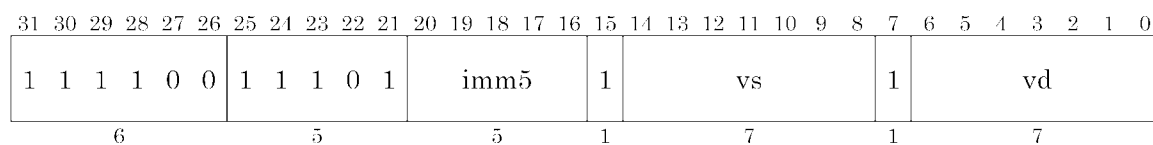
```

s <- ReadMatrix( SINGLEWORD, vs );
ts <- approx_sin( M_PI_2 * s[0] );
tc <- approx_cos( M_PI_2 * s[0] );
d[0] <- 0;
d[1] <- 0;
d[2] <- 0;
if( imm5[4] )
    ts <- -ts;
if( imm5[3:2]==imm5[1:0] )
    begin
        d[0] <- ts;
        d[1] <- ts;
        d[2] <- ts;
    end
else if( imm5[3:2] <3)
    d[imm5[3:2]] <- ts;
if( imm5[1:0] <3)
    d[imm5[1:0]] <- tc;
WriteMatrix( TRIPLEWORD, vd, d );

```

## vrot.q

Rotator Quad Word



VFPU

### Syntax:

```
vrot.q vd, vs, imm5
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8                  pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The rotators indicated by the imm5 field are calculated for the floating-point values of four elements from the matrix registers indicated by vs. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

If the write positions overlap, the cos values are overwritten.

The following mnemonics can be used for imm5.

Code (imm5)	Mnemonic
0	[C,S,S,S]
1	[S,C,0,0]
2	[S,0,C,0]
3	[S,0,0,C]
4	[C,S,0,0]
5	[S,C,S,S]
6	[0,S,C,0]
7	[0,S,0,C]
8	[C,0,S,0]
9	[0,C,S,0]

Code (imm5)	Mnemonic
10	[S,S,C,S]
11	[0,0,S,C]
12	[C,0,0,S]
13	[0,C,0,S]
14	[0,0,C,S]
15	[S,S,S,C]
16	[C,-S,-S,-S]
17	[-S,C,0,0]
18	[-S,0,C,0]
19	[-S,0,0,C]
20	[C,-S,0,0]
21	[-S,C,-S,-S]
22	[0,-S,C,0]
23	[0,-S,0,C]
24	[C,0,-S,0]
25	[0,C,-S,0]
26	[-S,-S,C,-S]
27	[0,0,-S,C]
28	[C,0,0,-S]
29	[0,C,0,-S]
30	[0,0,C,-S]
31	[-S,-S,-S,C]

#### Operation:

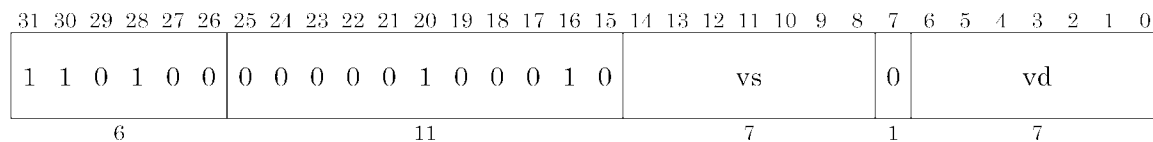
```

s <- ReadMatrix( SINGLEWORD, vs );
ts <- approx_sin( M_PI_2 * s[0] );
tc <- approx_cos( M_PI_2 * s[0] );
d[0] <- 0;
d[1] <- 0;
d[2] <- 0;
d[3] <- 0;
if( imm5[4] )
    ts <- -ts;
if( imm5[3:2]==imm5[1:0] )
    begin
        d[0] <- ts;
        d[1] <- ts;
        d[2] <- ts;
        d[3] <- ts;
    end
else if( imm5[3:2] <4)
    d[imm5[3:2]] <- ts;
if( imm5[1:0] <4)
    d[(imm5[1:0]] <- tc;
WriteMatrix( QUADWORD, vd, d );

```

## vrsq.s

Reciprocal Square Root Single Word



VFPU

### Syntax:

`vrsq.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The reciprocal of the square root of the floating-point value of one element from the matrix register indicated by `vs` is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal\_sqrt}(x) - (1/\sqrt{x}) | < 2^{-20}, 1.0 \leq x < 4.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal\_sqrt}(-0.0) = -\text{inf}$$

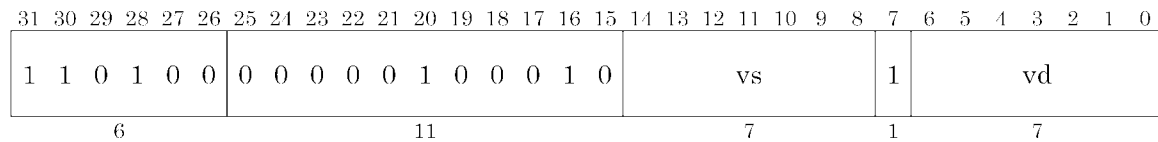
$$\text{approx\_reciprocal\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_reciprocal_sqrt( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vrsq.p

Reciprocal Square Root Pair Word



VFPU

### Syntax:

`vrsq.p vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The reciprocals of the square roots of the floating-point values of two elements from the matrix registers indicated by `vs` are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal\_sqrt}(x) - (1/\sqrt{x}) | < 2^{-20}, 1.0 \leq x < 4.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal\_sqrt}(-0.0) = -\text{inf}$$

$$\text{approx\_reciprocal\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

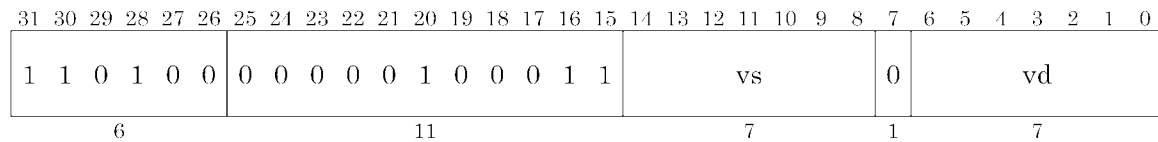
**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_reciprocal_sqrt( s[0] );
d[1] <- approx_reciprocal_sqrt( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```



## vrsq.t

### Reciprocal Square Root Triple Word



VFPU

#### Syntax:

`vrsq.t vd, vs`

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9      pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The reciprocals of the square roots of the floating-point values of three elements from the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal\_sqrt}(x) - (1/\sqrt{x}) | < 2^{-20}, 1.0 \leq x < 4.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal\_sqrt}(-0.0) = -\text{inf}$$

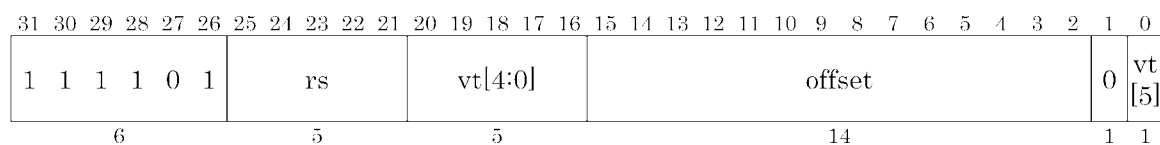
$$\text{approx\_reciprocal\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_reciprocal_sqrt( s[0] );
d[1] <- approx_reciprocal_sqrt( s[1] );
d[2] <- approx_reciprocal_sqrt( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## svl.q

Store Quad Word Left from VFPU



VFPU

### Syntax:

```
svl.q vt, offset(rs)
```

### Instruction Type

CPU interlock instruction

### Processing Time:

latency : 0      pitch : 7

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

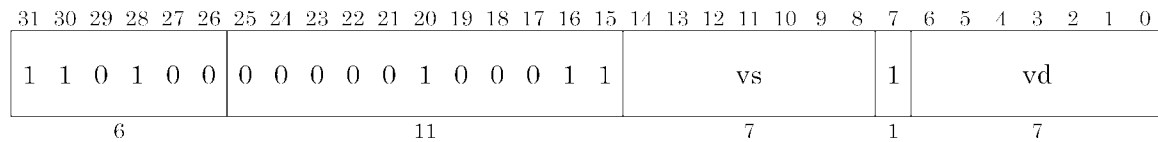
A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. From one to four words are read from locations in the matrix register file indicated by vt, then stored to memory such that the high-order word from the matrix register file is stored at the effective address, and the low-order word is stored at the end of the quadword boundary. The words are stored in memory starting with the leftmost word within the quadword from the matrix register file. Any remaining words to the right in the quadword are not stored in memory and are unaffected by this instruction. If the effective address is not word aligned, the CPU generates an address error exception.

### Operation:

```
vAddr <- sign_extend(({offset[15:2],2'b0}) + GPR[rs]);
```

## vrsq.q

### Reciprocal Square Root Quad Word



VFPU

#### Syntax:

`vrsq.q vd, vs`

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 10      pitch : 4

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

#### Description:

The reciprocals of the square roots of the floating-point values of four elements from the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$| \text{approx\_reciprocal\_sqrt}(x) - (1/\sqrt{x}) | < 2^{-20}, 1.0 \leq x < 4.0$$

Special solutions are as follows.

$$\text{approx\_reciprocal\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+\text{inf}) = +0.0$$

$$\text{approx\_reciprocal\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_reciprocal\_sqrt}(+0.0) = +\text{inf}$$

$$\text{approx\_reciprocal\_sqrt}(-0.0) = -\text{inf}$$

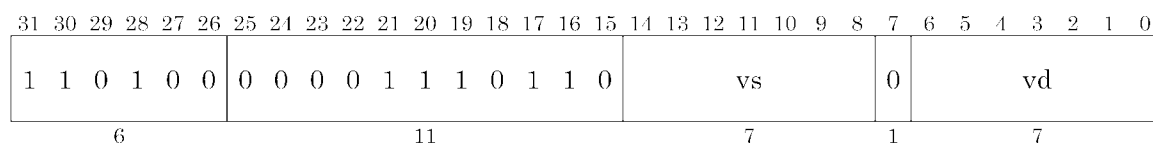
$$\text{approx\_reciprocal\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_reciprocal_sqrt( s[0] );
d[1] <- approx_reciprocal_sqrt( s[1] );
d[2] <- approx_reciprocal_sqrt( s[2] );
d[3] <- approx_reciprocal_sqrt( s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vs2i.s

Convert signed short to integer Single Word



VFPU

### Syntax:

```
vs2i.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Only write mask is valid

### Description:

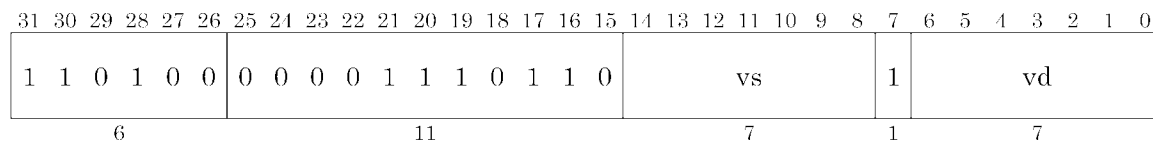
The 32-bit packed data from the matrix register indicated by vs is unpacked and converted from two signed 16-bit integers to two signed 32-bit integers. The two-element integer result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- {s[0][15: 0], 16'b0};
d[1] <- {s[0][31:16], 16'b0};
WriteMatrix( PAIRWORD, vd, d );
```

## vs2i.p

Convert signed short to integer Pair Word



VFPU

### Syntax:

```
vs2i.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Only write mask is valid

### Description:

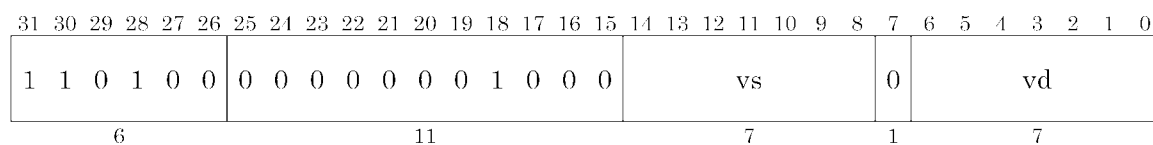
The 64-bit packed data from the matrix registers indicated by vs is unpacked and converted from four signed 16-bit integers to four signed 32-bit integers. The four-element integer result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- {s[0][15: 0], 16'b0};
d[1] <- {s[0][31:16], 16'b0};
d[2] <- {s[1][15: 0], 16'b0};
d[3] <- {s[1][31:16], 16'b0};
WriteMatrix( QUADWORD, vd, d );
```

## vsat0.s

Saturate to [0.0:1.0] Single Word



VFPU

### Syntax:

```
vsat0.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

If the floating-point value of one element from the matrix register indicated by vs is less than 0.0, it is saturated to 0.0. If the value is greater than 1.0, it is saturated to 1.0. Values between 0.0 and 1.0 are not changed. The saturated result is stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

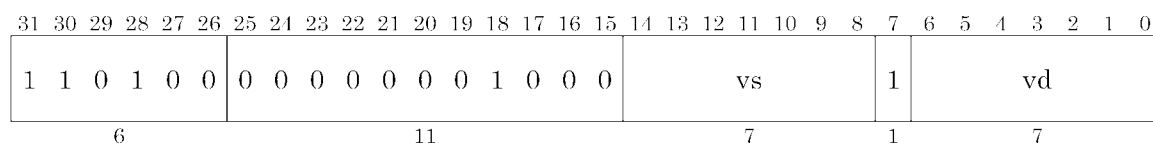
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- (s[0] < 0.0) ? 0.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
WriteMatrix( SINGLEWORD, vd, d );
```



## vsat0.p

Saturate to [0.0:1.0] Pair Word



VFPU

### Syntax:

```
vsat0.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

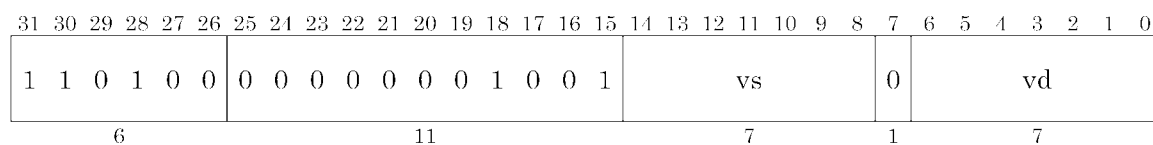
If any of the floating-point values of two elements from the matrix registers indicated by vs is less than 0.0, it is saturated to 0.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between 0.0 and 1.0 are not changed. The saturated result is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- (s[0] < 0.0) ? 0.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < 0.0) ? 0.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
WriteMatrix( PAIRWORD, vd, d );
```

## vsat0.t

Saturate to [0.0:1.0] Triple Word



VFPU

### Syntax:

```
vsat0.t vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

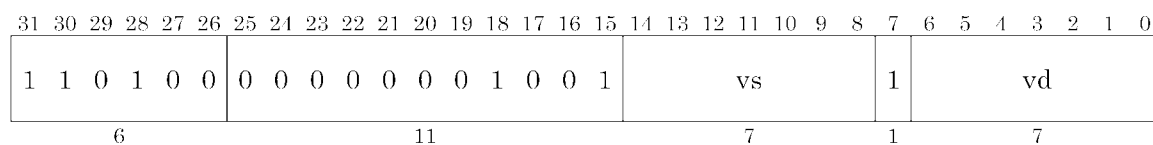
If any of the floating-point values of three elements from the matrix registers indicated by vs is less than 0.0, it is saturated to 0.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between 0.0 and 1.0 are not changed. The saturated result is stored as a three-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- (s[0] < 0.0) ? 0.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < 0.0) ? 0.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
d[2] <- (s[2] < 0.0) ? 0.0 : ( (s[2] > 1.0) ? 1.0 : s[2]);
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsat0.q

Saturate to [0.0:1.0] Quad Word



VFPU

### Syntax:

```
vsat0.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

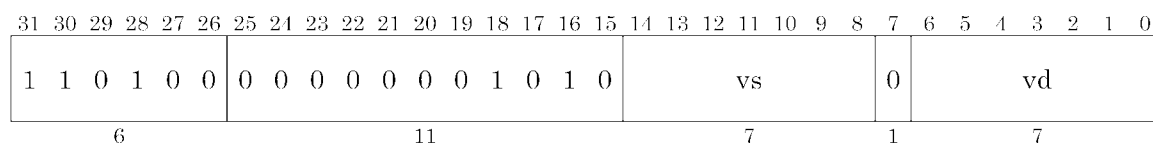
If any of the floating-point values of four elements from the matrix registers indicated by vs is less than 0.0, it is saturated to 0.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between 0.0 and 1.0 are not changed. The saturated result is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- (s[0] < 0.0) ? 0.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < 0.0) ? 0.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
d[2] <- (s[2] < 0.0) ? 0.0 : ( (s[2] > 1.0) ? 1.0 : s[2]);
d[3] <- (s[3] < 0.0) ? 0.0 : ( (s[3] > 1.0) ? 1.0 : s[3]);
WriteMatrix( QUADWORD, vd, d );
```

## vsat1.s

Saturate to [-1.0:1.0] Single Word



VFPU

### Syntax:

```
vsat1.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

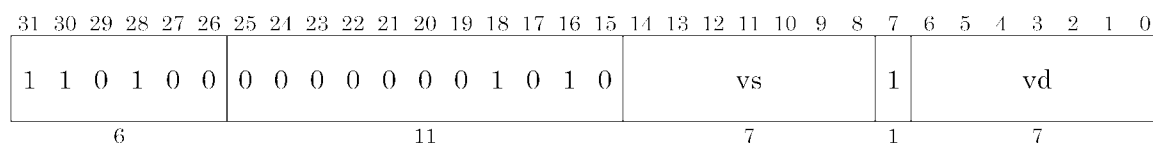
If the floating-point value of one element from the matrix register indicated by vs is less than -1.0, it is saturated to -1.0. If the value is greater than 1.0, it is saturated to 1.0. Values between -1.0 and 1.0 are not changed. The saturated result is stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- (s[0] < -1.0) ? -1.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
WriteMatrix( SINGLEWORD, vd, d );
```

