QPX Architecture

Quad Processing eXtension to the Power ISA^{TM}

May 9, 2012

Thomas Fox

foxy@us.ibm.com

Chapter 1. Quad-Vector Floating-Point Facility Overview

This document defines the Quad-Processing eXtension (QPX) to IBM's Power Instruction Set Architecture. Refer to IBM's Power ISA^{TM} AS architecture document for descriptions of the base Power instruction set, the storage model, and related facilities available to the application programmer.

The computational model of the QPX architecture is a vector Single Instruction Multiple Data (SIMD) model with four execution slots and a register file containing 32 registers of 256 bits. Each of the 32 registers can be envisioned as containing four elements of 64 bits, whereby each of the execution slots operates on one vector element.

1.1 Notation

The following notation is specific to and used throughout the QPX Architecture document.

QRT, QRA, QRB, and QRC refer to Quad Floating-Point Registers, which are 256-bit vector registers containing four elements with 64 bits per element. The vector elements are numbered 0,1,2,3, with element 0 comprising bits 0:63, element 1 comprising bits 64:127, element 2 comprising bits 128:191, and element 3 comprising bits 192:255.

QRT^x refers to element x of vector register QRT.

Chapter 2. Quad-Vector Floating-Point Facility Registers

2.1 Quad-Vector Floating-Point Registers

Implementations of this architecture provide 32 Quad-vector floating-Point Registers (QPRs), named QPR0 through QPR31. The QPX instruction formats provide 5-bit fields for specifying the QPRs to be used in the execution of the instruction.

Scalar floating-point computational instructions, defined in the Power ISA, operate on element 0 QPRs, which serve as both the scalar FPRs for scalar instructions and the element 0 QPRs for vector instructions.

The figure below shows the Quad floating-point registers.

QPR0^0	QPR0^1	QPR0^2	QPR0^3
QPR1^0	QPR1^1	QPR1^2	QPR1^3
. . . . . .			
QPR30^0	QPR30^1	QPR30^2	QPR30^3
QPR31^0	QPR31^1	QPR31^2	QPR31^3
0 63	64 127	128 191	192 255

Figure 1. Quad Floating-Point Registers

2.2 Floating-Point Status and Control Register

The Floating-Point Exception Summary bits (32:34) and the Floating-Point Exception bits (35:44 and 53:55) of the FPSCR are never updated by QPX instructions, neither implicitly nor explicitly. The remaining status bits (45:51) are never updated by QPX instructions.

The Floating-Point Exception Enable bits (56:60) are ignored by all QPX instructions, which execute as if these bits were disabled. The Floating-Point Non-IEEE Mode (NI) bit (61) and the Floating-Point Rounding Control (RN) bits (62:63) of the FPSCR affect the operations on all four vector elements for QPX instructions.

2.3 Store Exception Enable Registers

Certain QPX store instructions provide a novel mechanism for the detection and indication of numerically exceptional conditions at the store interface.

A Store Indicate NaN Exception occurs when the source operand of a Store with Indicate instruction contains a NaN value. The Store Nan Exception Enable (SNEE) register enables the indication of such an exception. If an enabled Store Indicate NaN Exception occurs, the Auxiliary Processor bit of the Exception Syndrome Register is set (ESR[AP] = ‘1’).

A Store Indicate Infinity Exception occurs when the source operand contains an Infinity value during a Store with Indicate instruction. The Store Infinity Exception Enable (SIEE) register enables the indication of such an exception. If an enabled Store Indicate Infinity Exception occurs, the Auxiliary Processor bit of the Exception Syndrome Register is set (ESR[AP] = ‘1’).

The precedence of simultaneously occurring indication exceptions and memory fault exceptions is implementation defined.

Implementation Note

In the QPU for BGQ, the following bits in the AXUCR0 Special Purpose Register contain the SNEE and SIEE state on a per thread basis:

```
axucr0(20)  :  Thread 0 SNEE
axucr0(21)  :  Thread 0 SIEE
axucr0(22)  :  Thread 1 SNEE
axucr0(23)  :  Thread 1 SIEE
axucr0(24)  :  Thread 2 SNEE
axucr0(25)  :  Thread 2 SIEE
axucr0(26)  :  Thread 3 SNEE
axucr0(27)  :  Thread 3 SIEE
```

Chapter 3. Scalar Instructions

Scalar floating-point load instructions, defined in the Power ISA, cause a replication of the source data across all elements of the target register.

Scalar floating-point move, arithmetic, rounding and conversion, compare, and select instructions, defined in the Power ISA, are executed in execution slot 0. Source operands for these instructions are read from element 0 QPRs, while target results are written to element 0 QPRs. Target elements 1, 2, and 3 are left in an undefined state.

Chapter 4. Quad-Vector Floating-Point Facility Instructions

4.1 Quad-Vector Floating-Point Load Instructions

Quad-Vector Load Floating-point Single indeXed *X-form*

```
qvlfsx    QRT,RA,RB    (X=0)
qvlfsxa   QRT,RA,RB    (X=1)
```

31	QRT	RA	RB	519	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MVAL ← MEM(EA, 16)
QRT^0 ← DOUBLE(MVAL_{0:31})
QRT^1 ← DOUBLE(MVAL_{32:63})
QRT^2 ← DOUBLE(MVAL_{64:95})
QRT^3 ← DOUBLE(MVAL_{96:127})
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as four single-precision vector elements, converted to double-precision format, and placed into register QRT.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Single with Update indeXed *X-form*

```
qvlfsux    QRT,RA,RB    (X=0)
qvlfsuxa   QRT,RA,RB    (X=1)
```

31	QRT	RA	RB	551	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MVAL ← MEM(EA, 16)
QRT^0 ← DOUBLE(MVAL_{0:31})
QRT^1 ← DOUBLE(MVAL_{32:63})
QRT^2 ← DOUBLE(MVAL_{64:95})
QRT^3 ← DOUBLE(MVAL_{96:127})
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as four single-precision vector elements, converted to double-precision format, and placed into register QRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Double indeXed *X-form*

qvlfdx	QRT,RA,RB	(X=0)
qvlfdxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	583	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFE0
QRT ← MEM(EA, 32)
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 32 bytes in storage addressed by the 32-byte-aligned EA are interpreted as four double-precision vector elements, and placed into register QRT.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Double with Update indeXed *X-form*

qvlfdux	QRT,RA,RB	(X=0)
qvlfduxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	615	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFE0
QRT ← MEM(EA, 32)
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The 32 bytes in storage addressed by the 32-byte-aligned EA are interpreted as four double-precision vector elements, and placed into register QRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Complex Single indeXed X-form

qvlfcsx QRT,RA,RB (X=0)
qvlfcsxa QRT,RA,RB (X=1)

31	QRT	RA	RB	7	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF8
MVAL ← MEM(EA, 8)
$QRT^0$ ← DOUBLE($MVAL_{0:31}$)
$QRT^1$ ← DOUBLE($MVAL_{32:63}$)
$QRT^2$ ← DOUBLE($MVAL_{0:31}$)
$QRT^3$ ← DOUBLE($MVAL_{32:63}$)
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 8 bytes in storage addressed by the 8-byte-aligned EA are interpreted as two single-precision vector elements, converted to double-precision format, and replicated into register QRT.

If the X bit is set, and the address is not aligned on an 8-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Complex Double indeXed X-form

qvlfcdx QRT,RA,RB (X=0)
qvlfcdxa QRT,RA,RB (X=1)

31	QRT	RA	RB	71	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MVAL ← MEM(EA, 16)
$QRT^0$ ← $MVAL^0$
$QRT^1$ ← $MVAL^1$
$QRT^2$ ← $MVAL^0$
$QRT^3$ ← $MVAL^1$
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as two double-precision vector elements, and replicated into register QRT.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Complex Single with Update indeXed X-form

qvlfcsux	QRT,RA,RB	(X=0)
qvlfcsuxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	39	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF8
MVAL ← MEM(EA, 8)
QRT$^0$ ← DOUBLE(MVAL$_{0:31}$)
QRT$^1$ ← DOUBLE(MVAL$_{32:63}$)
QRT$^2$ ← DOUBLE(MVAL$_{0:31}$)
QRT$^3$ ← DOUBLE(MVAL$_{32:63}$)
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The 8 bytes in storage addressed by the 8-byte-aligned EA are interpreted as two single-precision vector elements, converted to double-precision format, and replicated into register QRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 8-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point Complex Double with Update indeXed X-form

qvlfcdux	QRT,RA,RB	(X=0)
qvlfcduxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	103	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MVAL ← MEM(EA, 16)
QRT$^0$ ← MVAL$^0$
QRT$^1$ ← MVAL$^1$
QRT$^2$ ← MVAL$^0$
QRT$^3$ ← MVAL$^1$
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as two double-precision vector elements, and replicated into register QRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point as Integer Word Algebraic indeXed X-form

qvlfiwax	QRT,RA,RB	(X=0)
qvlfiwaxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	871	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
M ← MEM(EA, 16)
$QRT^0 ← {}^{32}(M_0)\ \|\ M_{0:31}$
$QRT^1 ← {}^{32}(M_{32})\ \|\ M_{32:63}$
$QRT^2 ← {}^{32}(M_{64})\ \|\ M_{64:95}$
$QRT^3 ← {}^{32}(M_{96})\ \|\ M_{96:127}$
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as 32-bit integers, sign extended to 64-bit integers, and placed into register QRT.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Floating-point as Integer Word and Zero indeXed X-form

qvlfiwzx	QRT,RA,RB	(X=0)
qvlfiwzxa	QRT,RA,RB	(X=1)

31	QRT	RA	RB	839	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
M ← MEM(EA, 16)
$QRT^0 ← {}^{32}0\ \|\ M_{0:31}$
$QRT^1 ← {}^{32}0\ \|\ M_{32:63}$
$QRT^2 ← {}^{32}0\ \|\ M_{64:95}$
$QRT^3 ← {}^{32}0\ \|\ M_{96:127}$
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The 16 bytes in storage addressed by the 16-byte-aligned EA are interpreted as 32-bit integers, zero extended to 64-bit integers, and placed into register QRT.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector Load Permute Control Left Double indeXed *X-form*

qvlpcldx QRT,RA,RB

31	QRT	RA	RB	582	/
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← b + (RB)
AA = EA & 0b11000
```
$QRT^0 \leftarrow$ 0x400 || (AA)$_{58:60}$ || $^{49}0$
$QRT^1 \leftarrow$ 0x400 || (AA+ 8)$_{58:60}$ || $^{49}0$
$QRT^2 \leftarrow$ 0x400 || (AA+16)$_{58:60}$ || $^{49}0$
$QRT^3 \leftarrow$ 0x400 || (AA+24)$_{58:60}$ || $^{49}0$

Let the effective address (EA) be the sum (RA|0)+(RB).

A quad-vector (32 bytes) describing a dynamic double-precision data alignment to be performed using the quad-vector permute instruction *qvfperm* is generated based on the address EA.

The instruction may raise a memory translation exception if EA is not a valid address.

The behavior of this instruction is boundedly undefined when the address does not correspond to at least the natural alignment of an IEEE double precision floating point number.

Special Registers Altered:
None

Programming Note

This instruction allows the implementation of a software based alignment sequence for double-precision floating-point quad-vectors

```
qvlpcldx  qalign, ra, rb
qvlfdux   qmem1, ra, rb
qvlfdux   qmem2, ra, rb
qvfperm   qaligned, qmem1, qmem2, qalign
```

Quad-Vector Load Permute Control Left Single indeXed *X-form*

qvlpclsx QRT,RA,RB

31	QRT	RA	RB	518	/
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← b + (RB)
AA = (EA * 2) & 0b11000
```
$QRT^0 \leftarrow$ 0x400 || (AA)$_{58:60}$ || $^{49}0$
$QRT^1 \leftarrow$ 0x400 || (AA+ 8)$_{58:60}$ || $^{49}0$
$QRT^2 \leftarrow$ 0x400 || (AA+16)$_{58:60}$ || $^{49}0$
$QRT^3 \leftarrow$ 0x400 || (AA+24)$_{58:60}$ || $^{49}0$

Let the effective address (EA) be the sum (RA|0)+(RB).

A quad-vector (32 bytes) describing a dynamic single-precision data alignment to be performed using the quad-vector permute instruction *qvfperm* is generated based on the address EA.

The instruction may raise a memory translation exception if EA is not a valid address.

The behavior of this instruction is boundedly undefined when the address does not correspond to at least the natural alignment of an IEEE double precision floating point number.

Special Registers Altered:
None

Programming Note

This instruction allows the implementation of a software based alignment sequence for single-precision floating-point quad-vectors

```
qvlpclsx  qalign, ra, rb
qvlfsux   qmem1, ra, rb
qvlfsux   qmem2, ra, rb
qvfperm   qaligned, qmem1, qmem2, qalign
```

Quad-Vector Load Permute Control Right Double indeXed *X-form*

qvlpcrdx QRT,RA,RB

31	QRT	RA	RB	70	/
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← b + (RB)
AA = (32 - (EA & 0b11000))
```

$QRT^0 \leftarrow$ 0x400 || $(AA\ )_{58:60}$ || $^{49}0$
$QRT^1 \leftarrow$ 0x400 || $(AA+ 8)_{58:60}$ || $^{49}0$
$QRT^2 \leftarrow$ 0x400 || $(AA+16)_{58:60}$ || $^{49}0$
$QRT^3 \leftarrow$ 0x400 || $(AA+24)_{58:60}$ || $^{49}0$

Let the effective address (EA) be the sum (RA|0)+(RB).

A quad-vector (32 bytes) describing a dynamic data alignment to be performed using the quad-vector permute instruction *qvfperm* is generated based on the address EA.

The instruction may raise a memory translation exception if EA is not a valid address.

The behavior of this instruction is boundedly undefined when the address does not correspond to at least the natural alignment of an IEEE double precision floating point number.

Special Registers Altered:
None

Programming Note

This instruction allows the implementation of a software based alignment sequence for double-precision floating-point quad-vectors.

Quad-Vector Load Permute Control Right Single indeXed *X-form*

qvlpcrsx QRT,RA,RB

31	QRT	RA	RB	6	/
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← b + (RB)
AA = (32 - ((EA * 2) & 0b11000))
```

$QRT^0 \leftarrow$ 0x400 || $(AA\ )_{58:60}$ || $^{49}0$
$QRT^1 \leftarrow$ 0x400 || $(AA+ 8)_{58:60}$ || $^{49}0$
$QRT^2 \leftarrow$ 0x400 || $(AA+16)_{58:60}$ || $^{49}0$
$QRT^3 \leftarrow$ 0x400 || $(AA+24)_{58:60}$ || $^{49}0$

Let the effective address (EA) be the sum (RA|0)+(RB).

A quad-vector (32 bytes) describing a dynamic data alignment to be performed using the quad-vector permute instruction *qvfperm* is generated based on the address EA.

The instruction may raise a memory translation exception if EA is not a valid address.