## vsat1.p

Saturate to [-1.0:1.0] Pair Word



VFPU

### Syntax:

```
vsat1.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

If any of the floating-point values of two elements from the matrix registers indicated by vs is less than -1.0, it is saturated to -1.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between -1.0 and 1.0 are not changed. The saturated result is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- (s[0] < -1.0) ? -1.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < -1.0) ? -1.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
WriteMatrix( PAIRWORD, vd, d );
```

```

pAddr    <- AddressTranslation(vAddr, DATA, STORE);
offset    <- pAddr[3:2];
dataword  <- LoadMemory( QUADWORD, pAddr, vAddr, DATA );
d         <- ReadMatrix( QUADWORD, vt );
switch( offset )
{
  case 0 : dataword[0] <- d[3]; break;
  case 1 : dataword[1] <- d[3];
           dataword[0] <- d[2]; break;
  case 2 : dataword[2] <- d[3];
           dataword[1] <- d[2];
           dataword[0] <- d[1]; break;
  case 3 : dataword[3] <- d[3];
           dataword[2] <- d[2];
           dataword[1] <- d[1];
           dataword[0] <- d[0]; break;
}
StoreMemory( QUADWORD, dataword, pAddr, vAddr, DATA);

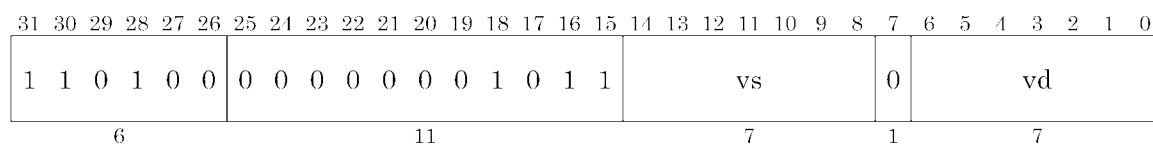
```

### Exceptions:

Address Error exception

## vsat1.t

Saturate to [-1.0:1.0] Triple Word



VFPU

### Syntax:

```
vsat1.t vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

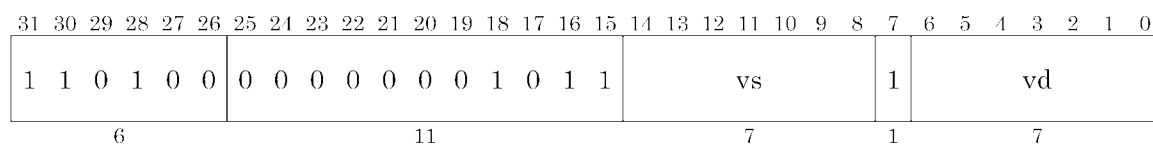
If any of the floating-point values of three elements from the matrix registers indicated by vs is less than -1.0, it is saturated to -1.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between -1.0 and 1.0 are not changed. The saturated result is stored as a three-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- (s[0] < -1.0) ? -1.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < -1.0) ? -1.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
d[2] <- (s[2] < -1.0) ? -1.0 : ( (s[2] > 1.0) ? 1.0 : s[2]);
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsat1.q

Saturate to [-1.0:1.0] Quad Word



VFPU

### Syntax:

```
vsat1.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Use prohibited

### Description:

If any of the floating-point values of four elements from the matrix registers indicated by vs is less than -1.0, it is saturated to -1.0. If any of the values is greater than 1.0, it is saturated to 1.0. Values between -1.0 and 1.0 are not changed. The saturated result is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

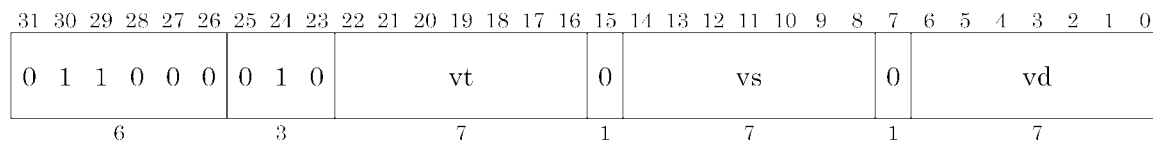
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- (s[0] < -1.0) ? -1.0 : ( (s[0] > 1.0) ? 1.0 : s[0]);
d[1] <- (s[1] < -1.0) ? -1.0 : ( (s[1] > 1.0) ? 1.0 : s[1]);
d[2] <- (s[2] < -1.0) ? -1.0 : ( (s[2] > 1.0) ? 1.0 : s[2]);
d[3] <- (s[3] < -1.0) ? -1.0 : ( (s[3] > 1.0) ? 1.0 : s[3]);
WriteMatrix( QUADWORD, vd, d );
```



## vsbn.s

ScaleBN Single Word



VFPU

### Syntax:

vsbn.s vd, vs, vt

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

The scaleBN calculation (change the binary logarithmic scale) is performed on the floating-point value of one element from the matrix register indicated by vs using the integer value of one element of the matrix register indicated by vt. The one-element floating-point result is stored at the location in the matrix register file indicated by vd. The input range of vt using two's complement representation is given by the following expression.

$$-127 \leq x \leq 127$$

Special solutions are as follows.

$$\text{scaleBN}(\text{nan}, t) = \text{nan}$$

$$\text{scaleBN}(+\text{inf}, t) = +\text{inf}$$

$$\text{scaleBN}(-\text{inf}, t) = -\text{inf}$$

$$\text{scaleBN}(+0.0, t) = +0.0$$

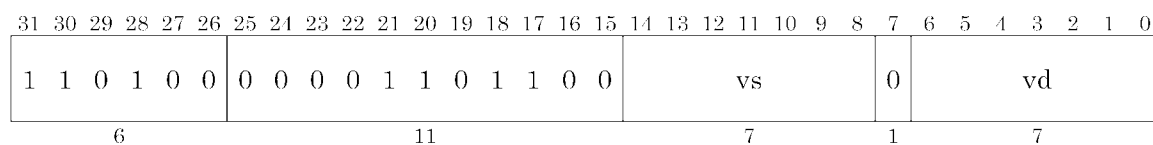
$$\text{scaleBN}(-0.0, t) = -0.0$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- scaleBN( s[0], t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vsbz.s

ScaleBZ Single Word



VFPU

### Syntax:

```
vsbz.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The remainder from the [1.0 ... 2.0] interval of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

scaleBZ is defined as follows:  $x = \text{scaleBZ}(x) * 2^{\log B(x)}$ ;  $1 \leq \text{scaleBZ}(x) < 2$ .

Special solutions are as follows.

$\text{scaleBZ}(\text{nan}) = \text{nan}$

$\text{scaleBZ}(+\text{inf}) = +1.0$

$\text{scaleBZ}(-\text{inf}) = -1.0$

$\text{scaleBZ}(+0.0) = +0.0$

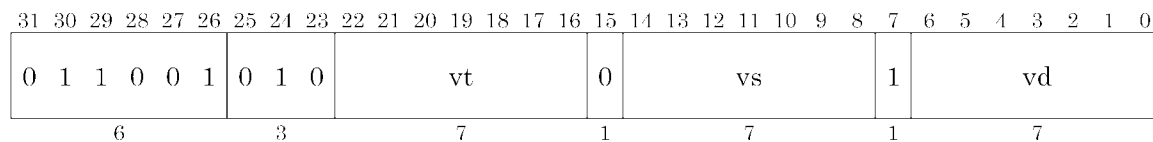
$\text{scaleBZ}(-0.0) = -0.0$

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- scaleBZ( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vscl.p

Scale Pair Word



VFPU

### Syntax:

```
vscl.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

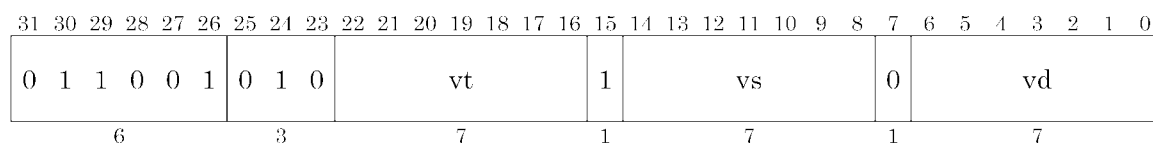
Two elements from the matrix registers indicated by vs are each multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
WriteMatrix( PAIRWORD, vd, d );
```

## vscl.t

Scale Triple Word



VFPU

### Syntax:

```
vscl.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

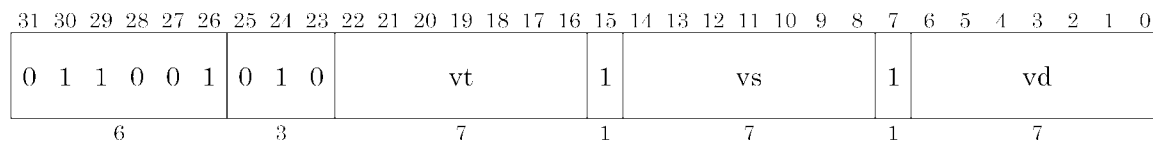
Three elements from the matrix registers indicated by vs are each multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
d[2] <- s[2] * t[0];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vscl.q

Scale Quad Word



VFPU

### Syntax:

```
vscl.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

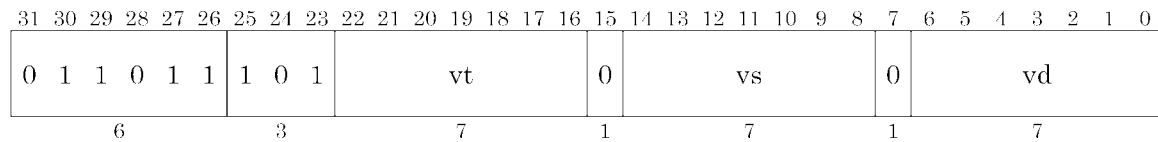
Four elements from the matrix registers indicated by vs are each multiplied by one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] * t[0];
d[1] <- s[1] * t[0];
d[2] <- s[2] * t[0];
d[3] <- s[3] * t[0];
WriteMatrix( QUADWORD, vd, d );
```

## vscmp.s

Set Compare Single Word



VFPU

### Syntax:

```
vscmp.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

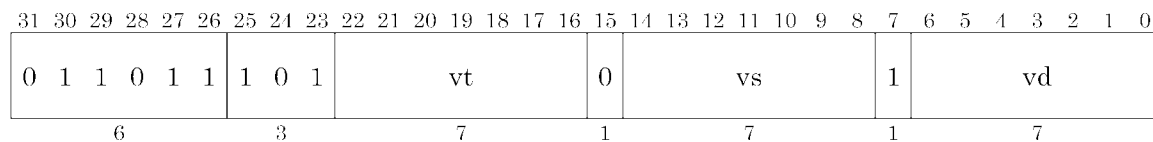
One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt, as floating-point numbers. If vs is equal to vt, the result is set to 0.0. If vs is greater than vt, the result is set to 1.0. If vs is less than vt, the result is set to -1.0. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- sign( s[0] - t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vscmp.p

Set Compare Pair Word



VFPU

### Syntax:

```
vscmp.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Two elements from the matrix registers indicated by vs are compared with two elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is equal to vt, the result is set to 0.0. If vs is greater than vt, the result is set to 1.0. If vs is less than vt, the result is set to -1.0. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

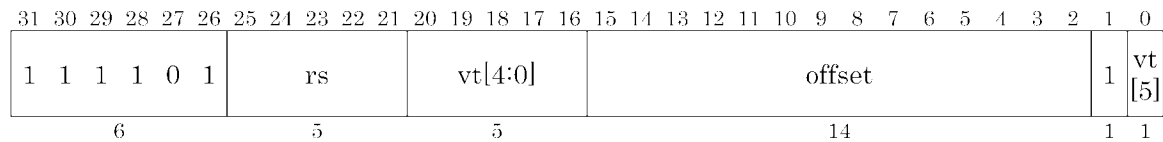
### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- sign( s[0] - t[0] );
d[1] <- sign( s[1] - t[1] );
WriteMatrix( PAIRWORD, vd, d );
```



## svr.q

### Store Quad Word Right from VFPU



VFPU

#### Syntax:

```
svr.q vt, offset(rs)
```

#### Instruction Type

CPU interlock instruction

#### Processing Time:

latency : 0      pitch : 7

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

#### Description:

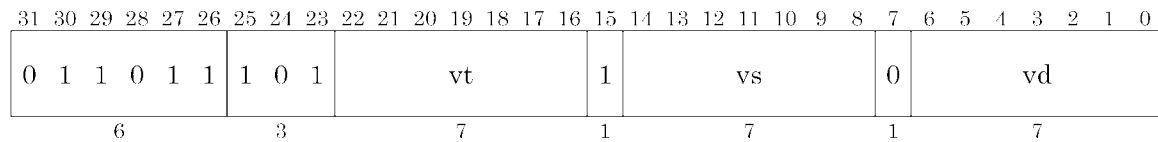
A virtual address is generated by ignoring the two low-order bits of the 16-bit offset, sign-extending the remaining 14 bits, and adding the result to the contents of CPU general-purpose register rs. From one to four words are read from locations in the matrix register file indicated by vt, then stored to memory such that the low-order word from the matrix register file is stored at the effective address, and the high-order word is stored at the quadword boundary. The words are stored in memory starting with the rightmost word within the quadword from the matrix register file. Any remaining words to the left in the quadword are not stored in memory and are unaffected by this instruction. If the effective address is not word aligned, the CPU generates an address error exception.

#### Operation:

```
vAddr  <- sign_extend({offset[15:2],2'b0}) + GPR[rs];
pAddr  <- AddressTranslation(vAddr, DATA, STORE);
offset <- pAddr[3:2];
```

## vscmp.t

### Set Compare Triple Word



VFPU

#### Syntax:

```
vscmp.t vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

#### Description:

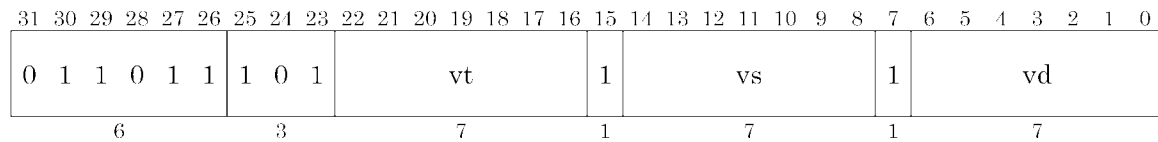
Three elements from the matrix registers indicated by vs are compared with three elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is equal to vt, the result is set to 0.0. If vs is greater than vt, the result is set to 1.0. If vs is less than vt, the result is set to -1.0. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- sign( s[0] - t[0] );
d[1] <- sign( s[1] - t[1] );
d[2] <- sign( s[2] - t[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vscmp.q

### Set Compare Quad Word



VFPU

#### Syntax:

```
vscmp.q vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

#### Description:

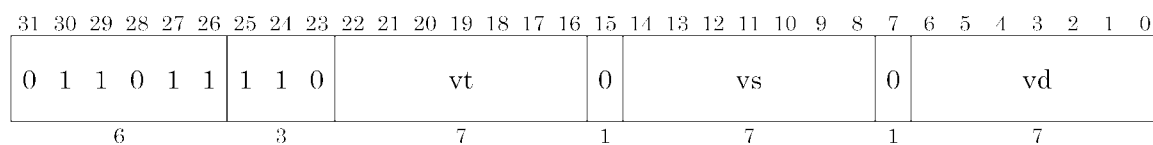
Four elements from the matrix registers indicated by vs are compared with four elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is equal to vt, the result is set to 0.0. If vs is greater than vt, the result is set to 1.0. If vs is less than vt, the result is set to -1.0. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- sign( s[0] - t[0] );
d[1] <- sign( s[1] - t[1] );
d[2] <- sign( s[2] - t[2] );
d[3] <- sign( s[3] - t[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsge.s

### Greater Equal Single Word to Value



VFPU

#### Syntax:

```
vsge.s vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

#### Description:

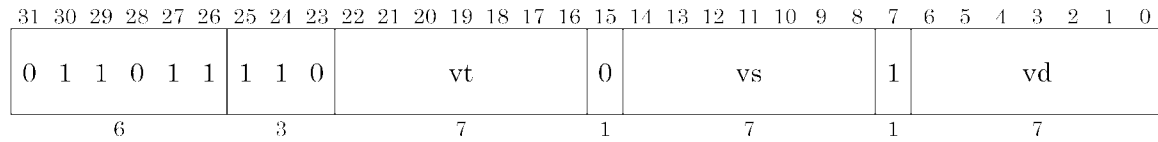
One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt, as floating-point numbers. If vs is greater than or equal to vt, the result is set to 1.0. If vs is less than vt, the result is set to 0.0. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- ( s[0] >= t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vsge.p

Greater Equal Pair Word to Value



VFPU

### Syntax:

```
vsge.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

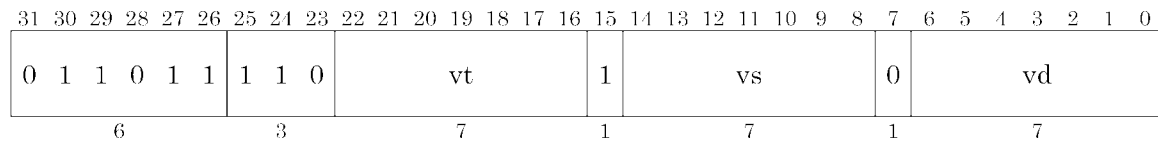
Two elements from the matrix registers indicated by vs are compared with two elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is greater than or equal to vt, the result is set to 1.0. If vs is less than vt, the result is set to 0.0. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- ( s[0] >= t[0] );
d[1] <- ( s[1] >= t[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vsge.t

Greater Equal Triple Word to Value



VFPU

### Syntax:

```
vsge.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

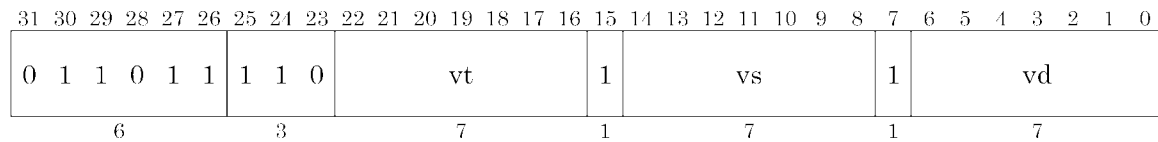
Three elements from the matrix registers indicated by vs are compared with three elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is greater than or equal to vt, the result is set to 1.0. If vs is less than vt, the result is set to 0.0. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- ( s[0] >= t[0] );
d[1] <- ( s[1] >= t[1] );
d[2] <- ( s[2] >= t[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsge.q

### Greater Equal Quad Word to Value



VFPU

#### Syntax:

```
vsge.q vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

#### Description:

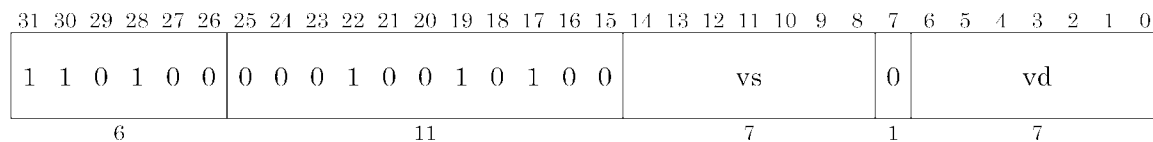
Four elements from the matrix registers indicated by vs are compared with four elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is greater than or equal to vt, the result is set to 1.0. If vs is less than vt, the result is set to 0.0. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- ( s[0] >= t[0] );
d[1] <- ( s[1] >= t[1] );
d[2] <- ( s[2] >= t[2] );
d[3] <- ( s[3] >= t[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsgn.s

Sign Single Word



VFPU

### Syntax:

```
vsgn.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

One element from the matrix register indicated by vs is examined. If the floating-point value of the element is less than 0.0, equal to 0.0, or greater than 0.0, the result is set to -1.0, 0.0, or 1.0, respectively. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

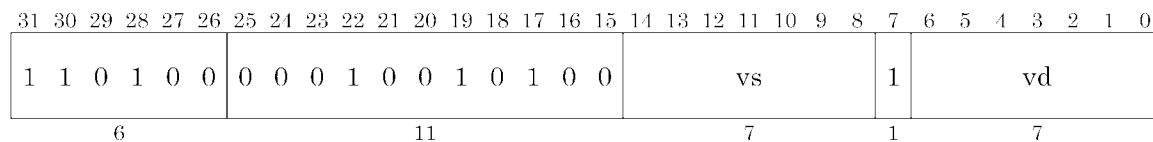
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- sign( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```



## vsgn.p

Sign Pair Word



VFPU

### Syntax:

```
vsgn.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

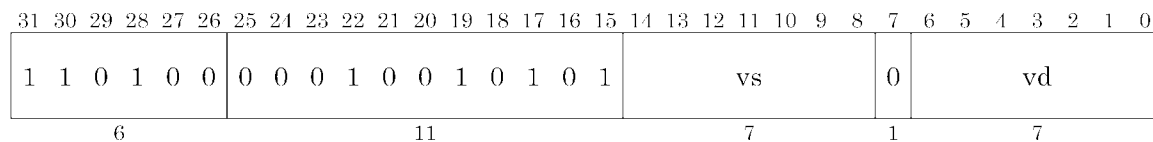
Two elements from the matrix registers indicated by vs are examined. If the floating-point value of each of the elements is less than 0.0, equal to 0.0, or greater than 0.0, the result is set to -1.0, 0.0, or 1.0, respectively, in the corresponding element. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- sign( s[0] );
d[1] <- sign( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vsgn.t

Sign Triple Word



VFPU

### Syntax:

```
vsgn.t vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

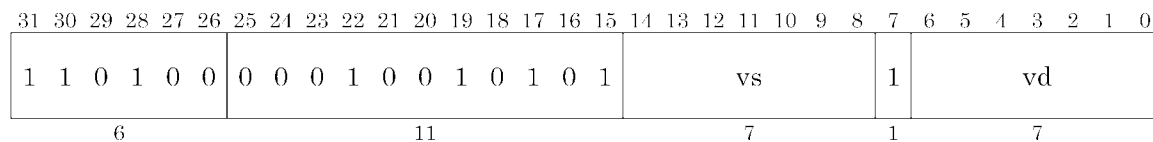
Three elements from the matrix registers indicated by vs are examined. If the floating-point value of each of the elements is less than 0.0, equal to 0.0, or greater than 0.0, the result is set to -1.0, 0.0, or 1.0, respectively, in the corresponding element. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- sign( s[0] );
d[1] <- sign( s[1] );
d[2] <- sign( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsgn.q

Sign Quad Word



VFPU

### Syntax:

```
vsgn.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Four elements from the matrix registers indicated by vs are examined. If the floating-point value of each of the elements is less than 0.0, equal to 0.0, or greater than 0.0, the result is set to -1.0, 0.0, or 1.0, respectively, in the corresponding element. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- sign( s[0] );
d[1] <- sign( s[1] );
d[2] <- sign( s[2] );
d[3] <- sign( s[3] );
WriteMatrix( QUADWORD, vd, d );
```

```

dataword <- LoadMemory( QUADWORD, pAddr, vAddr, DATA );
d      <- ReadMatrix( QUADWORD, vt );
switch( offset )
{
    case 0 : dataword[3] <- d[3];
             dataword[2] <- d[2];
             dataword[1] <- d[1];
             dataword[0] <- d[0]; break;
    case 1 : dataword[3] <- d[2];
             dataword[2] <- d[1];
             dataword[1] <- d[0]; break;
    case 2 : dataword[3] <- d[1];
             dataword[2] <- d[0]; break;
    case 3 : dataword[3] <- d[0]; break;
}
StoreMemory( QUADWORD, dataword, pAddr, vAddr, DATA);

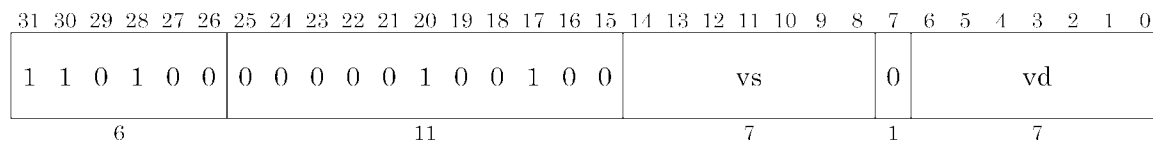
```

### Exceptions:

Address Error exception

## vsin.s

Sine Single Word



VFPU

### Syntax:

`vsin.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The sine of the floating-point value of one element from the matrix register indicated by `vs` is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_sin}(x) - \sin(x) | < 2^{-20} ; -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_sin}(\text{nan} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(-\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+0.0 * M\_PI\_2) = +0.0$$

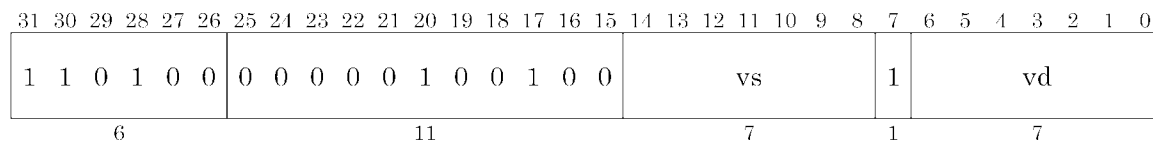
$$\text{approx\_sin}(-0.0 * M\_PI\_2) = -0.0$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_sin( M_PI_2 * s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

# vsin.p

Sine Pair Word



VFPU

## Syntax:

vsin.p vd, vs

## Instruction Type

Repeat (pipeline) instruction

## Processing Time:

latency : 8      pitch : 2

## Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

## Description:

The sines of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_sin}(x) - \sin(x) | < 2^{-20} ; -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_sin}(\text{nan} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(-\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+0.0 * M\_PI\_2) = +0.0$$

$$\text{approx\_sin}(-0.0 * M\_PI\_2) = -0.0$$

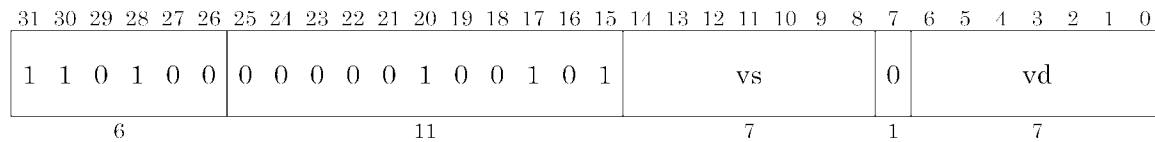
**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_sin( M_PI_2 * s[0] );
d[1] <- approx_sin( M_PI_2 * s[1] );
WriteMatrix( PAIRWORD, vd, d );
```



## vsin.t

Sine Triple Word



VFPU

### Syntax:

`vsin.t vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The sines of the floating-point values of three elements from the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_sin}(x) - \sin(x) | < 2^{-20} ; -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_sin}(\text{nan} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(-\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+0.0 * M\_PI\_2) = +0.0$$

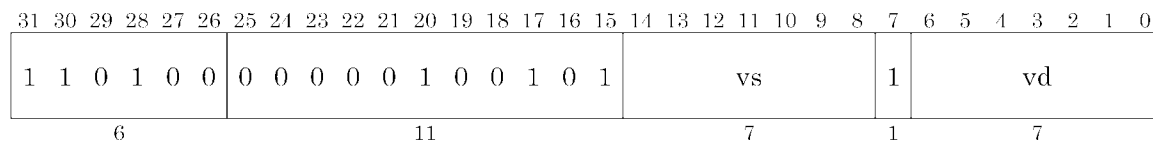
$$\text{approx\_sin}(-0.0 * M\_PI\_2) = -0.0$$

**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_sin( M_PI_2 * s[0] );
d[1] <- approx_sin( M_PI_2 * s[1] );
d[2] <- approx_sin( M_PI_2 * s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsin.q

Sine Quad Word



VFPU

### Syntax:

`vsin.q vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The sines of the floating-point values of four elements from the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_sin}(x) - \sin(x) | < 2^{-20} ; -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_sin}(\text{nan} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(-\text{inf} * M\_PI\_2) = \text{nan}$$

$$\text{approx\_sin}(+0.0 * M\_PI\_2) = +0.0$$

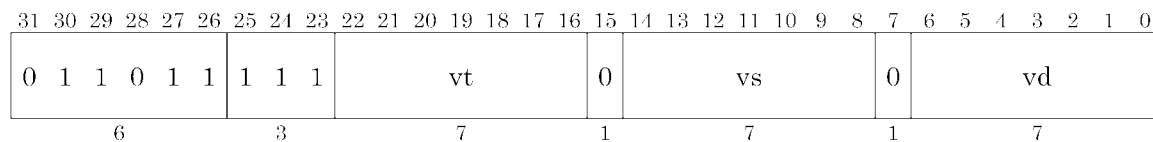
$$\text{approx\_sin}(-0.0 * M\_PI\_2) = -0.0$$

**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_sin( M_PI_2 * s[0] );
d[1] <- approx_sin( M_PI_2 * s[1] );
d[2] <- approx_sin( M_PI_2 * s[2] );
d[3] <- approx_sin( M_PI_2 * s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vslt.s

Less Than Single Word to Value



VFPU

### Syntax:

```
vslt.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

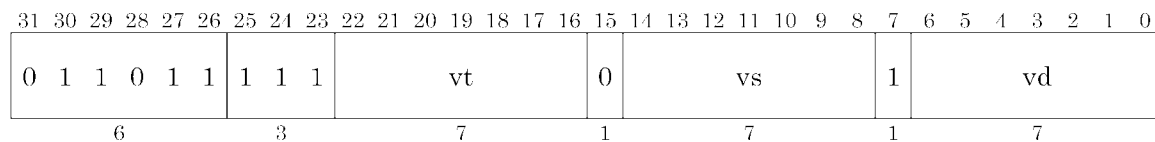
One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt, as floating-point numbers. If vs is less than vt, the result is set to 1.0. If vs is greater than or equal to vt, the result is set to 0.0. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- sign( s[0] < t[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vslt.p

Less Than Pair Word to Value



VFPU

### Syntax:

```
vslt.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

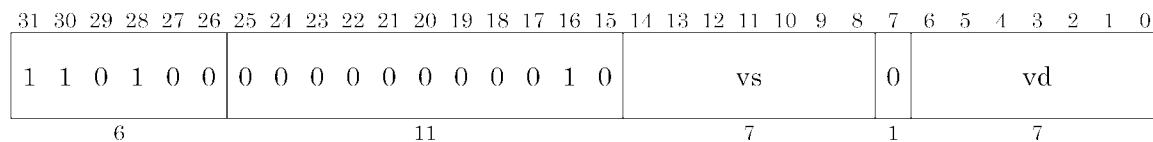
Two elements from the matrix registers indicated by vs are compared with two elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is less than vt, the result is set to 1.0. If vs is greater than or equal to vt, the result is set to 0.0. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- sign( s[0] < t[0] );
d[1] <- sign( s[1] < t[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vabs.s

Absolute Value Single Word



VFPU

### Syntax:

```
vabs.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

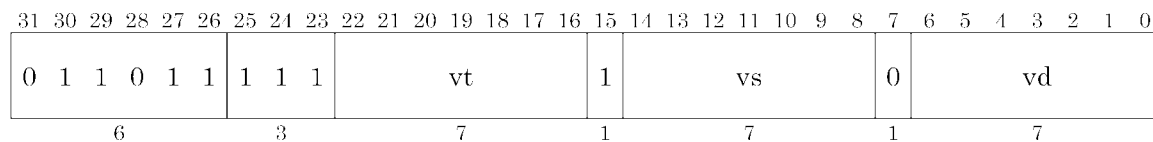
The absolute value of the floating-point value of one element from the matrix register indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- abs( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vslt.t

Less Than Triple Word to Value



VFPU

### Syntax:

```
vslt.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

Three elements from the matrix registers indicated by vs are compared with three elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is less than vt, the result is set to 1.0. If vs is greater than or equal to vt, the result is set to 0.0. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

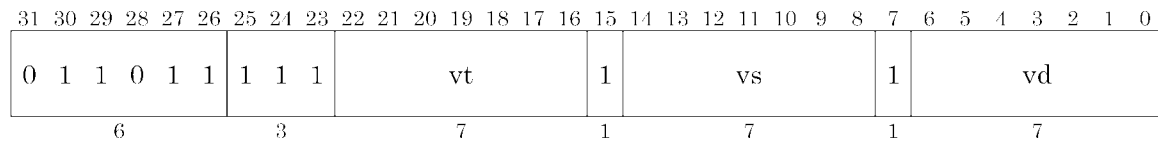
### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- sign( s[0] < t[0] );
d[1] <- sign( s[1] < t[1] );
d[2] <- sign( s[2] < t[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```



## vslt.q

Less Than Quad Word to Value



VFPU

### Syntax:

```
vslt.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

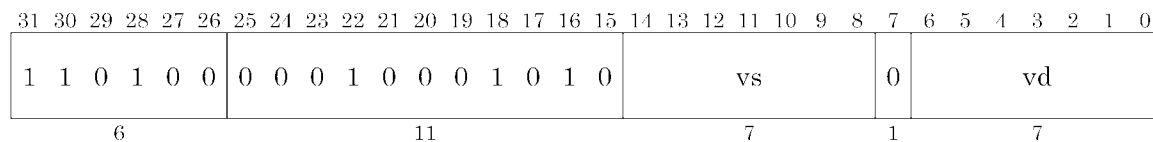
Four elements from the matrix registers indicated by vs are compared with four elements from the matrix registers indicated by vt, as floating-point numbers. For each pair of elements, if vs is less than vt, the result is set to 1.0. If vs is greater than or equal to vt, the result is set to 0.0. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- sign( s[0] < t[0] );
d[1] <- sign( s[1] < t[1] );
d[2] <- sign( s[2] < t[2] );
d[3] <- sign( s[3] < t[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsocp.s

Saturate and One's Complement Single Word



VFPU

### Syntax:

```
vsocp.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

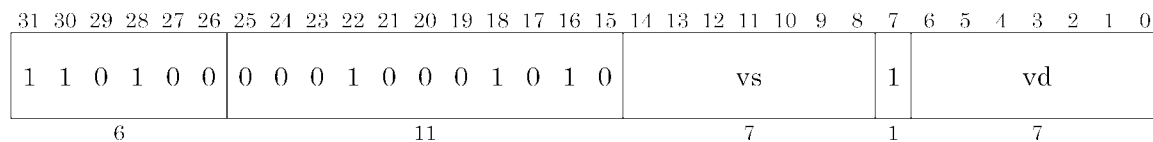
The one's complement of the floating-point value of one element from the matrix register indicated by vs is calculated. The original value of the element and its one's complement are each saturated to [0.0:1.0] and stored as a two-element floating-point result at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- 1.0 - s[0];
d[0] <- (d[0] < 0.0) ? 0.0 : ( (d[0] > 1.0) ? 1.0 : d[0]);
d[1] <- s[0];
d[1] <- (d[1] < 0.0) ? 0.0 : ( (d[1] > 1.0) ? 1.0 : d[1]);
WriteMatrix( PAIRWORD, vd, d );
```

## vsocp.p

Saturate and One's Complement Pair Word



VFPU

### Syntax:

```
vsocp.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

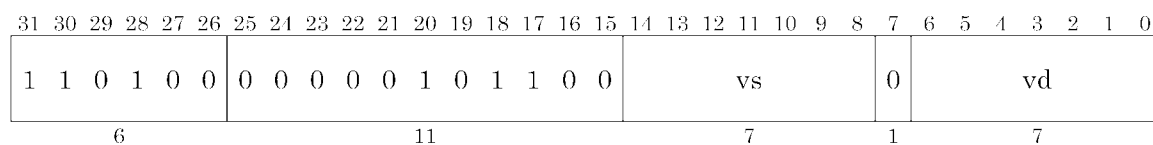
The one's complements of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The original values of the elements and their one's complements are each saturated to [0.0:1.0] and stored as a four-element floating-point result at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- 1.0 - s[0];
d[0] <- (d[0] < 0.0) ? 0.0 : ( (d[0] > 1.0) ? 1.0 : d[0]);
d[1] <- s[0];
d[1] <- (d[1] < 0.0) ? 0.0 : ( (d[1] > 1.0) ? 1.0 : d[1]);
d[2] <- 1.0 - s[1];
d[2] <- (d[2] < 0.0) ? 0.0 : ( (d[2] > 1.0) ? 1.0 : d[2]);
d[3] <- s[1];
d[3] <- (d[3] < 0.0) ? 0.0 : ( (d[3] > 1.0) ? 1.0 : d[3]);
WriteMatrix( QUADWORD, vd, d );
```

## vsqrt.s

Square Root Single Word



VFPU

### Syntax:

`vsqrt.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The square root of the floating-point value of one element of the matrix register indicated by *vs* is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by *vd*.

The precision of the calculation is given by the following expression.

$$|(\text{approx\_sqrt}(x) - \text{sqrt}(x)) / \text{sqrt}(x)| < 2^{-20};$$

Special solutions are as follows.

$$\text{approx\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_sqrt}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_sqrt}(+0.0) = +0.0$$

$$\text{approx\_sqrt}(-0.0) = +0.0$$

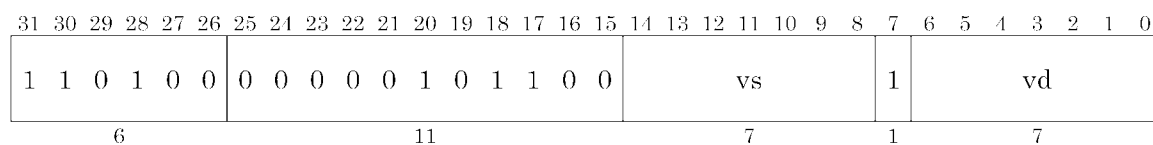
$$\text{approx\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_sqrt( s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vsqrt.p

Square Root Pair Word



VFPU

### Syntax:

`vsqrt.p vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The square roots of the floating-point values of two elements of the matrix registers indicated by `vs` are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$|(\text{approx\_sqrt}(x) - \text{sqrt}(x)) / \text{sqrt}(x)| < 2^{-20};$$

Special solutions are as follows.

$$\text{approx\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_sqrt}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_sqrt}(+0.0) = +0.0$$

$$\text{approx\_sqrt}(-0.0) = +0.0$$

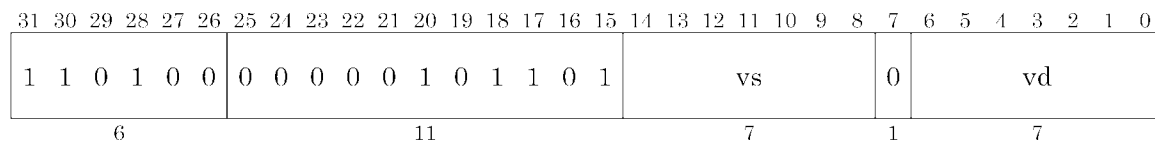
$$\text{approx\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_sqrt( s[0] );
d[1] <- approx_sqrt( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vsqrt.t

Square Root Triple Word



VFPU

### Syntax:

`vsqrt.t vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The square roots of the floating-point values of three elements of the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$|(\text{approx\_sqrt}(x) - \text{sqrt}(x)) / \text{sqrt}(x)| < 2^{-20};$$

Special solutions are as follows.

$$\text{approx\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_sqrt}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_sqrt}(+0.0) = +0.0$$

$$\text{approx\_sqrt}(-0.0) = +0.0$$

$$\text{approx\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

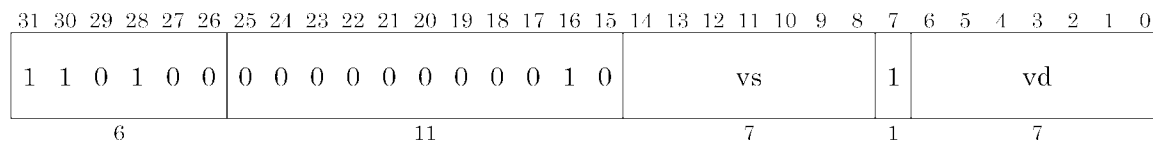


**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_sqrt( s[0] );
d[1] <- approx_sqrt( s[1] );
d[2] <- approx_sqrt( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vabs.p

Absolute Value Pair Word



VFPU

### Syntax:

vabs.p vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

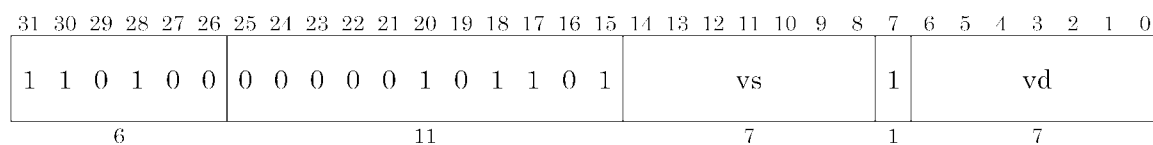
The absolute values of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- abs( s[0] );
d[1] <- abs( s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vsqrt.q

Square Root Quad Word



VFPU

### Syntax:

`vsqrt.q vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The square roots of the floating-point values of four elements of the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The precision of the calculation is given by the following expression.

$$|(\text{approx\_sqrt}(x) - \text{sqrt}(x)) / \text{sqrt}(x)| < 2^{-20};$$

Special solutions are as follows.

$$\text{approx\_sqrt}(\text{nan}) = \text{nan}$$

$$\text{approx\_sqrt}(+\text{inf}) = +\text{inf}$$

$$\text{approx\_sqrt}(-\text{inf}) = \text{nan}$$

$$\text{approx\_sqrt}(+0.0) = +0.0$$

$$\text{approx\_sqrt}(-0.0) = +0.0$$

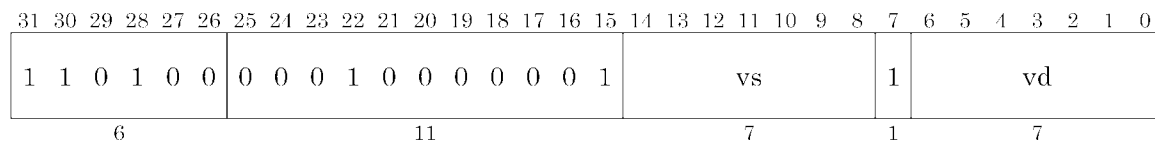
$$\text{approx\_sqrt}(x) = \text{nan}; -\text{inf} < x < -0.0$$

**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_sqrt( s[0] );
d[1] <- approx_sqrt( s[1] );
d[2] <- approx_sqrt( s[2] );
d[3] <- approx_sqrt( s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsrt1.q

Sort 1 Quad Word



VFPU

### Syntax:

```
vsrt1.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

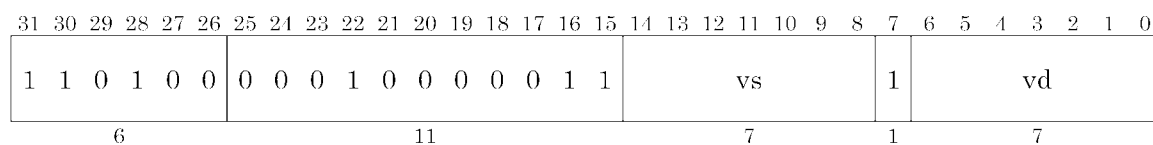
The floating-point values of four elements from the matrix registers indicated by vs are sorted. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- min( s[0] , s[1] );
d[1] <- max( s[0] , s[1] );
d[2] <- min( s[2] , s[3] );
d[3] <- max( s[2] , s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsrt2.q

Sort 2 Quad Word



VFPU

### Syntax:

```
vsrt2.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

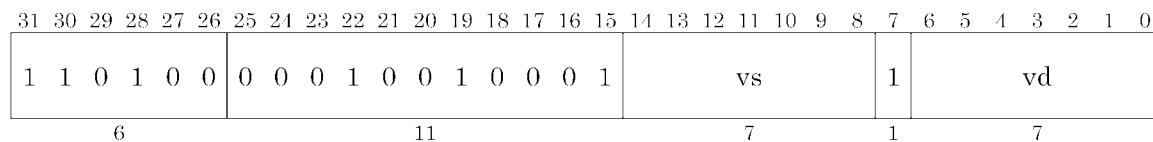
The floating-point values of four elements from the matrix registers indicated by vs are sorted. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- min( s[0] , s[3] );
d[1] <- min( s[1] , s[2] );
d[2] <- max( s[1] , s[2] );
d[3] <- max( s[0] , s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsrt3.q

Sort 3 Quad Word



VFPU

### Syntax:

```
vsrt3.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

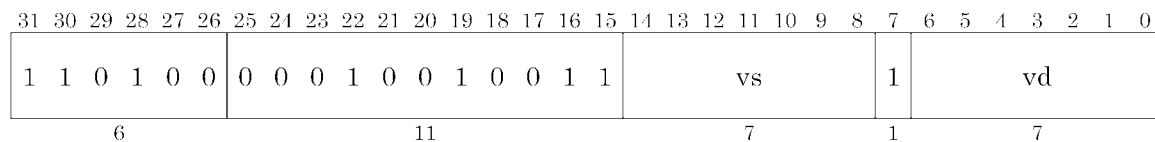
The floating-point values of four elements from the matrix registers indicated by vs are sorted. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- max( s[0] , s[1] );
d[1] <- min( s[0] , s[1] );
d[2] <- max( s[2] , s[3] );
d[3] <- min( s[2] , s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vsrt4.q

Sort 4 Quad Word



VFPU

### Syntax:

```
vsrt4.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

The floating-point values of four elements from the matrix registers indicated by vs are sorted. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

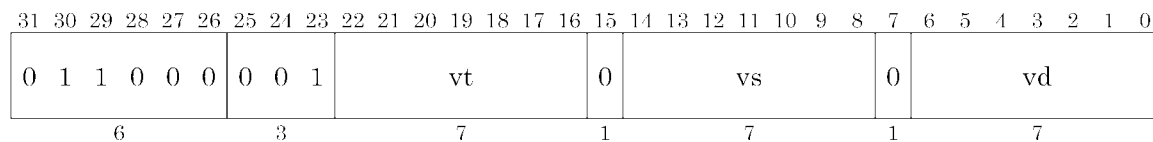
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- max( s[0] , s[3] );
d[1] <- max( s[1] , s[2] );
d[2] <- min( s[1] , s[2] );
d[3] <- min( s[0] , s[3] );
WriteMatrix( QUADWORD, vd, d );
```



## vsub.s

Subtract Single Word



VFPU

### Syntax:

```
vsub.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

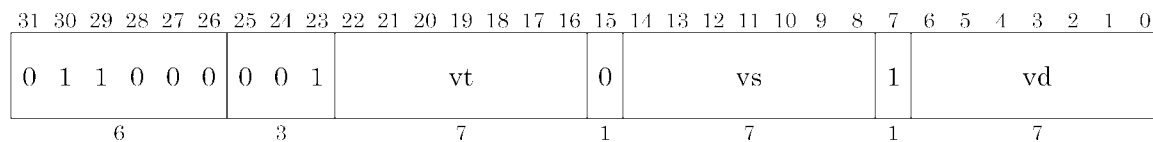
One element from the matrix register indicated by vt is subtracted from one element from the matrix register indicated by vs. The elements are treated as floating-point values. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] - t[0];
WriteMatrix( SINGLEWORD, vd, d );
```

## vsub.p

Subtract Pair Word



VFPU

### Syntax:

```
vsub.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

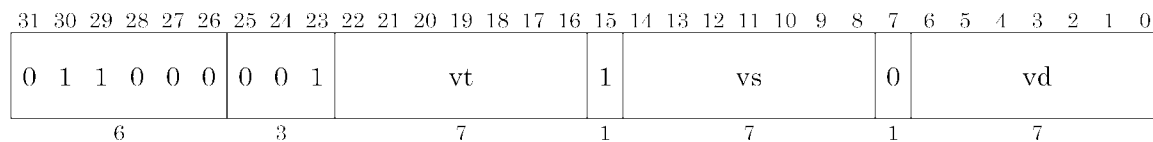
Two elements from the matrix registers indicated by vt are subtracted from two elements from the matrix registers indicated by vs. The elements are treated as floating-point values. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] - t[0];
d[1] <- s[1] - t[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vsub.t

Subtract Triple Word



VFPU

### Syntax:

```
vsub.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

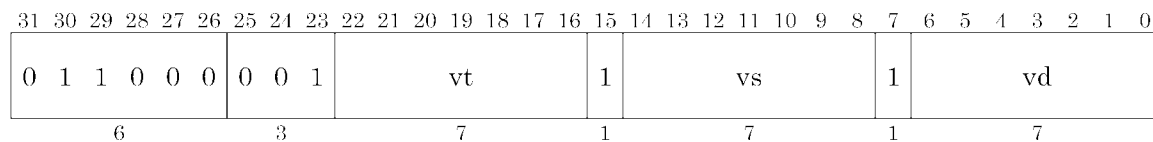
Three elements from the matrix registers indicated by vt are subtracted from three elements from the matrix registers indicated by vs. The elements are treated as floating-point values. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] - t[0];
d[1] <- s[1] - t[1];
d[2] <- s[2] - t[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsub.q

Subtract Quad Word



VFPU

### Syntax:

```
vsub.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

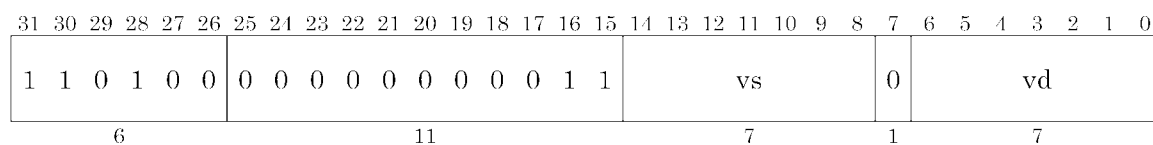
Four elements from the matrix registers indicated by vt are subtracted from four elements from the matrix registers indicated by vs. The elements are treated as floating-point values. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] - t[0];
d[1] <- s[1] - t[1];
d[2] <- s[2] - t[2];
d[3] <- s[3] - t[3];
WriteMatrix( QUADWORD, vd, d );
```

## vabs.t

Absolute Value Triple Word



VFPU

### Syntax:

vabs.t vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

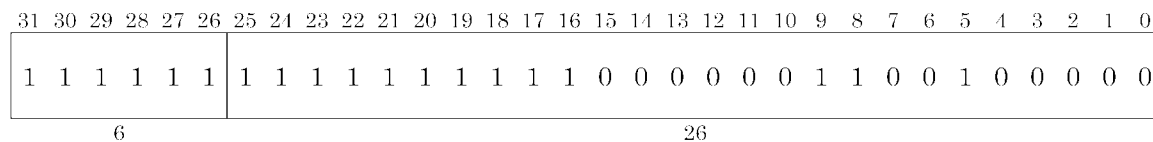
The absolute values of the floating-point values of three elements from the matrix registers indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- abs( s[0] );
d[1] <- abs( s[1] );
d[2] <- abs( s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vsync

Synchronize



VFPU

### Syntax:

`vsync`

### Instruction Type

Synchronization instruction

### Processing Time:

latency : 0      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	No effect

### Description:

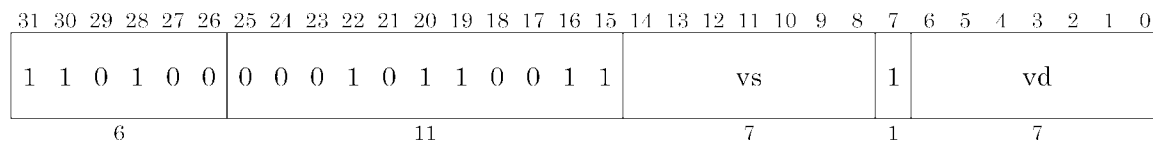
The VFPU's internal state is synchronized. Subsequent VFPU instructions stall until the pipeline has emptied. If the instruction following the vsync is not a VFPU instruction, a vnop should be inserted.

### Operation:

`Sync () ;`

## vt4444.q

Convert to color4444 from packed unsigned chars Quad Word



VFPU

### Syntax:

vt4444.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Use prohibited

### Description:

The packed, unsigned 8-bit data of four elements from the matrix registers indicated by vs is converted to packed 4444 color data and converted to 64 bits. This 64 bits is stored as two elements (32bits each) at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
tmp0[ 3: 0] <- (s[0]>> 4)&15;
tmp0[ 7: 4] <- (s[0]>>12)&15;
tmp0[11: 8] <- (s[0]>>20)&15;
tmp0[15:12] <- (s[0]>>28)&15;
tmp0[19:16] <- (s[1]>> 4)&15;
tmp0[23:20] <- (s[1]>>12)&15;
tmp0[27:24] <- (s[1]>>20)&15;
tmp0[31:28] <- (s[1]>>28)&15;
tmp1[ 3: 0] <- (s[2]>> 4)&15;
tmp1[ 7: 4] <- (s[2]>>12)&15;
tmp1[11: 8] <- (s[2]>>20)&15;
tmp1[15:12] <- (s[2]>>28)&15;
tmp1[19:16] <- (s[3]>> 4)&15;
```

```

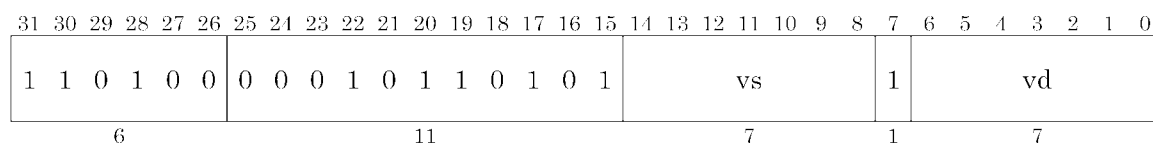
tmp1[23:20] <- (s[3]>>12)&15;
tmp1[27:24] <- (s[3]>>20)&15;
tmp1[31:28] <- (s[3]>>28)&15;
d[0] <- tmp0;
d[1] <- tmp1;
WriteMatrix( PAIRWORD, vd, d );

```



## vt5551.q

Convert to color5551 from packed unsigned chars Quad Word



VFPU

### Syntax:

vt5551.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Use prohibited

### Description:

The packed, unsigned 8-bit data of four elements from the matrix registers indicated by vs is converted to packed 5551 color data and converted to 64 bits. This 64 bits is stored as two elements (32bits each) at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
tmp0[ 4: 0] <- (s[0]>> 3)&31;
tmp0[ 9: 5] <- (s[0]>>11)&31;
tmp0[14:10] <- (s[0]>>19)&31;
tmp0[15]    <- (s[0]>>31)&1 ;
tmp0[20:16] <- (s[1]>> 3)&31;
tmp0[25:21] <- (s[1]>>11)&31;
tmp0[30:26] <- (s[1]>>19)&31;
tmp0[31]    <- (s[1]>>31)&1 ;
tmp1[ 4: 0] <- (s[2]>> 3)&31;
tmp1[ 9: 5] <- (s[2]>>11)&31;
tmp1[14:10] <- (s[2]>>19)&31;
tmp1[15]    <- (s[2]>>31)&1 ;
tmp1[20:16] <- (s[3]>> 3)&31;
```

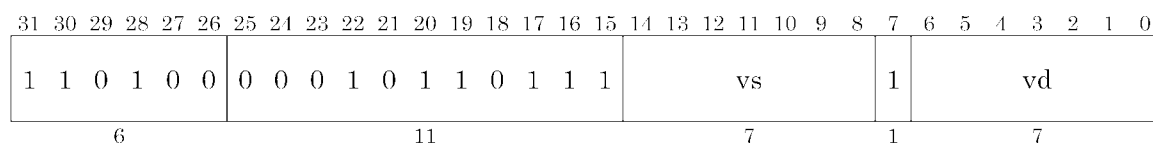
```

tmp1[25:21] <- (s[3]>>11)&31;
tmp1[30:26] <- (s[3]>>19)&31;
tmp1[31]    <- (s[3]>>31)&1 ;
d[0] <- tmp0;
d[1] <- tmp1;
WriteMatrix( PAIRWORD, vd, d );

```

## vt5650.q

Convert to color5650 from packed unsigned chars Quad Word



VFPU

### Syntax:

vt5650.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Use prohibited

### Description:

The packed, unsigned 8-bit data of four elements from the matrix registers indicated by vs is converted to packed 5650 color data and converted to 64 bits. This 64 bits is stored as two elements (32bits each) at locations in the matrix register file indicated by vd.