The behavior of this instruction is boundedly undefined when the address does not correspond to at least the natural alignment of an IEEE double precision floating point number.

Special Registers Altered:
None

Programming Note

This instruction allows the implementation of a software based alignment sequence for single-precision floating-point quad-vectors.

4.2 Quad-Vector Floating-Point Store Instructions

Quad-Vector STore Floating-point Single indeXed *X-form*

qvstfsx QRS,RA,RB (X=0)
qvstfsxa QRS,RA,RB (X=1)

31	QRS	RA	RB	647	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ←SINGLE(QRS0) || SINGLE(QRS1) ||
              SINGLE(QRS2) || SINGLE(QRS3)
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The four vector elements of register QRS are converted to single-precision format and stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Single with Update indeXed *X-form*

qvstfsux QRS,RA,RB (X=0)
qvstfsuxa QRS,RA,RB (X=1)

31	QRS	RA	RB	679	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ←SINGLE(QRS0) || SINGLE(QRS1) ||
              SINGLE(QRS2) || SINGLE(QRS3)
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The four vector elements of register QRS are converted to single-precision format and stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Single indeXed and Indicate X-form

qvstfsxi QRS,RA,RB (X=0)
qvstfsxia QRS,RA,RB (X=1)

31	QRS	RA	RB	645	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← SINGLE(QRS⁰) || SINGLE(QRS¹) ||
              SINGLE(QRS²) || SINGLE(QRS³)

if (SNEE = 1) then
  if (isNaN (QRS⁰) OR
      isNaN (QRS¹) OR
      isNaN (QRS²) OR
      isNaN (QRS³)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS⁰) OR
      isInf (QRS¹) OR
      isInf (QRS²) OR
      isInf (QRS³)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The four vector elements of register QRS are converted to single-precision format and stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Single with Update indeXed and Indicate X-form

qvstfsuxi QRS,RA,RB (X=0)
qvstfsuxia QRS,RA,RB (X=1)

31	QRS	RA	RB	677	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← SINGLE(QRS⁰) || SINGLE(QRS¹) ||
              SINGLE(QRS²) || SINGLE(QRS³)
RA ← EA

if (SNEE = 1) then
  if (isNaN (QRS⁰) OR
      isNaN (QRS¹) OR
      isNaN (QRS²) OR
      isNaN (QRS³)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS⁰) OR
      isInf (QRS¹) OR
      isInf (QRS²) OR
      isInf (QRS³)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA)+(RB).

The four vector elements of register QRS are converted to single-precision format and stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Double indeXed X-form

qvstfdx	QRS,RA,RB	(X=0)
qvstfdxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	711	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFE0
MEM(EA, 32) ← (QRS)
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The contents of register QRS are stored into the 32 bytes in storage addressed by the 32-byte-aligned EA.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Double with Update indeXed X-form

qvstfdux	QRS,RA,RB	(X=0)
qvstfduxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	743	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFE0
MEM(EA, 32) ← (QRS)
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

The contents of register QRS are stored into the 32 bytes in storage addressed by the 32-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Double indeXed and Indicate X-form

qvstfdxi	QRS,RA,RB	(X=0)
qvstfdxia	QRS,RA,RB	(X=1)

31	QRS	RA	RB	709	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFE0
MEM(EA, 32) ← (QRS)

if (SNEE = 1) then
  if (isNaN ($QRS^0$) OR
      isNaN ($QRS^1$) OR
      isNaN ($QRS^2$) OR
      isNaN ($QRS^3$)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf ($QRS^0$) OR
      isInf ($QRS^1$) OR
      isInf ($QRS^2$) OR
      isInf ($QRS^3$)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The contents of register QRS are stored into the 32 bytes in storage addressed by the 32-byte-aligned EA.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Double with Update indeXed and Indicate X-form

qvstfduxi	QRS,RA,RB	(X=0)
qvstfduxia	QRS,RA,RB	(X=1)

31	QRS	RA	RB	741	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFE0
MEM(EA, 32) ← (QRS)
RA ← EA

if (SNEE = 1) then
  if (isNaN ($QRS^0$) OR
      isNaN ($QRS^1$) OR
      isNaN ($QRS^2$) OR
      isNaN ($QRS^3$)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf ($QRS^0$) OR
      isInf ($QRS^1$) OR
      isInf ($QRS^2$) OR
      isInf ($QRS^3$)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA)+(RB).

The contents of register QRS are stored into the 32 bytes in storage addressed by the 32-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 32-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Complex Single indeXed *X-form*

qvstfcsx	QRS,RA,RB	(X=0)
qvstfcsxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	135	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF8
MEM(EA, 8) ← SINGLE(QRS⁰) || SINGLE(QRS¹)
```

Let the effective address (EA) be the sum (RA|0)+(RB).

Vector elements 0 and 1 of register QRS are converted to single-precision format and stored into the 8 bytes in storage addressed by the 8-byte-aligned EA.

If the X bit is set, and the address is not aligned on an 8-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Complex Double indeXed *X-form*

qvstfcdx	QRS,RA,RB	(X=0)
qvstfcdxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	199	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← QRS⁰ ‖ QRS¹
```

Let the effective address (EA) be the sum (RA|0)+(RB).

Vector elements 0 and 1 of register QRS are stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Complex Single indeXed and Indicate X-form

qvstfcsxi	QRS,RA,RB	(X=0)
qvstfcsxia	QRS,RA,RB	(X=1)

31	QRS	RA	RB	133	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF8
MEM(EA, 8) ← SINGLE(QRS^0) || SINGLE(QRS^1)

if (SNEE = 1) then
  if (isNaN (QRS^0) OR
      isNaN (QRS^1)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS^0) OR
      isInf (QRS^1)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA|0)+(RB).

Vector elements 0 and 1 of register QRS are converted to single-precision format and stored into the 8 bytes in storage addressed by the 8-byte-aligned EA.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on an 8-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Complex Double indeXed and Indicate X-form

qvstfcdxi	QRS,RA,RB	(X=0)
qvstfcdxia	QRS,RA,RB	(X=1)

31	QRS	RA	RB	197	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← QRS^0 ‖ QRS^1

if (SNEE = 1) then
  if (isNaN (QRS^0) OR
      isNaN (QRS^1)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS^0) OR
      isInf (QRS^1)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA|0)+(RB).

Vector elements 0 and 1 of register QRS are stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Complex Single with Update indeXed X-form

qvstfcsux	QRS,RA,RB	(X=0)
qvstfcsuxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	167	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF8
MEM(EA, 8) ← SINGLE(QRS$^0$) || SINGLE(QRS$^1$)
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

Vector elements 0 and 1 of register QRS are converted to single-precision format and stored into the 8 bytes in storage addressed by the 8-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on an 8-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Complex Double with Update indeXed X-form

qvstfcdux	QRS,RA,RB	(X=0)
qvstfcduxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	231	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← QRS$^0$ ∥ QRS$^1$
RA ← EA
```

Let the effective address (EA) be the sum (RA)+(RB).

Vector elements 0 and 1 of register QRS are stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

Quad-Vector STore Floating-point Complex Single with Update indeXed and Indicate X-form

```
qvstfcsuxi    QRS,RA,RB                (X=0)
qvstfcsuxia   QRS,RA,RB                (X=1)
```

31	QRS	RA	RB	165	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF8
MEM(EA, 8) ← SINGLE(QRS$^0$) || SINGLE(QRS$^1$)
RA ← EA

if (SNEE = 1) then
  if (isNaN (QRS$^0$) OR
      isNaN (QRS$^1$)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS$^0$) OR
      isInf (QRS$^1$)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA)+(RB).

Vector elements 0 and 1 of register QRS are converted to single-precision format and stored into the 8 bytes in storage addressed by the 8-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on an 8-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point Complex Double with Update indeXed and Indicate X-form

```
qvstfcduxi    QRS,RA,RB                (X=0)
qvstfcduxia   QRS,RA,RB                (X=1)
```

31	QRS	RA	RB	229	X
0	6	11	16	21	31

```
EA ← ((RA) + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ← QRS$^0$ ‖ QRS$^1$
RA ← EA

if (SNEE = 1) then
  if (isNaN (QRS$^0$) OR
      isNaN (QRS$^1$)) then
          ESR[AP] ← 1

if (SIEE = 1) then
  if (isInf (QRS$^0$) OR
      isInf (QRS$^1$)) then
          ESR[AP] ← 1
```

Let the effective address (EA) be the sum (RA)+(RB).

Vector elements 0 and 1 of register QRS are stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.

If any vector element being stored is a NaN (or Infinity), and the corresponding Store NaN (or Infinity) Exception is enabled, then the Auxiliary Processor bit of the Exception Syndrome Register (ESR[AP]) is set.

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
ESR[AP]

Quad-Vector STore Floating-point as Integer Word indeXed X-form

qvstfiwx	QRS,RA,RB	(X=0)
qvstfiwxa	QRS,RA,RB	(X=1)

31	QRS	RA	RB	967	X
0	6	11	16	21	31

```
if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFF0
MEM(EA, 16) ←  $QRS^0_{32:63}$ || $QRS^1_{32:63}$ ||
               $QRS^2_{32:63}$ || $QRS^3_{32:63}$
```

Let the effective address (EA) be the sum (RA|0)+(RB).

The least significant 32 bits of each vector element of register QRS are stored into the 16 bytes in storage addressed by the 16-byte-aligned EA.

If the contents of register QRS were produced, either directly or indirectly, by a *Load Floating-Point Single* instruction, a single-precision *Arithmetic* instruction, or ***frsp***, then the value stored is undefined. (The contents of register QRS are produced directly by such an instruction if QRS is the target register for the instruction. The contents of register QRS are produced indirectly by such an instruction if QRS is the final target register of a sequence of one or more *Floating-Point Move* instructions, with the input to the sequence having been produced directly by such an instruction.)

If the X bit is set, and the address is not aligned on a 16-byte boundary, an exception is raised.

Special Registers Altered:
None

4.3 Quad-Vector Floating-Point Move Instructions

Quad-Vector Floating-point Move Register X-form

qvfmr QRT,QRB

4	QRT	///	QRB	72	/
0	6	11	16	21	31

For each vector element, the contents of register QRB are placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point NEGate X-form

qvfneg QRT,QRB

4	QRT	///	QRB	40	/
0	6	11	16	21	31

For each vector element, the contents of register QRB, with bit 0 inverted, are placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point ABSolute value X-form

qvfabs QRT,QRB

4	QRT	///	QRB	264	/
0	6	11	16	21	31

For each vector element, the contents of register QRB, with bit 0 set to zero, are placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Negative ABSolute value X-form

qvfnabs QRT,QRB

4	QRT	///	QRB	136	/
0	6	11	16	21	31

For each vector element, the contents of register QRB, with bit 0 set to one, are placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point CoPy SiGN X-form

qvfcpsgn QRT,QRA,QRB

4	QRT	QRA	QRB	8	/
0	6	11	16	21	31