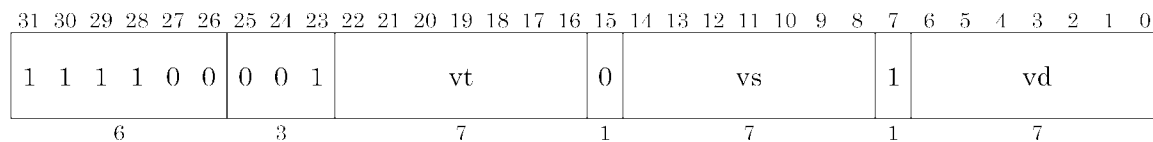
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
tmp0[ 4: 0] <- (s[0]>> 3)&31;
tmp0[10: 5] <- (s[0]>>10)&63;
tmp0[15:11] <- (s[0]>>19)&31;
tmp0[20:16] <- (s[1]>> 3)&31;
tmp0[26:21] <- (s[1]>>10)&63;
tmp0[31:27] <- (s[1]>>19)&31;
tmp1[ 4: 0] <- (s[2]>> 3)&31;
tmp1[10: 5] <- (s[2]>>10)&63;
tmp1[15:11] <- (s[2]>>19)&31;
tmp1[20:16] <- (s[3]>> 3)&31;
tmp1[26:21] <- (s[3]>>10)&63;
tmp1[31:27] <- (s[3]>>19)&31;
d[0] <- tmp0;
```

```
d[1] <- tmp1;
WriteMatrix( PAIRWORD, vd, d );
```

## vtfm2.p

Transform 2 Pair Word



VFPU

### Syntax:

```
vtfm2.p vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

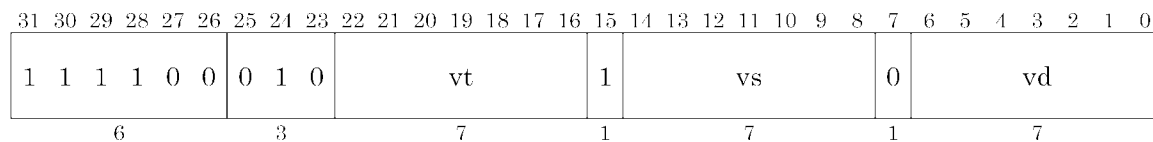
The transform of the elements of the 2x2 matrix from the matrix registers indicated by vs with the two elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point values. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRXPAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[0] + s[4] * t[1];
d[1] <- s[1] * t[0] + s[5] * t[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vtfm3.t

### Transform 3 Triple Word



VFPU

#### Syntax:

```
vtfm3.t  vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9                  pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

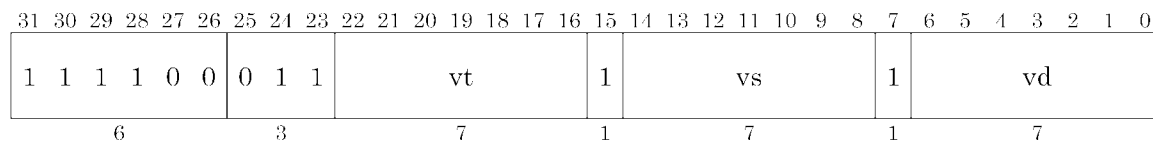
The transform of the elements of the 3x3 matrix from the matrix registers indicated by vs with the three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point values. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEXTRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] * t[0] + s[4] * t[1] + s[8] * t[2];
d[1] <- s[1] * t[0] + s[5] * t[1] + s[9] * t[2];
d[2] <- s[2] * t[0] + s[6] * t[1] + s[10] * t[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vtfm4.q

Transform 4 Quad Word



VFPU

### Syntax:

```
vtfm4.q vd, vs, vt
```

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

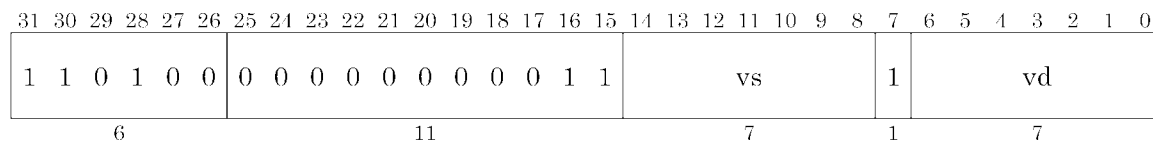
The transform of the elements of the 4x4 matrix from the matrix registers indicated by vs with the four elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point values. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADXQUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] * t[0] + s[4] * t[1] + s[8] * t[2] + s[12] * t[3];
d[1] <- s[1] * t[0] + s[5] * t[1] + s[9] * t[2] + s[13] * t[3];
d[2] <- s[2] * t[0] + s[6] * t[1] + s[10] * t[2] + s[14] * t[3];
d[3] <- s[3] * t[0] + s[7] * t[1] + s[11] * t[2] + s[15] * t[3];
WriteMatrix( QUADWORD, vd, d );
```

## vabs.q

Absolute Value Quad Word



VFPU

### Syntax:

vabs.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Only swizzle is valid	No effect	Valid

### Description:

The absolute values of the floating-point values of four elements from the matrix registers indicated by vs are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

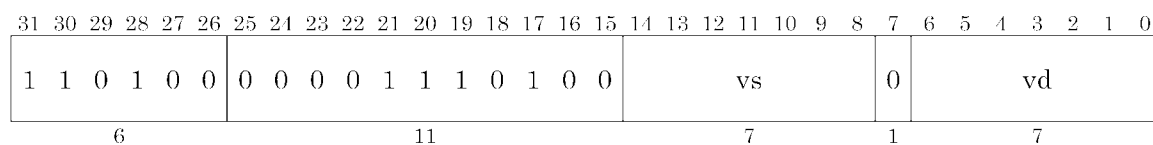
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- abs( s[0] );
d[1] <- abs( s[1] );
d[2] <- abs( s[2] );
d[3] <- abs( s[3] );
WriteMatrix( QUADWORD, vd, d );
```



## vus2i.s

Convert unsigned short to integer Single Word



VFPU

### Syntax:

```
vus2i.s vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Only write mask is valid

### Description:

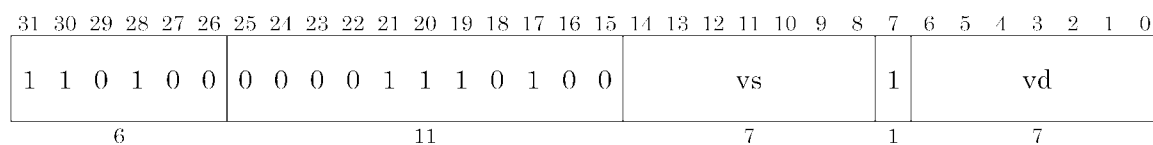
32-bit packed data from the matrix register indicated by vs is unpacked and converted from unsigned 16-bit integers to signed 32-bit integers. The two-element integer result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- {s[0][15: 0], 15'b0};
d[1] <- {s[0][31:16], 15'b0};
WriteMatrix( PAIRWORD, vd, d );
```

## vus2i.p

Convert unsigned short to integer Pair Word



VFPU

### Syntax:

```
vus2i.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Only write mask is valid

### Description:

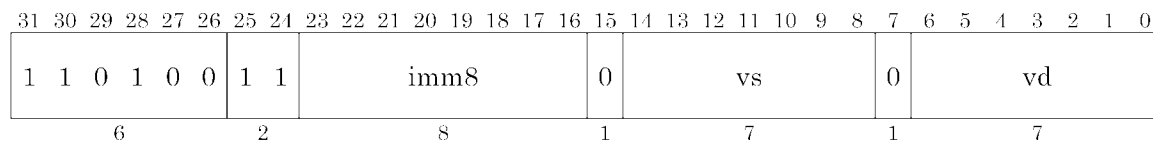
64-bit packed data from the matrix registers indicated by vs is unpacked and converted from unsigned 16-bit integers to signed 32-bit integers. The four-element integer result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- {s[0][15: 0], 15'b0};
d[1] <- {s[0][31:16], 15'b0};
d[2] <- {s[1][15: 0], 15'b0};
d[3] <- {s[1][31:16], 15'b0};
WriteMatrix( QUADWORD, vd, d );
```

## vwbn.s

WrapBN Single Word



VFPU

### Syntax:

```
vwbn.s vd, vs, imm8
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The modulus of the floating-point value of one element from the matrix register indicated by vs with the exponent indicated by the imm8 field is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

wrapBN is defined by the following expression.

$$\text{wrapBN}(s) = \text{fmod}(s, 2^{127-\text{imm8}}) + (s < 0.0 ? -2^{127-\text{imm8}} : 2^{127-\text{imm8}}).$$

The input range of imm8 is as follows.

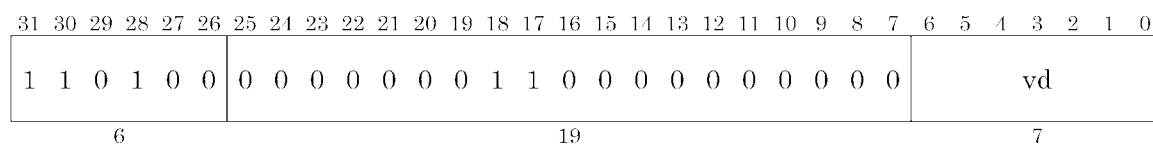
$$1 \leq \text{imm8} \leq 254$$

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- wrapBN( s[0], imm8 );
WriteMatrix( SINGLEWORD, vd, d );
```

## vzero.s

Set Zero Single Word



VFPU

### Syntax:

```
vzero.s vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

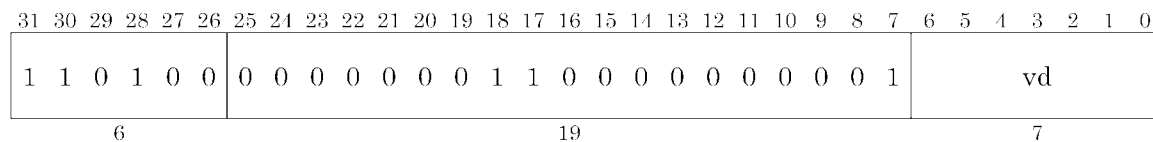
0.0 is stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 0.0;
WriteMatrix( SINGLEWORD, vd, d );
```

## vzero.p

Set Zero Pair Word



VFPU

### Syntax:

```
vzero.p vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

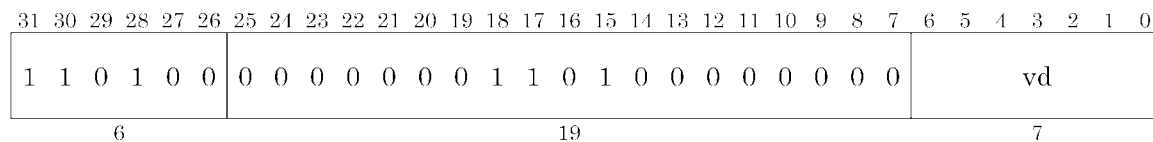
0.0 is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 0.0;
d[1] <- 0.0;
WriteMatrix( PAIRWORD, vd, d );
```

## vzero.t

Set Zero Triple Word



VFPU

### Syntax:

```
vzero.t vd
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

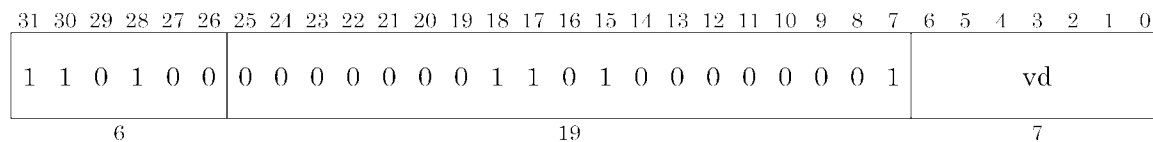
0.0 is stored as a three-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 0.0;
d[1] <- 0.0;
d[2] <- 0.0;
WriteMatrix( TRIPLEWORD, vd, d );
```

## vzero.q

Set Zero Quad Word



VFPU

### Syntax:

`vzero.q vd`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Valid

### Description:

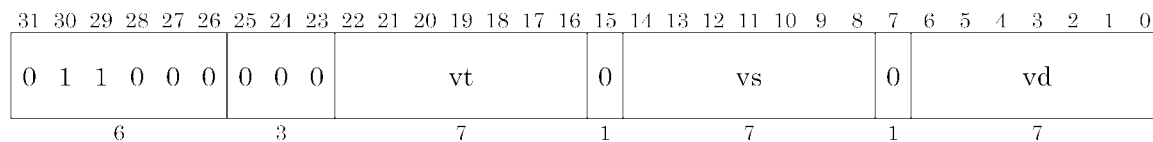
0.0 is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

### Operation:

```
d[0] <- 0.0;
d[1] <- 0.0;
d[2] <- 0.0;
d[3] <- 0.0;
WriteMatrix( QUADWORD, vd, d );
```

## vadd.s

Add Single Word



VFPU

### Syntax:

```
vadd.s vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

One element from the matrix register indicated by vs is added to one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

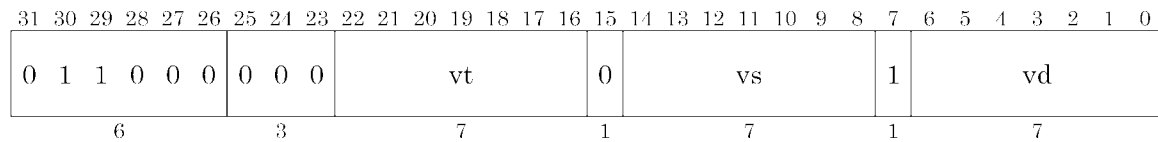
### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] + t[0];
WriteMatrix( SINGLEWORD, vd, d );
```



## vadd.p

Add Pair Word



VFPU

### Syntax:

```
vadd.p vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

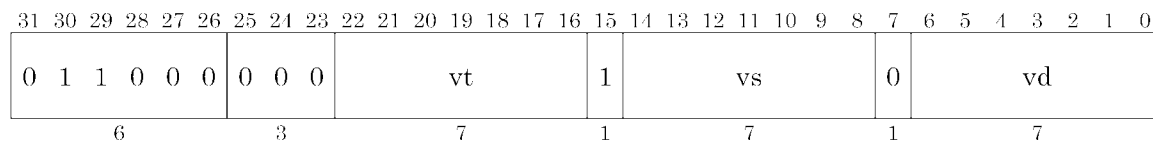
Two elements from the matrix registers indicated by vs are added to two elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] + t[0];
d[1] <- s[1] + t[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vadd.t

Add Triple Word



VFPU

### Syntax:

```
vadd.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

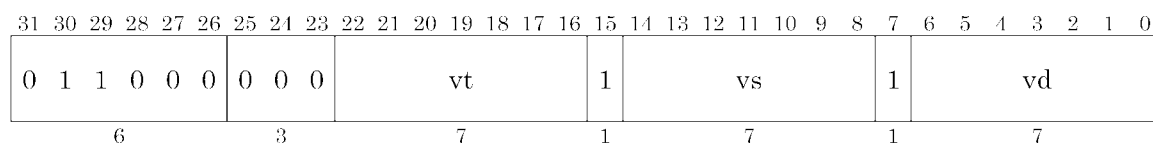
Three elements from the matrix registers indicated by vs are added to three elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] + t[0];
d[1] <- s[1] + t[1];
d[2] <- s[2] + t[2];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vadd.q

Add Quad Word



VFPU

### Syntax:

```
vadd.q vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

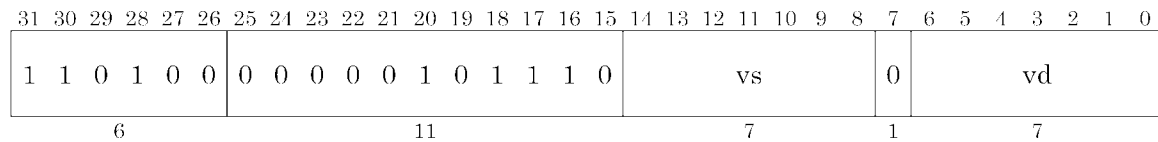
Four elements from the matrix registers indicated by vs are added to four elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
d[0] <- s[0] + t[0];
d[1] <- s[1] + t[1];
d[2] <- s[2] + t[2];
d[3] <- s[3] + t[3];
WriteMatrix( QUADWORD, vd, d );
```

## vasin.s

Arc Sine Single Word



VFPU

### Syntax:

`vasin.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The arcsine of the floating-point value of one element from the matrix register indicated by *vs* is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by *vd*.

The precision of the calculation is given by the following expression.

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-20} ; 0 \leq |x| \leq 0.9$$

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-6} ; \text{otherwise}$$

Special solutions are as follows.

$$\text{approx\_asin}(\text{nan}) = \text{nan}$$

$$\text{approx\_asin}(+\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(-\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(+0.0) = +0.0$$

$$\text{approx\_asin}(-0.0) = -0.0$$

$$\text{approx\_asin}(x) = \text{nan}; -\text{inf} < x < -1.0$$

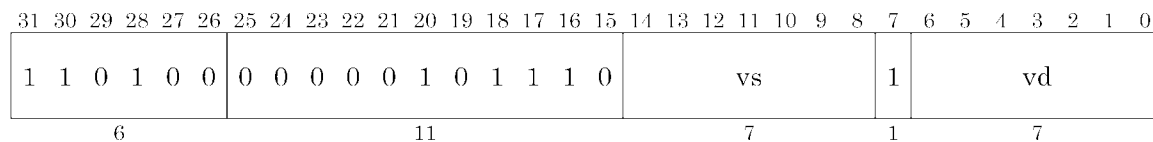
$$\text{approx\_asin}(x) = \text{nan}; +1.0 < x < +\text{inf}$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_asin( s[0] ) / M_PI_2;
WriteMatrix( SINGLEWORD, vd, d );
```