```
$QRT_{0}$ ← $QRA_{0}$
$QRT_{1:63}$ ← $QRB_{1:63}$
$QRT_{64}$ ← $QRA_{64}$
$QRT_{65:127}$ ← $QRB_{65:127}$
$QRT_{128}$ ← $QRA_{128}$
$QRT_{129:191}$ ← $QRB_{129:191}$
$QRT_{192}$ ← $QRA_{192}$
$QRT_{193:255}$ ← $QRB_{193:255}$
```

For each vector element, the contents of register QRB, with bit 0 set to the value of bit 0 of register QRA, are placed into register QRT.

Special Registers Altered:
None

4.4 Quad-Vector Floating-Point Arithmetic Instructions

4.4.1 Quad-Vector Floating-Point Elementary Arithmetic Instructions

Quad-Vector Floating-point ADD [Single] A-form

qvfadd QRT,QRA,QRB

4	QRT	QRA	QRB	///	21	/
0	6	11	16	21	26	31

qvfadds QRT,QRA,QRB

0	QRT	QRA	QRB	///	21	/
0	6	11	16	21	26	31

For each vector element, the floating-point operand in register QRA is added to the floating-point operand in register QRB.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands, to form an intermediate sum. All 53 bits of the significand as well as all three guard bits (G, R, and X) enter into the computation.

If a carry occurs, the sum's significand is shifted right one bit position and the exponent is increased by one.

Special Registers Altered:
None

Quad-Vector Floating-point SUBtract [Single] A-form

qvfsub QRT,QRA,QRB

4	QRT	QRA	QRB	///	20	/
0	6	11	16	21	26	31

qvfsubs QRT,QRA,QRB

0	QRT	QRA	QRB	///	20	/
0	6	11	16	21	26	31

For each vector element, the floating-point operand in register QRB is subtracted from the floating-point operand in register QRA.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

The execution of the Floating Subtract instruction is identical to that of Floating Add, except that the contents of QRB participate in the operation with the sign bit (bit 0) inverted.

Special Registers Altered:
None

Quad-Vector Floating-point MULtiply [Single] A-form

qvfmul QRT,QRA,QRC

4	QRT	QRA	///	QRC	25	/
0	6	11	16	21	26	31

qvfmuls QRT,QRA,QRC

0	QRT	QRA	///	QRC	25	/
0	6	11	16	21	26	31

For each vector element, the floating-point operand in register QRA is multiplied by the floating-point operand in register QRC.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Floating-point multiplication is based on exponent addition and multiplication of the significands.

Special Registers Altered:
None

Quad-Vector Floating-point Reciprocal Estimate [Single] A-form

qvfre QRT,QRB

4	QRT	///	QRB	///	24	/
0	6	11	16	21	26	31

qvfres QRT,QRB

0	QRT	///	QRB	///	24	/
0	6	11	16	21	26	31

For each vector element, an estimate of the reciprocal of the floating-point operand in register QRB is placed into register QRT. The estimate placed into register QRT is correct to a precision of one part in 16384 of the reciprocal of (QRB), i.e.,

$$\mathrm{ABS}\left(\frac{\mathrm{estimate} - 1/x}{1/x}\right) \le \frac{1}{16384}$$

where x is the initial value in QRB.

Operation with various special values of the operand is summarized below.

Operand	**Result**
-∞	-0
-0	-∞
+0	+∞
+∞	+0
SNaN	QNaN
QNaN	QNaN

The results of executing this instruction may vary between implementations.

Special Registers Altered:
None

Quad-Vector Floating-point Reciprocal SQuare RooT Estimate [Single] A-form

qvfrsqrte QRT,QRB

4	QRT	///	QRB	///	26	/
0	6	11	16	21	26	31

qvfrsqrtes QRT,QRB

0	QRT	///	QRB	///	26	/
0	6	11	16	21	26	31

For each vector element, an estimate of the reciprocal of the square root of the floating-point operand in register QRB is placed into register QRT. The estimate placed into register QRT is correct to a precision of one part in 16384 of the reciprocal of the square root of (QRB), i.e.,

$$\text{ABS}\left(\frac{\text{estimate} - 1/(\sqrt{x})}{1/(\sqrt{x})}\right) \le \frac{1}{16384}$$

where x is the initial value in QRB.

Operation with various special values of the operand is summarized below.

Operand	**Result**
-∞	QNaN
< 0	QNaN
-0	-∞
+0	+∞
+∞	+0
SNaN	QNaN
QNaN	QNaN

The results of executing this instruction may vary between implementations.

Special Registers Altered:
None

4.4.2 Quad-Vector Floating-Point Multiply-Add Instructions

Quad-Vector Floating-point Multiply-ADD [Single] A-form

qvfmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	29	/
0	6	11	16	21	26	31

qvfmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	29	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow [(QRA^0)\times(QRC^0)] + (QRB^0)$
$QRT^1 \leftarrow [(QRA^1)\times(QRC^1)] + (QRB^1)$
$QRT^2 \leftarrow [(QRA^2)\times(QRC^2)] + (QRB^2)$
$QRT^3 \leftarrow [(QRA^3)\times(QRC^3)] + (QRB^3)$

are performed.

For each vector element, the floating-point operand in register QRA is multiplied by the floating-point operand in register QRC. The floating-point operand in register QRB is added to this intermediate result.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Multiply-SUBtract [Single] A-form

qvfmsub QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	28	/
0	6	11	16	21	26	31

qvfmsubs QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	28	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow [(QRA^0)\times(QRC^0)] - (QRB^0)$
$QRT^1 \leftarrow [(QRA^1)\times(QRC^1)] - (QRB^1)$
$QRT^2 \leftarrow [(QRA^2)\times(QRC^2)] - (QRB^2)$
$QRT^3 \leftarrow [(QRA^3)\times(QRC^3)] - (QRB^3)$

are performed.

For each vector element, the floating-point operand in register QRA is multiplied by the floating-point operand in register QRC. The floating-point operand in register QRB is subtracted from this intermediate result.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Negative Multiply-ADD [Single] A-form

qvfnmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	31	/
0	6	11	16	21	26	31

qvfnmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	31	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow -\ (\ [(QRA^0)\times(QRC^0)] + (QRB^0)\ )$
$QRT^1 \leftarrow -\ (\ [(QRA^1)\times(QRC^1)] + (QRB^1)\ )$
$QRT^2 \leftarrow -\ (\ [(QRA^2)\times(QRC^2)] + (QRB^2)\ )$
$QRT^3 \leftarrow -\ (\ [(QRA^3)\times(QRC^3)] + (QRB^3)\ )$

are performed.

For each vector element, the floating-point operand in register QRA is multiplied by the floating-point operand in register QRC. The floating-point operand in register QRB is added to this intermediate result.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, then negated and placed into register QRT.

This instruction produces the same result as would be obtained by using the *qvfmadd* instruction and then negating the result, with the following exceptions.

QNaNs propagate with no effect on their “sign” bit.
QNaNs that are generated as the result of a disabled Invalid Operation Exception have a “sign” bit of 0.
SNaNs that are converted to QNaNs as the result of a disabled Invalid Operation Exception retain the “sign” bit of the SNaN.

Special Registers Altered:
None

Quad-Vector Floating-point Negative Multiply-SUBtract [Single] A-form

qvfnmsub QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	30	/
0	6	11	16	21	26	31

qvfnmsubs QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	30	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow -\ (\ [(QRA^0)\times(QRC^0)] - (QRB^0)\ )$
$QRT^1 \leftarrow -\ (\ [(QRA^1)\times(QRC^1)] - (QRB^1)\ )$
$QRT^2 \leftarrow -\ (\ [(QRA^2)\times(QRC^2)] - (QRB^2)\ )$
$QRT^3 \leftarrow -\ (\ [(QRA^3)\times(QRC^3)] - (QRB^3)\ )$

are performed.

For each vector element, the floating-point operand in register QRA is multiplied by the floating-point operand in register QRC. The floating-point operand in register QRB is subtracted from this intermediate result.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, then negated and placed into register QRT.

This instruction produces the same result as would be obtained by using the *qvfmsub* instruction and then negating the result, with the following exceptions.

QNaNs propagate with no effect on their “sign” bit.
QNaNs that are generated as the result of a disabled Invalid Operation Exception have a “sign” bit of 0.
SNaNs that are converted to QNaNs as the result of a disabled Invalid Operation Exception retain the “sign” bit of the SNaN.

Special Registers Altered:
None

Quad-Vector Floating-point cross (X) Multiply-ADD [Single] A-form

qvfxmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	9	/
0	6	11	16	21	26	31

qvfxmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	9	/
0	6	11	16	21	26	31

The operations

$$QRT^0 \leftarrow [(QRA^0)\times(QRC^0)] + (QRB^0)$$
$$QRT^1 \leftarrow [(QRA^0)\times(QRC^1)] + (QRB^1)$$
$$QRT^2 \leftarrow [(QRA^2)\times(QRC^2)] + (QRB^2)$$
$$QRT^3 \leftarrow [(QRA^2)\times(QRC^3)] + (QRB^3)$$

are performed.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Programming Note

This instruction is typically used in cross-product multiplication, and in conjunction with qvfxxnp-madd.

Quad-Vector Floating-point double-cross complex (XXNP) Multiply-ADD [Single] A-form

qvfxxnpmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	11	/
0	6	11	16	21	26	31

qvfxxnpmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	11	/
0	6	11	16	21	26	31

The operations

$$QRT^0 \leftarrow -\,(\ [(QRA^1)\times(QRC^1)] - (QRB^0)\ )$$
$$QRT^1 \leftarrow [(QRA^0)\times(QRC^1)] + (QRB^1)$$
$$QRT^2 \leftarrow -\,(\ [(QRA^3)\times(QRC^3)] - (QRB^2)\ )$$
$$QRT^3 \leftarrow [(QRA^2)\times(QRC^3)] + (QRB^3)$$

are performed.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR. For vector elements 0 and 2, the rounded result is negated and placed into register QRT. For vector elements 1 and 3, the rounded result is placed into register QRT.

Special Registers Altered:
None

Programming Note

This instruction is typically used in cross-product multiplication of complex numbers, in conjunction with qvfxmul or qvfxmadd.

$$\begin{bmatrix} M_0+N_0i & M_1+N_1i \\ M_2+N_2i & M_3+N_3i \end{bmatrix} \qquad \begin{bmatrix} P_0+Q_0i & P_2+Q_2i \\ P_1+Q_1i & P_3+Q_3i \end{bmatrix}$$

Consecutive Memory Locations:	M_0	N_0	M_1	N_1	M_2	N_2	M_3	N_3
Separate from above, but consecutive in memory:	P_0	Q_0	P_1	Q_1	P_2	Q_2	P_3	Q_3

A 2x2 matrix MN times a 2x2 matrix PQ produces a resultant 2x2 matrix R

Entry Row1Column1 of the Resultant Matrix R
$= (M_0+N_0i)(P_0+Q_0i) + (M_1+N_1i)(P_1+Q_1i)$
$= M_0P_0+M_0Q_0i+N_0P_0i-N_0Q_0 + M_1P_1+M_1Q_1i+N_1P_1i-N_1Q_1$

	$Element^0$	$Element^1$	$Element^2$	$Element^3$
QPR20	M_0	N_0	M_1	N_1

	$Element^0$	$Element^1$	$Element^2$	$Element^3$
QPR21	P_0	Q_0	P_1	Q_1

qvfxmul QPR22, QPR20, QPR21 (notice A=QPR20 and C=QPR21) yields:

	$Element^0$	$Element^1$	$Element^2$	$Element^3$
QPR22	M_0P_0	M_0Q_0	M_1P_1	M_1Q_1

qvfxxnpmadd QPR23, QPR21, QPR20, QPR22 (notice A=QPR21 and C=QPR20) yields:

	$Element^0$	$Element^1$	$Element^2$	$Element^3$
QPR23	$M_0P_0 - N_0Q_0$	$M_0Q_0 + N_0P_0$	$M_1P_1 - N_1Q_1$	$M_1Q_1 + N_1P_1$

Now need to add $Element^0+Element^2$ and $Element^1+Element^3$

Quad-Vector Floating-point double-cross conjugate (XXCPN) Multiply-ADD [Single] A-form

qvfxxcpnmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	3	/