## vasin.p

Arc Sine Pair Word



VFPU

### Syntax:

vasin.p vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The arcsines of the floating-point values of two elements from the matrix registers indicated by vs are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-20} ; 0 \leq |x| \leq 0.9$$

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-6} ; \text{otherwise}$$

Special solutions are as follows.

$$\text{approx\_asin}(\text{nan}) = \text{nan}$$

$$\text{approx\_asin}(+\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(-\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(+0.0) = +0.0$$

$$\text{approx\_asin}(-0.0) = -0.0$$

$$\text{approx\_asin}(x) = \text{nan}; -\text{inf} < x < -1.0$$

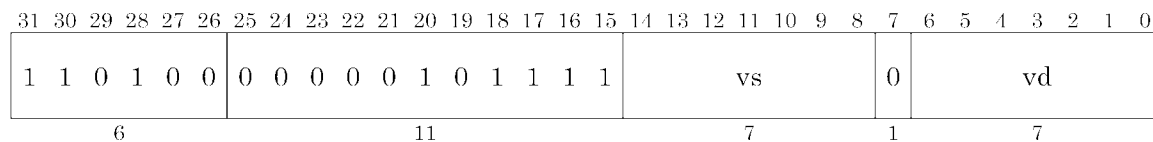
$$\text{approx\_asin}(x) = \text{nan}; +1.0 < x < +\text{inf}$$

**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_asin( s[0] ) / M_PI_2;
d[1] <- approx_asin( s[1] ) / M_PI_2;
WriteMatrix( PAIRWORD, vd, d );
```

## vasin.t

Arc Sine Triple Word



VFPU

### Syntax:

vasin.t vd, vs

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The arcsines of the floating-point values of three elements from the matrix registers indicated by vs are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

The precision of the calculation is given by the following expression.

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-20} ; 0 \leq |x| \leq 0.9$$

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-6} ; \text{otherwise}$$

Special solutions are as follows.

$$\text{approx\_asin}(\text{nan}) = \text{nan}$$

$$\text{approx\_asin}(+\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(-\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(+0.0) = +0.0$$

$$\text{approx\_asin}(-0.0) = -0.0$$

$$\text{approx\_asin}(x) = \text{nan}; -\text{inf} < x < -1.0$$

$$\text{approx\_asin}(x) = \text{nan}; +1.0 < x < +\text{inf}$$

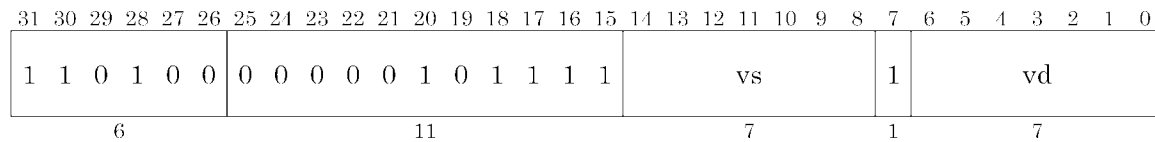


**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_asin( s[0] ) / M_PI_2;
d[1] <- approx_asin( s[1] ) / M_PI_2;
d[2] <- approx_asin( s[2] ) / M_PI_2;
WriteMatrix( TRIPLEWORD, vd, d );
```

## vasin.q

Arc Sine Quad Word



VFPU

### Syntax:

`vasin.q vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The arcsines of the floating-point values of four elements from the matrix registers indicated by *vs* are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by *vd*.

The precision of the calculation is given by the following expression.

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-20} ; 0 \leq |x| \leq 0.9$$

$$| \text{approx\_asin}(x) - \text{asin}(x) | < 2^{-6} ; \text{otherwise}$$

Special solutions are as follows.

$$\text{approx\_asin}(\text{nan}) = \text{nan}$$

$$\text{approx\_asin}(+\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(-\text{inf}) = \text{nan}$$

$$\text{approx\_asin}(+0.0) = +0.0$$

$$\text{approx\_asin}(-0.0) = -0.0$$

$$\text{approx\_asin}(x) = \text{nan}; -\text{inf} < x < -1.0$$

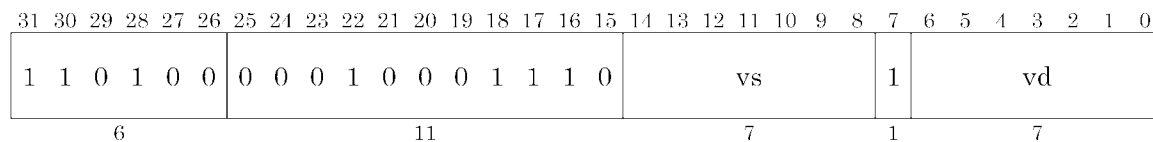
$$\text{approx\_asin}(x) = \text{nan}; +1.0 < x < +\text{inf}$$

# **Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_asin( s[0] ) / M_PI_2;
d[1] <- approx_asin( s[1] ) / M_PI_2;
d[2] <- approx_asin( s[2] ) / M_PI_2;
d[3] <- approx_asin( s[3] ) / M_PI_2;
WriteMatrix( QUADWORD, vd, d );
```

## vavg.p

Average Pair Word



VFPU

### Syntax:

vavg.p vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

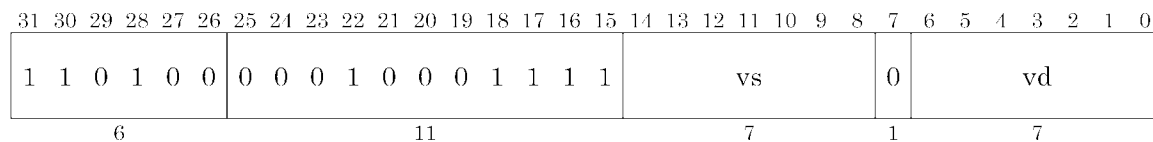
The average of the floating-point values of two elements from the matrix registers indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- s[0]/2;
d[0] <- d[0] + s[1]/2;
WriteMatrix( SINGLEWORD, vd, d );
```

## vavg.t

Average Triple Word



VFPU

### Syntax:

vavg.t vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

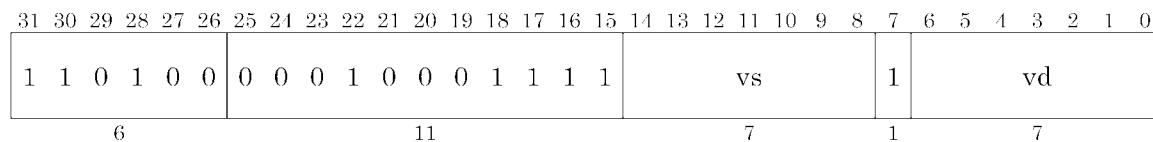
The average of the floating-point values of three elements from the matrix registers indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- s[0]/3;
d[0] <- d[0] + s[1]/3;
d[0] <- d[0] + s[2]/3;
WriteMatrix( SINGLEWORD, vd, d );
```

## vavg.q

Average Quad Word



VFPU

### Syntax:

vavg.q vd, vs

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

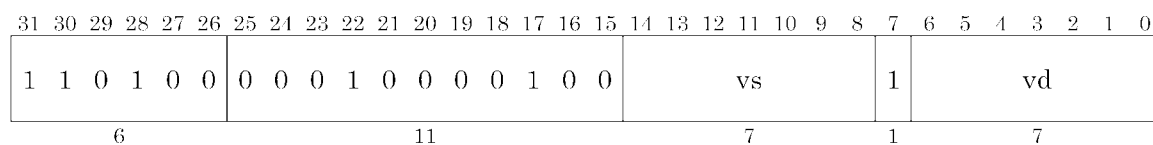
The average of the floating-point values of four elements from the matrix registers indicated by vs is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- s[0]/4;
d[0] <- d[0] + s[1]/4;
d[0] <- d[0] + s[2]/4;
d[0] <- d[0] + s[3]/4;
WriteMatrix( SINGLEWORD, vd, d );
```

## vbfy1.p

Butterfly 1 Pair Word



VFPU

### Syntax:

```
vbfy1.p vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

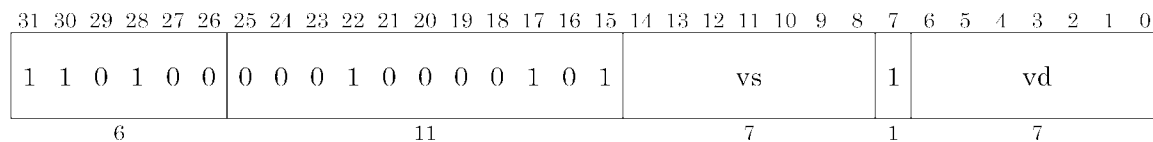
The butterfly of the floating-point values of two elements from the matrix registers indicated by vs is calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- s[0] + s[1];
d[1] <- s[0] - s[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vbfy1.q

Butterfly 1 Quad Word



VFPU

### Syntax:

```
vbfy1.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

The butterfly of the floating-point values of four elements from the matrix registers indicated by vs is calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

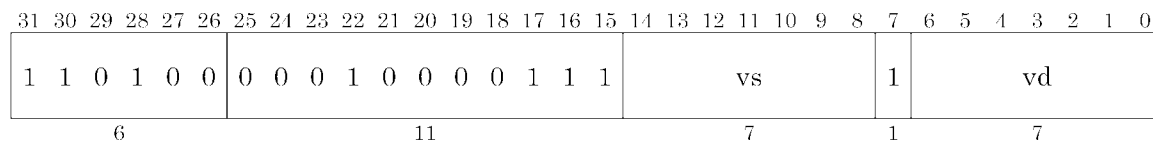
### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- s[0] + s[1];
d[1] <- s[0] - s[1];
d[2] <- s[2] + s[3];
d[3] <- s[2] - s[3];
WriteMatrix( QUADWORD, vd, d );
```



## vbfy2.q

Butterfly 2 Quad Word



VFPU

### Syntax:

```
vbfy2.q vd, vs
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

### Description:

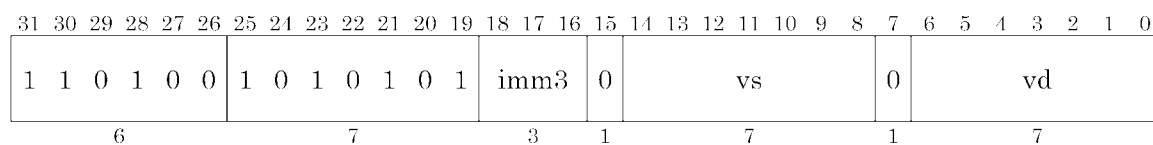
The butterfly of the floating-point values of four elements from the matrix registers indicated by vs is calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- s[0] + s[2];
d[1] <- s[1] + s[3];
d[2] <- s[0] - s[2];
d[3] <- s[1] - s[3];
WriteMatrix( QUADWORD, vd, d );
```

## vcmovf.s

Conditional Move on False Single Word



VFPU

### Syntax:

```
vcmovf.s vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is false (0), one floating-point element from the matrix register indicated by vs is copied to the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
if( imm3<6 )
begin
    if( !c[imm3] )
    begin
        d[0] <- s[0];
    end
end
else if( imm3==6 )
begin
    if( !c[0] )
```

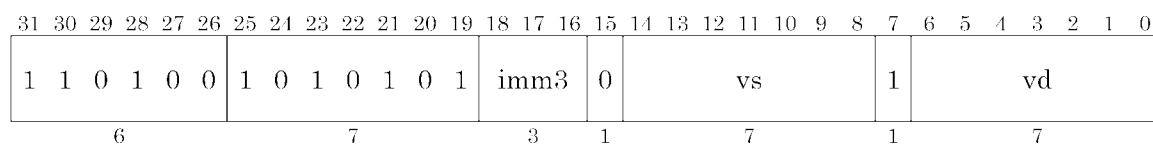
```

        d[0] <- s[0];
    end
    WriteMatrix( SINGLEWORD, vd, d );

```

## vcmovf.p

Conditional Move on False Pair Word



VFPU

### Syntax:

```
vcmovf.p vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is false (0), two floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
if( imm3<6 )
begin
    if( !c[imm3] )
    begin
        d[0] <- s[0];
        d[1] <- s[1];
    end
end
else if( imm3==6 )
```

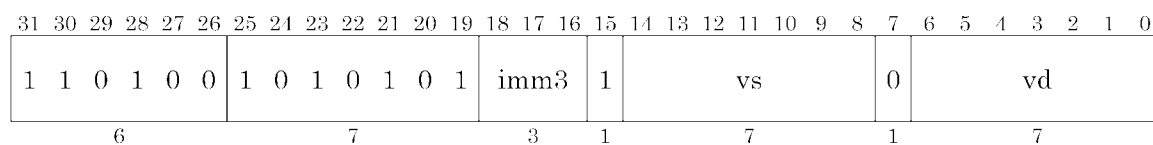
```

begin
  if( !c[0] )
    d[0] <- s[0];
  if( !c[1] )
    d[1] <- s[1];
end
WriteMatrix( PAIRWORD, vd, d );

```

## vcmovf.t

Conditional Move on False Triple Word



VFPU

### Syntax:

```
vcmovf.t vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is false (0), three floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
d[2] <- t[2];
if( imm3<6 )
begin
    if( !c[imm3] )
    begin
        d[0] <- s[0];
        d[1] <- s[1];
        d[2] <- s[2];
    end
end
```

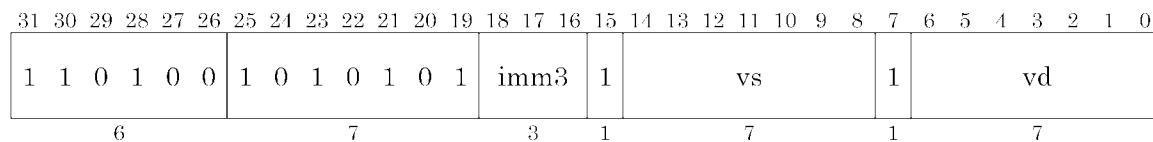
```

        end
    else if( imm3==6 )
        begin
            if( !c[0] )
                d[0] <- s[0];
            if( !c[1] )
                d[1] <- s[1];
            if( !c[2] )
                d[2] <- s[2];
        end
    WriteMatrix( TRIPLEWORD, vd, d );

```

## vcmovf.q

Conditional Move on False Quad Word



VFPU

### Syntax:

```
vcmovf.q vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is false (0), four floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
d[2] <- t[2];
d[3] <- t[3];
if( imm3<6 )
begin
    if( !c[imm3] )
begin
    d[0] <- s[0];
    d[1] <- s[1];
    d[2] <- s[2];
```

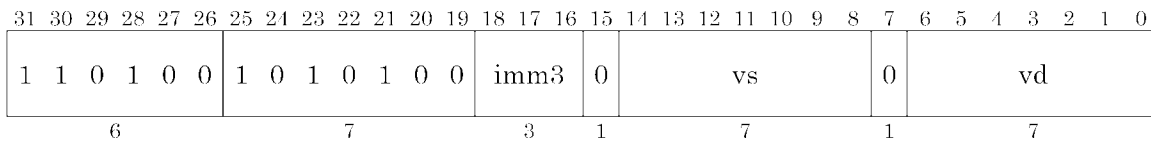


```

                d[3] <- s[3];
            end
        end
    else if( imm3==6 )
        begin
            if( !c[0] )
                d[0] <- s[0];
            if( !c[1] )
                d[1] <- s[1];
            if( !c[2] )
                d[2] <- s[2];
            if( !c[3] )
                d[3] <- s[3];
            end
        WriteMatrix( QUADWORD, vd, d );
    
```

## vcmovt.s

Conditional Move on True Single Word



VFPU

### Syntax:

```
vcmovt.s vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5          pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is true (1), one floating-point element from the matrix register indicated by vs is copied to the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
if( imm3<6 )
begin
    if( c[imm3] )
    begin
        d[0] <- s[0];
    end
end
else if( imm3==6 )
begin
    if( c[0] )
```

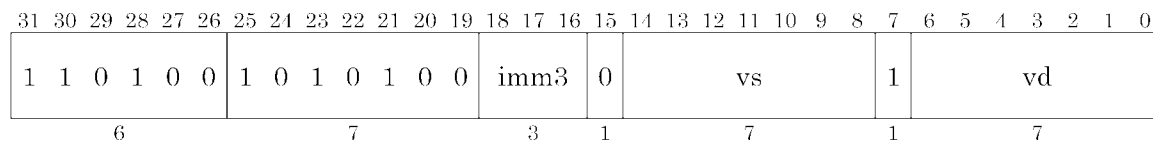
```

        d[0] <- s[0];
    end
    WriteMatrix( SINGLEWORD, vd, d );

```

## vcmovt.p

Conditional Move on True Pair Word



VFPU

### Syntax:

```
vcmovt.p vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is true (1), two floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
if( imm3<6 )
begin
    if( c[imm3] )
    begin
        d[0] <- s[0];
        d[1] <- s[1];
    end
end
else if( imm3==6 )
```

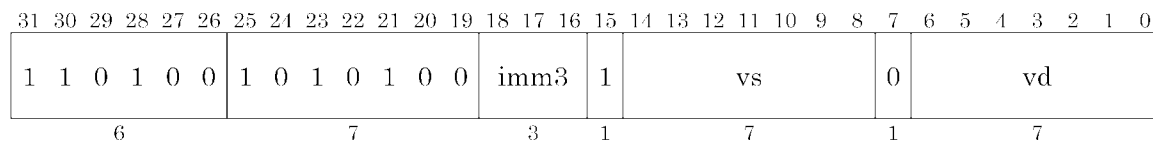
```

begin
  if( c[0] )
    d[0] <- s[0];
  if( c[1] )
    d[1] <- s[1];
end
WriteMatrix( PAIRWORD, vd, d );

```

## vcmovt.t

### Conditional Move on True Triple Word



VFPU

#### Syntax:

```
vcmovt.t vd, vs, imm3
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 5      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

#### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is true (1), three floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
d[2] <- t[2];
if( imm3<6 )
begin
    if( c[imm3] )
    begin
        d[0] <- s[0];
        d[1] <- s[1];
        d[2] <- s[2];
    end
end
```

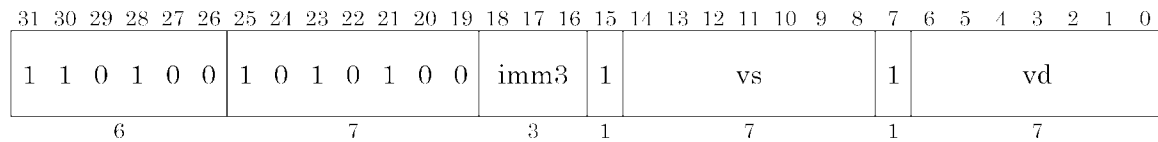
```

        end
    else if( imm3==6 )
        begin
            if( c[0] )
                d[0] <- s[0];
            if( c[1] )
                d[1] <- s[1];
            if( c[2] )
                d[2] <- s[2];
        end
    WriteMatrix( TRIPLEWORD, vd, d );

```

## vcmovt.q

Conditional Move on True Quad Word



VFPU

### Syntax:

```
vcmovt.q vd, vs, imm3
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

If the condition of the VFPU\_CC control register bit indicated by the imm3 field is true (1), four floating-point elements from the matrix registers indicated by vs are copied to locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vd );
c <- ReadControlBit( VFPU_CC );
d[0] <- t[0];
d[1] <- t[1];
d[2] <- t[2];
d[3] <- t[3];
if( imm3<6 )
begin
    if( c[imm3] )
begin
    d[0] <- s[0];
    d[1] <- s[1];
    d[2] <- s[2];
```



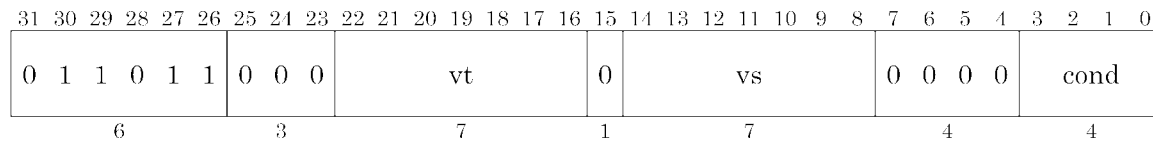
```

        d[3] <- s[3];
    end
end
else if( imm3==6 )
begin
    if( c[0] )
        d[0] <- s[0];
    if( c[1] )
        d[1] <- s[1];
    if( c[2] )
        d[2] <- s[2];
    if( c[3] )
        d[3] <- s[3];
    end
WriteMatrix( QUADWORD, vd, d );

```

## vcmp.s

Compare Single Word



VFPU

### Syntax:

`vcmp.s cond, vs, vt`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	No effect

### Description:

One element from the matrix register indicated by vs is compared with one element from the matrix register indicated by vt according to the condition cond. The elements are treated as floating-point numbers. The 6-bit comparison result is stored in control register VFPU\_CC.

The following mnemonics can be used for cond.

Code (cond)	Mnemonic	Function
0	FL	Always false
1	EQ	Equal
2	LT	Less than
3	LE	Less than or equal
4	TR	Always true
5	NE	Not equal
6	GE	Greater than or equal
7	GT	Greater than
8	EZ	Equal to zero
9	EN	Equal to NaN

Code (cond)	Mnemonic	Function
10	EI	Absolute value equal to infinity
11	ES	Equal to infinity or NaN
12	NZ	Not equal to zero
13	NN	Not equal to NaN
14	NI	Absolute value not equal to infinity
15	NS	Not equal to infinity and not equal to NaN

**Operation:**

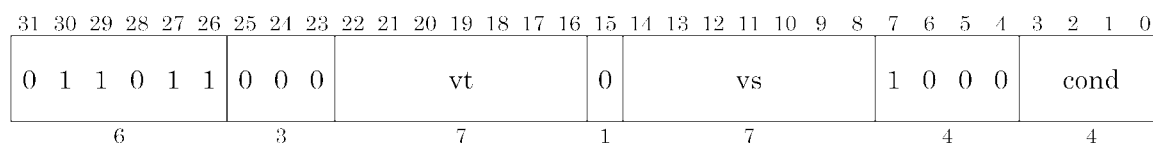
```

s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
c[0] <- Compare( cond, s[0], t[0] );
c[4] <- c[0];
c[5] <- c[0];
WriteControlBit( VFPU_CC, 0, c[0] );
WriteControlBit( VFPU_CC, 4, c[4] );
WriteControlBit( VFPU_CC, 5, c[5] );

```

## vcmp.p

Compare Pair Word



VFPU

### Syntax:

`vcmp.p cond, vs, vt`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	No effect

### Description:

Two elements from the matrix registers indicated by vs are compared with two elements from the matrix registers indicated by vt according to the condition cond. The elements are treated as floating-point numbers. The 6-bit comparison result is stored in control register VFPU\_CC.

The following mnemonics can be used for cond.

Code (cond)	Mnemonic	Function
0	FL	Always false
1	EQ	Equal
2	LT	Less than
3	LE	Less than or equal
4	TR	Always true
5	NE	Not equal
6	GE	Greater than or equal
7	GT	Greater than
8	EZ	Equal to zero
9	EN	Equal to NaN

Code (cond)	Mnemonic	Function
10	EI	Absolute value equal to infinity
11	ES	Equal to infinity or NaN
12	NZ	Not equal to zero
13	NN	Not equal to NaN
14	NI	Absolute value not equal to infinity
15	NS	Not equal to infinity and not equal to NaN

#### Operation:

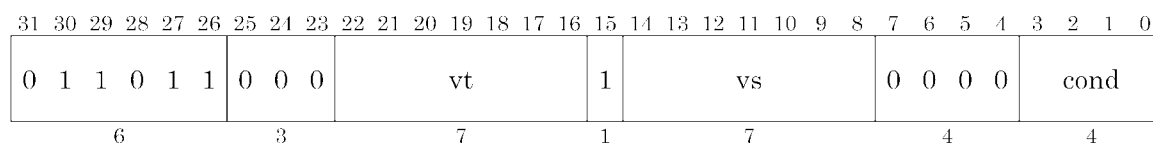
```

s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
c[0] <- Compare( cond, s[0], t[0] );
c[4] <- c[0];
c[5] <- c[0];
WriteControlBit( VFPU_CC, 0, c[0] );
c[1] <- Compare( cond, s[1], t[1] );
c[4] <- c[4] | c[1];
c[5] <- c[5] & c[1];
WriteControlBit( VFPU_CC, 1, c[1] );
WriteControlBit( VFPU_CC, 4, c[4] );
WriteControlBit( VFPU_CC, 5, c[5] );

```

## vcmp.t

Compare Triple Word



VFPU

### Syntax:

`vcmp.t cond, vs, vt`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	No effect

### Description:

Three elements from the matrix registers indicated by vs are compared with three elements from the matrix registers indicated by vt according to the condition cond. The elements are treated as floating-point numbers. The 6-bit comparison result is stored in control register VFPU\_CC.

The following mnemonics can be used for cond.

Code (cond)	Mnemonic	Function
0	FL	Always false
1	EQ	Equal
2	LT	Less than
3	LE	Less than or equal
4	TR	Always true
5	NE	Not equal
6	GE	Greater than or equal
7	GT	Greater than
8	EZ	Equal to zero
9	EN	Equal to NaN

Code (cond)	Mnemonic	Function
10	EI	Absolute value equal to infinity
11	ES	Equal to infinity or NaN
12	NZ	Not equal to zero
13	NN	Not equal to NaN
14	NI	Absolute value not equal to infinity
15	NS	Not equal to infinity and not equal to NaN

**Operation:**

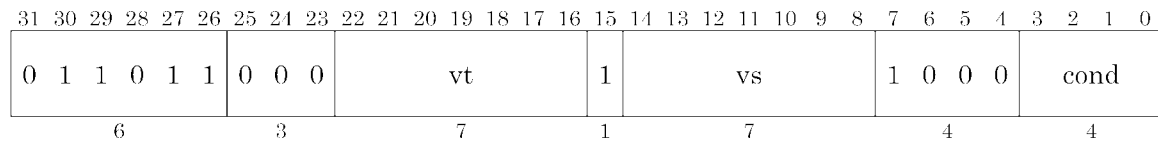
```

s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
c[0] <- Compare( cond, s[0], t[0] );
c[4] <- c[0];
c[5] <- c[0];
WriteControlBit( VFPU_CC, 0, c[0] );
c[1] <- Compare( cond, s[1], t[1] );
c[4] <- c[4] | c[1];
c[5] <- c[5] & c[1];
WriteControlBit( VFPU_CC, 1, c[1] );
c[2] <- Compare( cond, s[2], t[2] );
c[4] <- c[4] | c[2];
c[5] <- c[5] & c[2];
WriteControlBit( VFPU_CC, 2, c[2] );
WriteControlBit( VFPU_CC, 4, c[4] );
WriteControlBit( VFPU_CC, 5, c[5] );

```

## vcmp.q

Compare Quad Word



VFPU

### Syntax:

`vcmp.q cond, vs, vt`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	No effect

### Description:

Four elements from the matrix registers indicated by vs are compared with four elements from the matrix registers indicated by vt according to the condition cond. The elements are treated as floating-point numbers. The 6-bit comparison result is stored in control register VFPU\_CC.

The following mnemonics can be used for cond.

Code (cond)	Mnemonic	Function
0	FL	Always false
1	EQ	Equal
2	LT	Less than
3	LE	Less than or equal
4	TR	Always true
5	NE	Not equal
6	GE	Greater than or equal
7	GT	Greater than
8	EZ	Equal to zero
9	EN	Equal to NaN



Code (cond)	Mnemonic	Function
10	EI	Absolute value equal to infinity
11	ES	Equal to infinity or NaN
12	NZ	Not equal to zero
13	NN	Not equal to NaN
14	NI	Absolute value not equal to infinity
15	NS	Not equal to infinity and not equal to NaN

#### Operation:

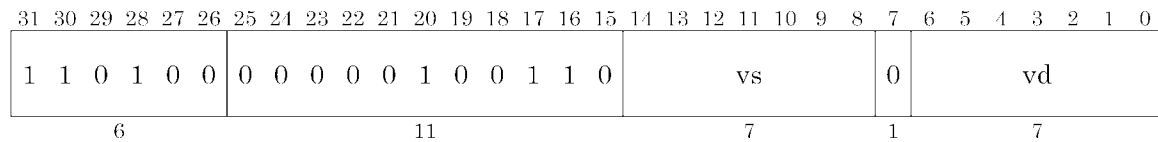
```

s <- ReadMatrix( QUADWORD, vs );
t <- ReadMatrix( QUADWORD, vt );
c[0] <- Compare( cond, s[0], t[0] );
c[4] <- c[0];
c[5] <- c[0];
WriteControlBit( VFPU_CC, 0, c[0] );
c[1] <- Compare( cond, s[1], t[1] );
c[4] <- c[4] | c[1];
c[5] <- c[5] & c[1];
WriteControlBit( VFPU_CC, 1, c[1] );
c[2] <- Compare( cond, s[2], t[2] );
c[4] <- c[4] | c[2];
c[5] <- c[5] & c[2];
WriteControlBit( VFPU_CC, 2, c[2] );
c[3] <- Compare( cond, s[3], t[3] );
c[4] <- c[4] | c[3];
c[5] <- c[5] & c[3];
WriteControlBit( VFPU_CC, 3, c[3] );
WriteControlBit( VFPU_CC, 4, c[4] );
WriteControlBit( VFPU_CC, 5, c[5] );

```

## VCOS.S

Cosine Single Word



VFPU

### Syntax:

`vcos.s vd, vs`

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 7      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	No effect	Valid

### Description:

The cosine of the floating-point value of one element from the matrix register indicated by `vs` is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_cos}(\frac{\pi}{2} \times x) - \cos(x) | < 2^{-20} ; \quad -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_cos}(\text{nan}) = \text{nan}$$

$$\text{approx\_cos}(+\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(-\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(+0.0) = +1.0$$

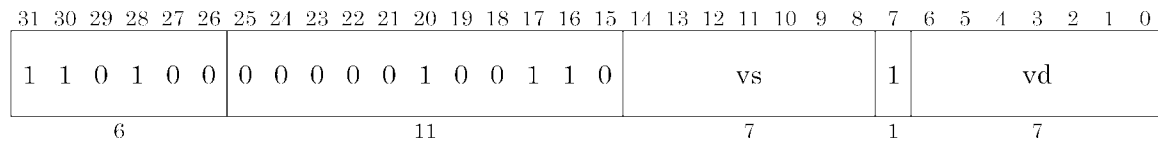
$$\text{approx\_cos}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( SINGLEWORD, vs );
d[0] <- approx_cos( M_PI_2 * s[0] );
WriteMatrix( SINGLEWORD, vd, d );
```

## vcos.p

Cosine Pair Word



VFPU

### Syntax:

`vcos.p vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 8      pitch : 2

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The cosines of the floating-point values of two elements from the matrix registers indicated by `vs` are calculated. The two-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_cos}(\frac{\pi}{2} \times x) - \cos(x) | < 2^{-20}; \quad -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_cos}(\text{nan}) = \text{nan}$$

$$\text{approx\_cos}(+\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(-\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(+0.0) = +1.0$$

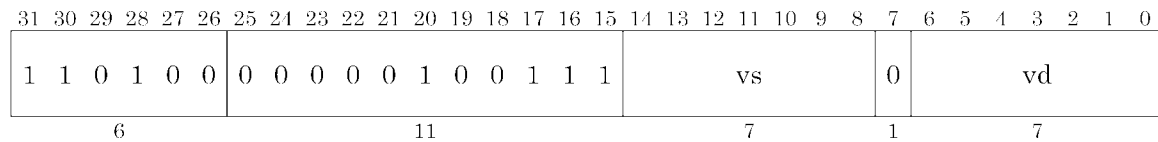
$$\text{approx\_cos}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( PAIRWORD, vs );
d[0] <- approx_cos( M_PI_2 * s[0] );
d[1] <- approx_cos( M_PI_2 * s[1] );
WriteMatrix( PAIRWORD, vd, d );
```

## vcos.t

Cosine Triple Word



VFPU

### Syntax:

`vcos.t vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 9      pitch : 3

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The cosines of the floating-point values of three elements from the matrix registers indicated by `vs` are calculated. The three-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_cos}(\frac{\pi}{2} \times x) - \cos(x) | < 2^{-20}; \quad -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_cos}(\text{nan}) = \text{nan}$$

$$\text{approx\_cos}(+\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(-\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(+0.0) = +1.0$$

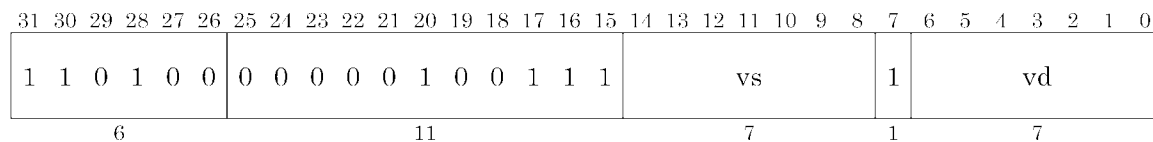
$$\text{approx\_cos}(-0.0) = +1.0$$

**Operation:**

```
s <- ReadMatrix( TRIPLEWORD, vs );
d[0] <- approx_cos( M_PI_2 * s[0] );
d[1] <- approx_cos( M_PI_2 * s[1] );
d[2] <- approx_cos( M_PI_2 * s[2] );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vcos.q

Cosine Quad Word



VFPU

### Syntax:

`vcos.q vd, vs`

### Instruction Type

Repeat (pipeline) instruction

### Processing Time:

latency : 10      pitch : 4

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	No effect	Use prohibited

### Description:

The cosines of the floating-point values of four elements from the matrix registers indicated by `vs` are calculated. The four-element floating-point result is stored at locations in the matrix register file indicated by `vd`.