0	6	11	16	21	26	31

qvfxxcpnmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	3	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow [(QRA^1)\times(QRC^1)] + (QRB^0)$
$QRT^1 \leftarrow -\,([(QRA^0)\times(QRC^1)] - (QRB^1))$
$QRT^2 \leftarrow [(QRA^3)\times(QRC^3)] + (QRB^2)$
$QRT^3 \leftarrow -\,([(QRA^2)\times(QRC^3)] - (QRB^3))$

are performed.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR. For vector elements 0 and 2, the rounded result is placed into register QRT. For vector elements 1 and 3, the rounded result is negated and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point double-cross (XX) Multiply-ADD [Single] A-form

qvfxxmadd QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	1	/
0	6	11	16	21	26	31

qvfxxmadds QRT,QRA,QRC,QRB

0	QRT	QRA	QRB	QRC	1	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow [(QRA^1)\times(QRC^1)] + (QRB^0)$
$QRT^1 \leftarrow [(QRA^0)\times(QRC^1)] + (QRB^1)$
$QRT^2 \leftarrow [(QRA^3)\times(QRC^3)] + (QRB^2)$
$QRT^3 \leftarrow [(QRA^2)\times(QRC^3)] + (QRB^3)$

are performed.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point cross (X) MULtiply [Single] A-form

qvfxmul QRT,QRA,QRC

4	QRT	QRA	///	QRC	17	/
0	6	11	16	21	26	31

qvfxmuls QRT,QRA,QRC

0	QRT	QRA	///	QRC	17	/
0	6	11	16	21	26	31

The operations

$QRT^0 \leftarrow (QRA^0) \times (QRC^0)$
$QRT^1 \leftarrow (QRA^0) \times (QRC^1)$
$QRT^2 \leftarrow (QRA^2) \times (QRC^2)$
$QRT^3 \leftarrow (QRA^2) \times (QRC^3)$

are performed.

For each vector element, the resultant significand may require normalization or denormalization, depending on the values of the two most significant bits of the resultant significand and on the value of the resultant exponent. The result is rounded to the target precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

4.5 Quad-Vector Floating-Point Rounding and Conversion Instructions

4.5.1 Quad-Vector Floating-Point Rounding Instruction

Quad-Vector Floating-point Round to Single-Precision X-form

qvfrsp QRT,QRB

4	QRT	///	QRB	12	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to single-precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:

None

4.5.2 Quad-Vector Floating-Point Convert To/From Integer Instructions

Quad-Vector Floating-point Convert To Integer Doubleword X-form

qvfctid QRT,QRB

4	QRT	///	QRB	814	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer under control of the Floating-Point Rounding Control field (RN) of the FPSCR.

For each vector element, if the rounded floating-point integer is greater than 2^{63} - 1, then QRT is set to `0x7FFF_FFFF_FFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than -2^{63}, then QRT is set to `0x8000_0000_0000_0000`.

Otherwise, for each vector element, QRT is set to the value of the rounded floating-point integer converted to 64-bit signed-integer format.

Special Registers Altered:
None

Quad-Vector Floating-point Convert To Integer Doubleword Unsigned X-form

qvfctidu QRT,QRB

4	QRT	///	QRB	942	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer under control of the Floating-Point Rounding Control field (RN) of the FPSCR.

For each vector element, if the rounded floating-point integer is greater than 2^{64} - 1, then QRT is set to `0xFFFF_FFFF_FFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than 0.0, then QRT is set to `0x0000_0000_0000_0000`.

Otherwise, for each vector element, QRT is set to the value of the rounded floating-point integer converted to 64-bit unsigned-integer format.

Special Registers Altered:
None

Quad-Vector Floating-point Convert To Integer Doubleword with round toward Zero X-form

qvfctidz QRT,QRB

4	QRT	///	QRB	815	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward Zero.

For each vector element, if the rounded floating-point integer is greater than 2^{63} - 1, then QRT is set to `0x7FFF_FFFF_FFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than -2^{63}, then QRT is set to `0x8000_0000_0000_0000`.

Otherwise, for each vector element, QRT is set to the value of the rounded floating-point integer converted to 64-bit signed-integer format.

Special Registers Altered:
None

Quad-Vector Floating-point Convert To Integer Doubleword Unsigned with round toward Zero X-form

qvfctiduz QRT,QRB

4	QRT	///	QRB	943	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward Zero.

For each vector element, if the rounded floating-point integer is greater than 2^{64} - 1, then QRT is set to `0xFFFF_FFFF_FFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than 0.0, then QRT is set to `0x0000_0000_0000_0000`.

Otherwise, for each vector element, QRT is set to the value of the rounded floating-point integer converted to 64-bit unsigned-integer format.

Special Registers Altered:
None

Quad-Vector Floating-point Convert To Integer Word X-form

qvfctiw QRT,QRB

4	QRT	///	QRB	14	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer under control of the Floating-Point Rounding Control field (RN) of the FPSCR.

For each vector element, if the rounded floating-point integer is greater than 2^{31} - 1, then $QRT_{32:63}$ is set to `0x7FFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than -2^{31}, then $QRT_{32:63}$ is set to `0x8000_0000`.

Otherwise, for each vector element, $QRT_{32:63}$ is set to the value of the rounded floating-point integer converted to 32-bit signed-integer format.

$QRT_{0:31}$ of each vector element is undefined.

Special Registers Altered:
None

Implementation Note

In the QPU of BGQ, for each vector element, $QRT_{0:31} \leftarrow$ 0x7FF80000

Quad-Vector Floating-point Convert To Integer Word Unsigned X-form

qvfctiwu QRT,QRB

4	QRT	///	QRB	142	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer under control of the Floating-Point Rounding Control field (RN) of the FPSCR.

For each vector element, if the rounded floating-point integer is greater than 2^{32} - 1, then $QRT_{32:63}$ is set to `0xFFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than 0.0, then $QRT_{32:63}$ is set to `0x0000_0000`.

Otherwise, for each vector element, $QRT_{32:63}$ is set to the value of the rounded floating-point integer converted to 32-bit unsigned-integer format.

$QRT_{0:31}$ of each vector element is undefined.

Special Registers Altered:
None

Implementation Note

In the QPU of BGQ, for each vector element, $QRT_{0:31} \leftarrow$ 0x7FF80000

Quad-Vector Floating-point Convert To Integer Word with round toward Zero X-form

qvfctiwz QRT,QRB

4	QRT	///	QRB	15	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward Zero.

For each vector element, if the rounded floating-point integer is greater than $2^{31} - 1$, then $QRT_{32:63}$ is set to `0x7FFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than -2^{31}, then $QRT_{32:63}$ is set to `0x8000_0000`.

Otherwise, for each vector element, $QRT_{32:63}$ is set to the value of the rounded floating-point integer converted to 32-bit signed-integer format.

$QRT_{0:31}$ of each vector element is undefined.

Special Registers Altered:
None

Implementation Note

In the QPU of BGQ, for each vector element, $QRT_{0:31} \leftarrow$ 0x7FF80000

Quad-Vector Floating-point Convert To Integer Word Unsigned with round toward Zero X-form

qvfctiwuz QRT,QRB

4	QRT	///	QRB	143	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward Zero.

For each vector element, if the rounded floating-point integer is greater than $2^{32} - 1$, then $QRT_{32:63}$ is set to `0xFFFF_FFFF`.

For each vector element, if the rounded floating-point integer is less than 0.0, then $QRT_{32:63}$ is set to `0x0000_0000`.

Otherwise, for each vector element, $QRT_{32:63}$ is set to the value of the rounded floating-point integer converted to 32-bit unsigned-integer format.

$QRT_{0:31}$ of each vector element is undefined.

Special Registers Altered:
None

Implementation Note

In the QPU of BGQ, for each vector element, $QRT_{0:31} \leftarrow$ 0x7FF80000

Quad-Vector Floating-point Convert From Integer Doubleword X-form

qvfcfid QRT,QRB

4	QRT	///	QRB	846	/
0	6	11	16	21	31

For each vector element, the 64-bit signed fixed-point operand in register QRB is converted to an infinitely precise floating-point integer. The result of the conversion is rounded to double-precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Convert From Integer Doubleword Unsigned X-form

qvfcfidu QRT,QRB

4	QRT	///	QRB	974	/
0	6	11	16	21	31

For each vector element, the 64-bit unsigned fixed-point operand in register QRB is converted to an infinitely precise floating-point integer. The result of the conversion is rounded to double-precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Convert From Integer Doubleword Single X-form

qvfcfids QRT,QRB

0	QRT	///	QRB	846	/
0	6	11	16	21	31

For each vector element, the 64-bit signed fixed-point operand in register QRB is converted to an infinitely precise floating-point integer. The result of the conversion is rounded to single-precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Convert From Integer Doubleword Unsigned Single X-form

qvfcfidus QRT,QRB

0	QRT	///	QRB	974	/
0	6	11	16	21	31

For each vector element, the 64-bit unsigned fixed-point operand in register QRB is converted to an infinitely precise floating-point integer. The result of the conversion is rounded to single-precision under control of the Floating-Point Rounding Control field (RN) of the FPSCR, and placed into register QRT.

Special Registers Altered:
None

4.5.3 Quad-Vector Floating-Point Round to Integer Instructions

Quad-Vector Floating-point Round to Integer Nearest X-form

qvfrin QRT,QRB

4	QRT	///	QRB	392	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer as follows, with the result placed into register QRT. If the sign of the operand is positive, (QRB) + 0.5 is truncated to a floating-point integer, otherwise (QRB) - 0.5 is truncated to a floating-point integer.

Special Registers Altered:
None

Quad-Vector Floating-point Round to Integer Plus X-form

qvfrip QRT,QRB

4	QRT	///	QRB	456	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward +Infinity, and the result is placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Round to Integer toward Zero X-form

qvfriz QRT,QRB

4	QRT	///	QRB	424	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward Zero, and the result is placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point Round to Integer Minus X-form

qvfrim QRT,QRB

4	QRT	///	QRB	488	/
0	6	11	16	21	31

For each vector element, the floating-point operand in register QRB is rounded to a floating-point integer using the rounding mode Round toward -Infinity, and the result is placed into register QRT.

Special Registers Altered:
None

4.6 Quad-Vector Floating-Point Compare Instructions

Quad-Vector Floating-point TeST for NAN X-form

qvftstnan QRT,QRA,QRB

4	QRT	QRA	QRB	64	/
0	6	11	16	21	31

```
if isNaN(QRA^0) OR isNaN(QRB^0)
   then QRT^0← 0x3FF0_0000_0000_0000
   else QRT^0← 0xBFF0_0000_0000_0000
if isNaN(QRA^1) OR isNaN(QRB^1)
   then QRT^1← 0x3FF0_0000_0000_0000
   else QRT^1← 0xBFF0_0000_0000_0000
if isNaN(QRA^2) OR isNaN(QRB^2)
   then QRT^2← 0x3FF0_0000_0000_0000
   else QRT^2← 0xBFF0_0000_0000_0000
if isNaN(QRA^3) OR isNaN(QRB^3)
   then QRT^3← 0x3FF0_0000_0000_0000
   else QRT^3← 0xBFF0_0000_0000_0000
```

Each vector element is compared for the specified condition, and the result is encoded. The Boolean value TRUE is encoded as 1.0. The Boolean value of FALSE is encoded as -1.0.

Special Registers Altered:
None

Quad-Vector Floating-point CoMPare Greater Than X-form

qvfcmpgt QRT,QRA,QRB

4	QRT	QRA	QRB	32	/
0	6	11	16	21	31

```
if (QRA^0) > (QRB^0)
   then QRT^0← 0x3FF0_0000_0000_0000
   else QRT^0← 0xBFF0_0000_0000_0000
if (QRA^1) > (QRB^1)
   then QRT^1← 0x3FF0_0000_0000_0000
   else QRT^1← 0xBFF0_0000_0000_0000
if (QRA^2) > (QRB^2)
   then QRT^2← 0x3FF0_0000_0000_0000
   else QRT^2← 0xBFF0_0000_0000_0000
if (QRA^3) > (QRB^3)
   then QRT^3← 0x3FF0_0000_0000_0000
   else QRT^3← 0xBFF0_0000_0000_0000
```

Each vector element is compared for the specified condition, and the result is encoded. The Boolean value TRUE is encoded as 1.0. The Boolean value of FALSE is encoded as -1.0.

When one of the operands is a NaN, the value -1.0 (FALSE) is returned.

Special Registers Altered:
None

Quad-Vector Floating-point CoMPare Less Than X-form

qvfcmplt QRT,QRA,QRB

4	QRT	QRA	QRB	96	/
0	6	11	16	21	31

```
if (QRA⁰) < (QRB⁰)
   then QRT⁰ ← 0x3FF0_0000_0000_0000
   else QRT⁰ ← 0xBFF0_0000_0000_0000
if (QRA¹) < (QRB¹)
   then QRT¹ ← 0x3FF0_0000_0000_0000
   else QRT¹ ← 0xBFF0_0000_0000_0000
if (QRA²) < (QRB²)
   then QRT² ← 0x3FF0_0000_0000_0000
   else QRT² ← 0xBFF0_0000_0000_0000
if (QRA³) < (QRB³)
   then QRT³ ← 0x3FF0_0000_0000_0000
   else QRT³ ← 0xBFF0_0000_0000_0000
```

Each vector element is compared for the specified condition, and the result is encoded. The Boolean value TRUE is encoded as 1.0. The Boolean value of FALSE is encoded as -1.0.

When one of the operands is a NaN, the value -1.0 (FALSE) is returned.

Special Registers Altered:
None

Quad-Vector Floating-point CoMPare EQual X-form

qvfcmpeq QRT,QRA,QRB

4	QRT	QRA	QRB	0	/
0	6	11	16	21	31

```
if (QRA⁰) = (QRB⁰)
   then QRT⁰ ← 0x3FF0_0000_0000_0000
   else QRT⁰ ← 0xBFF0_0000_0000_0000
if (QRA¹) = (QRB¹)
   then QRT¹ ← 0x3FF0_0000_0000_0000
   else QRT¹ ← 0xBFF0_0000_0000_0000
if (QRA²) = (QRB²)
   then QRT² ← 0x3FF0_0000_0000_0000
   else QRT² ← 0xBFF0_0000_0000_0000
if (QRA³) = (QRB³)
   then QRT³ ← 0x3FF0_0000_0000_0000
   else QRT³ ← 0xBFF0_0000_0000_0000
```

Each vector element is compared for the specified condition, and the result is encoded. The Boolean value TRUE is encoded as 1.0. The Boolean value of FALSE is encoded as -1.0.

When one of the operands is a NaN, the value -1.0 (FALSE) is returned.

Special Registers Altered:
None

4.7 Quad Floating-Point Select Instruction

Quad-Vector Floating-point SELectA-form

qvfsel QRT,QRA,QRC,QRB

4	QRT	QRA	QRB	QRC	23	/
0	6	11	16	21	26	31