The valid input range is as follows.

$$-2^{32} < x < 2^{32}$$

The precision of the calculation is given by the following expression.

$$| \text{approx\_cos}(\frac{\pi}{2} \times x) - \cos(x) | < 2^{-20}; \quad -2^{32} < x < 2^{32}$$

Special solutions are as follows.

$$\text{approx\_cos}(\text{nan}) = \text{nan}$$

$$\text{approx\_cos}(+\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(-\text{inf}) = \text{nan}$$

$$\text{approx\_cos}(+0.0) = +1.0$$

$$\text{approx\_cos}(-0.0) = +1.0$$

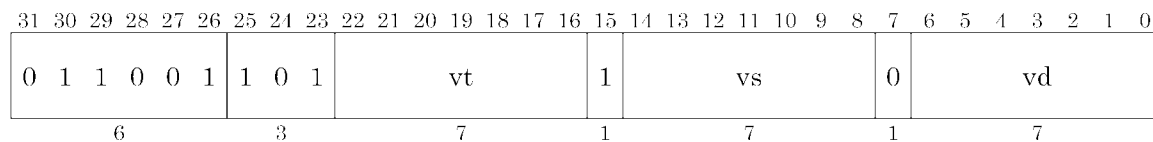


**Operation:**

```
s <- ReadMatrix( QUADWORD, vs );
d[0] <- approx_cos( M_PI_2 * s[0] );
d[1] <- approx_cos( M_PI_2 * s[1] );
d[2] <- approx_cos( M_PI_2 * s[2] );
d[3] <- approx_cos( M_PI_2 * s[3] );
WriteMatrix( QUADWORD, vd, d );
```

## vcrs.t

Cross Triple Word



VFPU

### Syntax:

```
vcrs.t vd, vs, vt
```

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 5      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Valid

### Description:

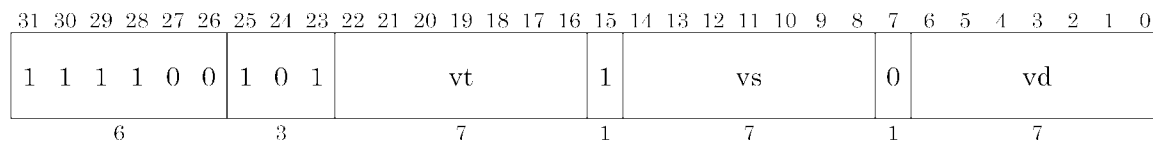
The first part of the cross product between three elements from the matrix registers indicated by vs and three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[1] * t[2];
d[1] <- s[2] * t[0];
d[2] <- s[0] * t[1];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vcrsp.t

### Cross Product Triple Word



VFPU

#### Syntax:

```
vcrsp.t vd, vs, vt
```

#### Instruction Type

Repeat (pipeline) instruction

#### Processing Time:

latency : 9      pitch : 3

#### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

#### Description:

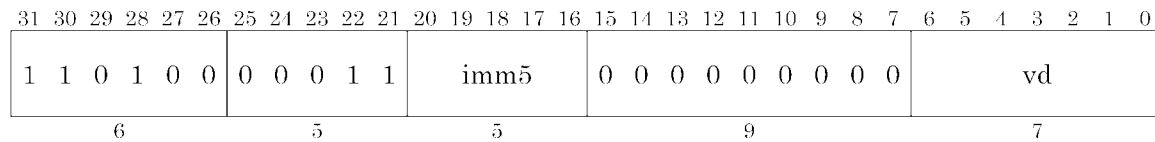
The cross product between three elements from the matrix registers indicated by vs and three elements from the matrix registers indicated by vt is calculated. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- + s[1]*t[2] - s[2]*t[1];
d[1] <- + s[2]*t[0] - s[0]*t[2];
d[2] <- + s[0]*t[1] - s[1]*t[0];
WriteMatrix( TRIPLEWORD, vd, d );
```

## vcst.s

Set Constant Single Word



VFPU

### Syntax:

vcst.s vd, imm5

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

The constant indicated by imm5 is stored as a one-element floating-point value at the location in the matrix register file indicated by vd.

The table below shows the constants that can be specified for imm5 and their corresponding mnemonics.

Code (imm5)	Mnemonic	Value	Expression
0	-	Undefined	-
1	VFPU_HUGE	3.402823e+38	Maximum value that can be represented by a 32-bit single-precision floating-point number
2	VFPU_SQRT2	1.414214e+00	$\sqrt{2}$
3	VFPU_SQRT1_2	7.071068e-01	$\sqrt{\frac{1}{2}}$
4	VFPU_2_SQRTPI	1.128379e+00	$\frac{2}{\sqrt{\pi}}$
5	VFPU_2_PI	6.366197e-01	$\frac{2}{\pi}$
6	VFPU_1_PI	3.183099e-01	$\frac{1}{\pi}$
7	VFPU_PI_4	7.853982e-01	$\frac{\pi}{4}$

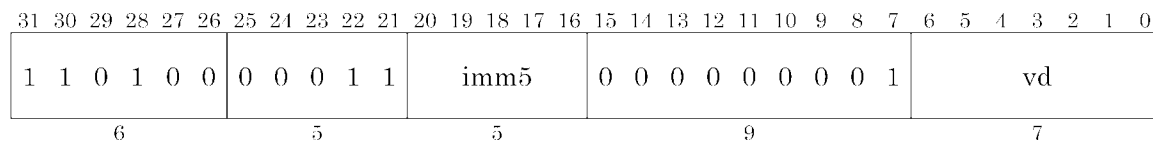
Code (imm5)	Mnemonic	Value	Expression
8	VFPU_PI_2	1.570796e+00	$\frac{\pi}{2}$
9	VFPU_PI	3.141593e+00	$\pi$
10	VFPU_E	2.718282e+00	e (base of natural logarithm)
11	VFPU_LOG2E	1.442695e+00	$\log_2 e$
12	VFPU_LOG10E	4.342945e-01	$\log_{10} e$
13	VFPU_LN2	6.931472e-01	$\ln 2$
14	VFPU_LN10	2.302585e+00	$\ln 10$
15	VFPU_2PI	6.283185e+00	$2\pi$
16	VFPU_PI_6	5.235988e-01	$\frac{\pi}{6}$
17	VFPU_LOG10TWO	3.010300e-01	$\log_{10} 2$
18	VFPU_LOG2TEN	3.321928e+00	$\log_2 10$
19	VFPU_SQRT3_2	8.660254e-01	$\frac{\sqrt{3}}{2}$
20	-	Undefined	-
21	-	Undefined	-
22	-	Undefined	-
23	-	Undefined	-
24	-	Undefined	-
25	-	Undefined	-
26	-	Undefined	-
27	-	Undefined	-
28	-	Undefined	-
29	-	Undefined	-
30	-	Undefined	-
31	-	Undefined	-

#### Operation:

```
d[0] <- const( imm5 );
WriteMatrix( SINGLEWORD, vd, d );
```

## vcst.p

Set Constant Pair Word



VFPU

### Syntax:

vcst.p vd, imm5

### Instruction Type

Pipeline instruction

### Processing Time:

latency : 3      pitch : 1

### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

### Description:

The constant indicated by imm5 is stored as a two-element floating-point value at locations in the matrix register file indicated by vd.

The table below shows the constants that can be specified for imm5 and their corresponding mnemonics.

Code (imm5)	Mnemonic	Value	Expression
0	-	Undefined	-
1	VFPU_HUGE	3.402823e+38	Maximum value that can be represented by a 32-bit single-precision floating-point number
2	VFPU_SQRT2	1.414214e+00	$\sqrt{2}$
3	VFPU_SQRT1_2	7.071068e-01	$\sqrt{\frac{1}{2}}$
4	VFPU_2_SQRTPI	1.128379e+00	$\frac{2}{\sqrt{\pi}}$
5	VFPU_2_PI	6.366197e-01	$\frac{2}{\pi}$
6	VFPU_1_PI	3.183099e-01	$\frac{1}{\pi}$
7	VFPU_PI_4	7.853982e-01	$\frac{\pi}{4}$

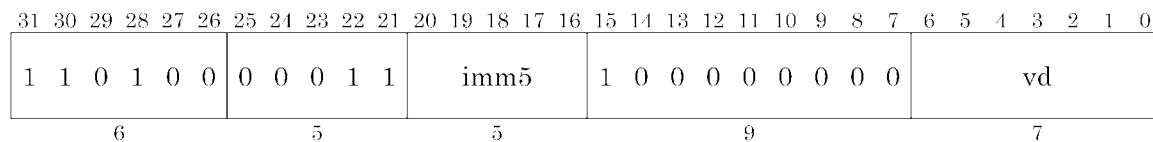
Code (imm5)	Mnemonic	Value	Expression
8	VFPU_PI_2	1.570796e+00	$\frac{\pi}{2}$
9	VFPU_PI	3.141593e+00	$\pi$
10	VFPU_E	2.718282e+00	e (base of natural logarithm)
11	VFPU_LOG2E	1.442695e+00	$\log_2 e$
12	VFPU_LOG10E	4.342945e-01	$\log_{10} e$
13	VFPU_LN2	6.931472e-01	$\ln 2$
14	VFPU_LN10	2.302585e+00	$\ln 10$
15	VFPU_2PI	6.283185e+00	$2\pi$
16	VFPU_PI_6	5.235988e-01	$\frac{\pi}{6}$
17	VFPU_LOG10TWO	3.010300e-01	$\log_{10} 2$
18	VFPU_LOG2TEN	3.321928e+00	$\log_2 10$
19	VFPU_SQRT3_2	8.660254e-01	$\frac{\sqrt{3}}{2}$
20	-	Undefined	-
21	-	Undefined	-
22	-	Undefined	-
23	-	Undefined	-
24	-	Undefined	-
25	-	Undefined	-
26	-	Undefined	-
27	-	Undefined	-
28	-	Undefined	-
29	-	Undefined	-
30	-	Undefined	-
31	-	Undefined	-

#### Operation:

```
d[0] <- const( imm5 );
d[1] <- const( imm5 );
WriteMatrix( PAIRWORD, vd, d );
```

## vcst.t

### Set Constant Triple Word



VFPU

#### Syntax:

vcst.t vd, imm5

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

#### Description:

The constant indicated by imm5 is stored as a three-element floating-point value at locations in the matrix register file indicated by vd.

The table below shows the constants that can be specified for imm5 and their corresponding mnemonics.

Code (imm5)	Mnemonic	Value	Expression
0	-	Undefined	-
1	VFPU_HUGE	3.402823e+38	Maximum value that can be represented by a 32-bit single-precision floating-point number
2	VFPU_SQRT2	1.414214e+00	$\sqrt{2}$
3	VFPU_SQRT1_2	7.071068e-01	$\sqrt{\frac{1}{2}}$
4	VFPU_2_SQRTPI	1.128379e+00	$\frac{2}{\sqrt{\pi}}$
5	VFPU_2_PI	6.366197e-01	$\frac{2}{\pi}$
6	VFPU_1_PI	3.183099e-01	$\frac{1}{\pi}$
7	VFPU_PI_4	7.853982e-01	$\frac{\pi}{4}$



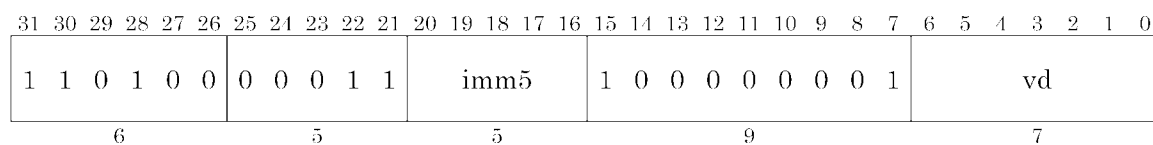
Code (imm5)	Mnemonic	Value	Expression
8	VFPU_PI_2	1.570796e+00	$\frac{\pi}{2}$
9	VFPU_PI	3.141593e+00	$\pi$
10	VFPU_E	2.718282e+00	e (base of natural logarithm)
11	VFPU_LOG2E	1.442695e+00	$\log_2 e$
12	VFPU_LOG10E	4.342945e-01	$\log_{10} e$
13	VFPU_LN2	6.931472e-01	$\ln 2$
14	VFPU_LN10	2.302585e+00	$\ln 10$
15	VFPU_2PI	6.283185e+00	$2\pi$
16	VFPU_PI_6	5.235988e-01	$\frac{\pi}{6}$
17	VFPU_LOG10TWO	3.010300e-01	$\log_{10} 2$
18	VFPU_LOG2TEN	3.321928e+00	$\log_2 10$
19	VFPU_SQRT3_2	8.660254e-01	$\frac{\sqrt{3}}{2}$
20	-	Undefined	-
21	-	Undefined	-
22	-	Undefined	-
23	-	Undefined	-
24	-	Undefined	-
25	-	Undefined	-
26	-	Undefined	-
27	-	Undefined	-
28	-	Undefined	-
29	-	Undefined	-
30	-	Undefined	-
31	-	Undefined	-

#### Operation:

```
d[0] <- const( imm5 );
d[1] <- const( imm5 );
d[2] <- const( imm5 );
WriteMatrix( TRIPLEWORD, vd, d );
```

## vcst.q

### Set Constant Quad Word



VFPU

#### Syntax:

vcst.q vd, imm5

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 3      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
No effect	No effect	Valid

#### Description:

The constant indicated by imm5 is stored as a four-element floating-point value at locations in the matrix register file indicated by vd.

The table below shows the constants that can be specified for imm5 and their corresponding mnemonics.

Code (imm5)	Mnemonic	Value	Expression
0	-	Undefined	-
1	VFPU_HUGE	3.402823e+38	Maximum value that can be represented by a 32-bit single-precision floating-point number
2	VFPU_SQRT2	1.414214e+00	$\sqrt{2}$
3	VFPU_SQRT1_2	7.071068e-01	$\sqrt{\frac{1}{2}}$
4	VFPU_2_SQRTPI	1.128379e+00	$\frac{2}{\sqrt{\pi}}$
5	VFPU_2_PI	6.366197e-01	$\frac{2}{\pi}$
6	VFPU_1_PI	3.183099e-01	$\frac{1}{\pi}$
7	VFPU_PI_4	7.853982e-01	$\frac{\pi}{4}$

Code (imm5)	Mnemonic	Value	Expression
8	VFPU_PI_2	1.570796e+00	$\frac{\pi}{2}$
9	VFPU_PI	3.141593e+00	$\pi$
10	VFPU_E	2.718282e+00	e (base of natural logarithm)
11	VFPU_LOG2E	1.442695e+00	$\log_2 e$
12	VFPU_LOG10E	4.342945e-01	$\log_{10} e$
13	VFPU_LN2	6.931472e-01	$\ln 2$
14	VFPU_LN10	2.302585e+00	$\ln 10$
15	VFPU_2PI	6.283185e+00	$2\pi$
16	VFPU_PI_6	5.235988e-01	$\frac{\pi}{6}$
17	VFPU_LOG10TWO	3.010300e-01	$\log_{10} 2$
18	VFPU_LOG2TEN	3.321928e+00	$\log_2 10$
19	VFPU_SQRT3_2	8.660254e-01	$\frac{\sqrt{3}}{2}$
20	-	Undefined	-
21	-	Undefined	-
22	-	Undefined	-
23	-	Undefined	-
24	-	Undefined	-
25	-	Undefined	-
26	-	Undefined	-
27	-	Undefined	-
28	-	Undefined	-
29	-	Undefined	-
30	-	Undefined	-
31	-	Undefined	-

#### Operation:

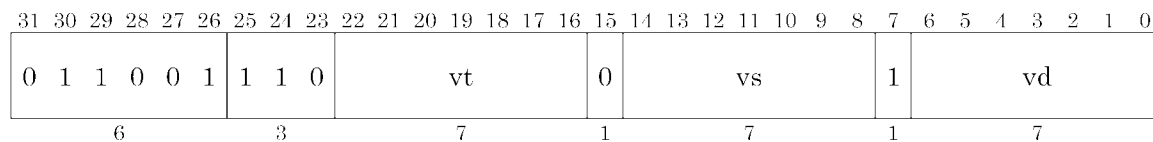
```

d[0] <- const( imm5 );
d[1] <- const( imm5 );
d[2] <- const( imm5 );
d[3] <- const( imm5 );
WriteMatrix( QUADWORD, vd, d );

```

## vdet.p

### 2X2 Matrix Determinant



VFPU

#### Syntax:

```
vdet.p vd, vs, vt
```

#### Instruction Type

Pipeline instruction

#### Processing Time:

latency : 7      pitch : 1

#### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Use prohibited	Valid

#### Description:

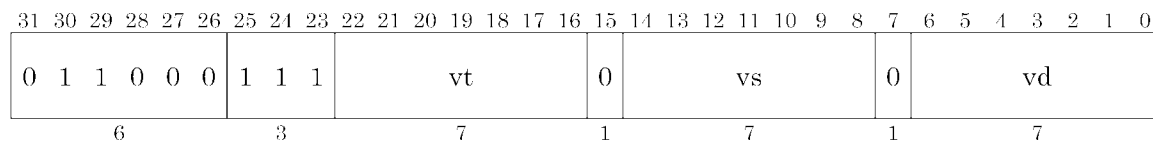
Two elements from the matrix registers indicated by vs and two elements from the matrix registers indicated by vt are treated as elements of a 2x2 matrix. The elements are treated as floating-point numbers and the determinant of the matrix is calculated. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

#### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] * t[1] - s[1] * t[0];
WriteMatrix( SINGLEWORD, vd, d );
```

## vdiv.s

Divide Single Word



VFPU

### Syntax:

```
vdiv.s vd, vs, vt
```

### Instruction Type

Multi-cycle instruction

### Processing Time:

latency : 17      pitch : 14

### Prefixing:

vpfxs	vpfxt	vpfxd
Valid	Valid	Valid

### Description:

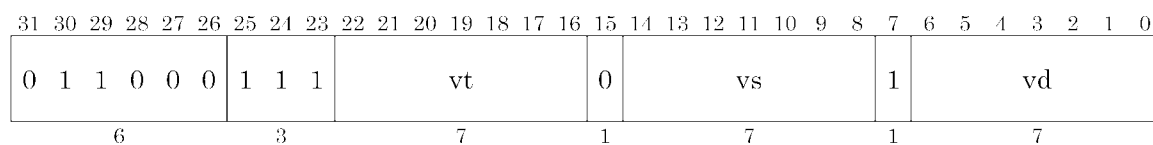
One element from the matrix register indicated by vs is divided by one element from the matrix register indicated by vt. The elements are treated as floating-point numbers. The one-element floating-point result is stored at the location in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( SINGLEWORD, vs );
t <- ReadMatrix( SINGLEWORD, vt );
d[0] <- s[0] / t[0];
WriteMatrix( SINGLEWORD, vd, d );
```

## vdiv.p

Divide Pair Word



VFPU

### Syntax:

```
vdiv.p vd, vs, vt
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 31      pitch : 28

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

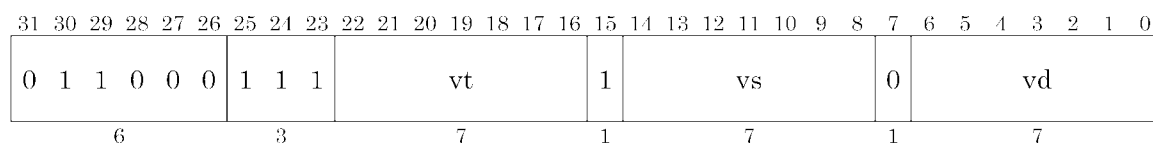
Two elements from the matrix registers indicated by vs are divided by two elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The two-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( PAIRWORD, vs );
t <- ReadMatrix( PAIRWORD, vt );
d[0] <- s[0] / t[0];
d[1] <- s[1] / t[1];
WriteMatrix( PAIRWORD, vd, d );
```

## vdiv.t

Divide Triple Word



VFPU

### Syntax:

```
vdiv.t vd, vs, vt
```

### Instruction Type

Repeat (multi-cycle) instruction

### Processing Time:

latency : 45      pitch : 42

### Prefixing:

vpfxs	vpfxt	vpfxd
Use prohibited	Use prohibited	Use prohibited

### Description:

Three elements from the matrix registers indicated by vs are divided by three elements from the matrix registers indicated by vt. The elements are treated as floating-point numbers. The three-element floating-point result is stored at locations in the matrix register file indicated by vd.

### Operation:

```
s <- ReadMatrix( TRIPLEWORD, vs );
t <- ReadMatrix( TRIPLEWORD, vt );
d[0] <- s[0] / t[0];
d[1] <- s[1] / t[1];
d[2] <- s[2] / t[2];
WriteMatrix( TRIPLEWORD, vd, d );
```