```
if (QRA$^0$) $\geq$ 0.0
   then QRT$^0$ $\leftarrow$ (QRC$^0$)
   else QRT$^0$ $\leftarrow$ (QRB$^0$)
if (QRA$^1$) $\geq$ 0.0
   then QRT$^1$ $\leftarrow$ (QRC$^1$)
   else QRT$^1$ $\leftarrow$ (QRB$^1$)
if (QRA$^2$) $\geq$ 0.0
   then QRT$^2$ $\leftarrow$ (QRC$^2$)
   else QRT$^2$ $\leftarrow$ (QRB$^2$)
if (QRA$^3$) $\geq$ 0.0
   then QRT$^3$ $\leftarrow$ (QRC$^3$)
   else QRT$^3$ $\leftarrow$ (QRB$^3$)
```

For each vector element, the floating-point operand in register QRA is compared to the value zero. If the operand is greater than or equal to zero, register QRT is set to the contents of register QRC. If the operand is less than zero or is a NaN, register QRT is set to the contents of register QRB. The comparison ignores the sign of zero (i.e., regards +0 as equal to -0).

Special Registers Altered:
None

4.8 Quad-Vector Alignment and Formatting Instructions

Quad-Vector ALIGN Immediate Z23-form

qvaligni QRT,QRA,QRB,VD

4	QRT	QRA	QRB	VD	5	/
0	6	11	16	21	23	31

```
if      VD = 00 then
   QRT ← (QRA)
else if VD = 01 then
   QRT ← (QRA$^1$) || (QRA$^2$) || (QRA$^3$) || (QRB$^0$)
else if VD = 10 then
   QRT ← (QRA$^2$) || (QRA$^3$) || (QRB$^0$) || (QRB$^1$)
else if VD = 11 then
   QRT ← (QRA$^3$) || (QRB$^0$) || (QRB$^1$) || (QRB$^2$)
```

The contents of registers QRA and QRB are concatenated, and a quad-vector is extracted starting at the vector element specified by field VD. The resulting quad-vector is placed into register QRT.

Special Registers Altered:
None

Quad-Vector Floating-point PERMute A-form

qvfperm QRT,QRA,QRB,QRC

4	QRT	QRA	QRB	QRC	6	/
0	6	11	16	21	26	31

For each vector element,

```
   if QRC$_{1:11}$ = 0x400 then
      case QRC$_{12:14}$
         QRT ← (QRA$^0$)  when  000
         QRT ← (QRA$^1$)  when  001
         QRT ← (QRA$^2$)  when  010
         QRT ← (QRA$^3$)  when  011
         QRT ← (QRB$^0$)  when  100
         QRT ← (QRB$^1$)  when  101
         QRT ← (QRB$^2$)  when  110
         QRT ← (QRB$^3$)  when  111
   else
         QRT ← Undefined
```

The contents of registers QRA and QRB are concatenated. A quad-vector is composed from vector elements extracted from the concatenated registers, as specified by the contents of register QRC.

Special Registers Altered:
None

Quad-Vector Element SPLAT Immediate Z23-form

qvesplati QRT,QRA,VD

4	QRT	QRA	//	VD	37	/
0	6	11	16	21	23	31

```
if      VD = 00 then
   QRT ← (QRA$^0$) || (QRA$^0$) || (QRA$^0$) || (QRA$^0$)
else if VD = 01 then
   QRT ← (QRA$^1$) || (QRA$^1$) || (QRA$^1$) || (QRA$^1$)
else if VD = 10 then
   QRT ← (QRA$^2$) || (QRA$^2$) || (QRA$^2$) || (QRA$^2$)
else if VD = 11 then
   QRT ← (QRA$^3$) || (QRA$^3$) || (QRA$^3$) || (QRA$^3$)
```

The vector element from register QRA, specified by field VD, is placed into each vector element of register QRT.

Special Registers Altered:
None

Quad-Vector Generate Permute Control Immediate Z23-form

qvgpci QRT,GPC

4	QRT	GPC	133	/
0	6	11	23	31

```
   QRT$^0$ ← 0x400 || GPC$_{0:2}$ || $^{49}$0
   QRT$^1$ ← 0x400 || GPC$_{3:5}$ || $^{49}$0
   QRT$^2$ ← 0x400 || GPC$_{6:8}$ || $^{49}$0
   QRT$^3$ ← 0x400 || GPC$_{9:11}$ || $^{49}$0
```

Register QRT is loaded with the 12-bit immediate field GPC, dispersed across its four elements, to serve as control for a QVFPERM instruction.

Special Registers Altered:
None

4.9 Floating-Point Boolean Instruction

Quad-Vector Floating-point boolean LOGICAL *X-form*

qvflogical QRT,QRA,QRB,TT

4	QRT	QRA	QRB	TT	4	/
0	6	11	16	21	25	31

For each vector element,

```
if [(QRA) < 0.0 OR isNaN(QRA)]  AND
   [(QRB) < 0.0 OR isNan(QRB)]  then
    if $TT_0$ = 1 then  QRT ← 0x3FF0_0000_0000_0000
                else  QRT ← 0xBFF0_0000_0000_0000
if [(QRA) ≥ 0.0]                AND
   [(QRB) < 0.0 OR isNaN(QRB)]  then
    if $TT_1$ = 1 then  QRT ← 0x3FF0_0000_0000_0000
                else  QRT ← 0xBFF0_0000_0000_0000
if [(QRA) < 0.0 OR isNaN(QRA)]  AND
   [(QRB) ≥ 0.0]                then
    if $TT_2$ = 1 then  QRT ← 0x3FF0_0000_0000_0000
                else  QRT ← 0xBFF0_0000_0000_0000
if [(QRA) ≥ 0.0]                AND
   [(QRB) ≥ 0.0]                then
    if $TT_3$ = 1 then  QRT ← 0x3FF0_0000_0000_0000
                else  QRT ← 0xBFF0_0000_0000_0000
```

The floating-point operands in registers QRA and QRB are treated as boolean values of TRUE if greater than or equal to +/- 0.0, and as FALSE if less than 0.0 or a NaN. Immediate field TT is used in conjunction with these values to create a logical operation.

Programming Note

Some common logical operations can be accessed via pseudo mnemonics, expressed in the table below.

Extended Mnemonic		Equivalent		Function
qvfclr	QRT	qvflogical	QRT,QRT,QRT,0	clear (set as FALSE)
qvfand	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,1	and
qvfandc	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,4	and complement B
qvfctfb	QRT,QRA	qvflogical	QRT,QRA,QRA,5	convert to float-boolean A
qvfxor	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,6	xor
qvfor	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,7	or
qvfnor	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,8	nor
qvfequ	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,9	Boolean equivalent (XNOR)
qvfnot	QRT,QRA	qvflogical	QRT,QRA,QRA,10	not
qvforc	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,13	or complement B
qvfnand	QRT,QRA,QRB	qvflogical	QRT,QRA,QRB,14	nand
qvfset	QRT	qvflogical	QRT,QRT,QRT,15	set (set as TRUE)