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™ 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.

![Figure 1. Quad Floating-Point Registers](image-url)

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)

```
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)

```
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

```
qvldfx QRT,RA,RB (X=0)
qvldfxa QRT,RA,RB (X=1)
```

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 indexed X-form with Update indexed X-form

```
qvldfdux QRT,RA,RB (X=0)
qvldfduxa QRT,RA,RB (X=1)
```

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

<table>
<thead>
<tr>
<th></th>
<th>QRT, RA, RB</th>
<th>(X=0)</th>
<th>QRT, RA, RB</th>
<th>(X=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvlfcxs</td>
<td></td>
<td></td>
<td>qvlfcxda</td>
<td></td>
</tr>
<tr>
<td>qvlfcxsx</td>
<td></td>
<td></td>
<td>qvlfcxda</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>QRT</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>X</td>
</tr>
</tbody>
</table>

- if RA = 0 then b ← 0
- else b ← (RA)
- EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFFFF8
- 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

<table>
<thead>
<tr>
<th></th>
<th>QRT, RA, RB</th>
<th>(X=0)</th>
<th>QRT, RA, RB</th>
<th>(X=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvlfodx</td>
<td></td>
<td></td>
<td>qvlfoxda</td>
<td></td>
</tr>
<tr>
<td>qvlfoxda</td>
<td></td>
<td></td>
<td>qvlfoxda</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>QRT</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>71</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>6</td>
<td>X</td>
</tr>
</tbody>
</table>

- if RA = 0 then b ← 0
- else b ← (RA)
- EA ← (b + (RB)) & 0xFFFFFFFFFFFFFFFFF0
- MVAL ← MEM(EA, 16)
- QRT[0] ← MVAL[0]
- QRT[2] ← MVAL[0]

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

<table>
<thead>
<tr>
<th>qvlfcsux</th>
<th>QRT, RA, RB</th>
<th>(X=0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvlfcsuxa</td>
<td>QRT, RA, RB, RB</td>
<td>(X=1)</td>
</tr>
</tbody>
</table>

- EA ← ((RA) + (RB)) & 0xFFFFFFFF0
- MVAL ← MEM(EA, 8)
- QRT<sub>0</sub> ← DOUBLE(MVAL<sub>0:31</sub>)
- QRT<sub>1</sub> ← DOUBLE(MVAL<sub>32:63</sub>)

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

<table>
<thead>
<tr>
<th>qvlfcdux</th>
<th>QRT, RA, RB</th>
<th>(X=0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvlfcduxa</td>
<td>QRT, RA, RB, RB</td>
<td>(X=1)</td>
</tr>
</tbody>
</table>

- EA ← ((RA) + (RB)) & 0xFFFFFFFF0
- MVAL ← MEM(EA, 16)
- QRT<sub>0</sub> ← MVAL<sub>0</sub>
- QRT<sub>1</sub> ← MVAL<sub>1</sub>
- QRT<sub>2</sub> ← MVAL<sub>0</sub>
- QRT<sub>3</sub> ← MVAL<sub>1</sub>

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

\[
\text{qvlfiwax} \quad \text{QRT}, \text{RA}, \text{RB} \quad (X=0)
\]
\[
\text{qvlfiwaxa} \quad \text{QRT}, \text{RA}, \text{RB} \quad (X=1)
\]

\[
\begin{array}{cccccc}
\text{QRT} & \text{RA} & \text{RB} & \text{X} \\
0 & q_{31} & 0 & 6 & 11 & 16 & 21 & 31 \\
1 & q_{31} & 0 & 6 & 11 & 16 & 21 & 31 \\
\end{array}
\]

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

\[
\text{qvlfiwzx} \quad \text{QRT}, \text{RA}, \text{RB} \quad (X=0)
\]
\[
\text{qvlfiwzxa} \quad \text{QRT}, \text{RA}, \text{RB} \quad (X=1)
\]

\[
\begin{array}{cccccc}
\text{QRT} & \text{RA} & \text{RB} & \text{X} \\
0 & q_{31} & 0 & 6 & 11 & 16 & 21 & 31 \\
1 & q_{31} & 0 & 6 & 11 & 16 & 21 & 31 \\
\end{array}
\]

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

<table>
<thead>
<tr>
<th>QRT</th>
<th>RA</th>
<th>RB</th>
<th>582</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

if RA = 0 then b ← 0
else b ← (RA)

EA ← b + (RB)

AA = EA & 0b11000

QRT^0 ← 0x400 || (AA ) |58:60| 49
QRT^1 ← 0x400 || (AA+8) |58:60| 49
QRT^2 ← 0x400 || (AA+16) |58:60| 49
QRT^3 ← 0x400 || (AA+24) |58:60| 49

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 qvpfmt 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
qvpfmt qaligned, qmem1, qmem2, qalign

Quad-Vector Load Permute Control Left
Single indexed X-form

qvlpclsx QRT, RA, RB

<table>
<thead>
<tr>
<th>QRT</th>
<th>RA</th>
<th>RB</th>
<th>518</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>31</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

if RA = 0 then b ← 0
else b ← (RA)

EA ← b + (RB)

AA = (EA * 2) & 0b11000

QRT^0 ← 0x400 || (AA ) |58:60| 49
QRT^1 ← 0x400 || (AA+8) |58:60| 49
QRT^2 ← 0x400 || (AA+16) |58:60| 49
QRT^3 ← 0x400 || (AA+24) |58:60| 49

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 qvpfmt 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
qvpfmt qaligned, qmem1, qmem2, qalign
Quad-Vector Load Permute Control Right
Double indexed X-form

\[
\text{qvlpcrdx} \quad \text{QRT,RA,RB}
\]

<table>
<thead>
<tr>
<th>31</th>
<th>QRT</th>
<th>RA</th>
<th>RB</th>
<th>70</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>16</td>
<td>21</td>
<td>70</td>
<td>/</td>
</tr>
</tbody>
</table>

if RA = 0 then b ← 0
else b ← (RA)

EA ← b + (RB)
AA = (32 - (EA & 0b11000))
QRT0 ← 0x400 | (AA) 58:60 | 49
QRT1 ← 0x400 | (AA+8) 58:60 | 49
QRT2 ← 0x400 | (AA+16) 58:60 | 49
QRT3 ← 0x400 | (AA+24) 58:60 | 49

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 permut instruction \text{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

\[
\text{qvlpcrsx} \quad \text{QRT,RA,RB}
\]

<table>
<thead>
<tr>
<th>31</th>
<th>QRT</th>
<th>RA</th>
<th>RB</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>16</td>
<td>21</td>
<td>6</td>
</tr>
</tbody>
</table>

if RA = 0 then b ← 0
else b ← (RA)

EA ← b + (RB)
AA = (32 - ((EA * 2) & 0b11000))
QRT0 ← 0x400 | (AA) 58:60 | 49
QRT1 ← 0x400 | (AA+8) 58:60 | 49
QRT2 ← 0x400 | (AA+16) 58:60 | 49
QRT3 ← 0x400 | (AA+24) 58:60 | 49

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 permut instruction \text{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 X-form

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
<th>X-form</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvstfsx</td>
<td>QRS,RA,RB</td>
<td>(X=0)</td>
</tr>
<tr>
<td>qvstfsxa</td>
<td>QRS,RA,RB</td>
<td>(X=1)</td>
</tr>
</tbody>
</table>

#### qvstfsx

- 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, 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 X-form

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
<th>X-form</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvstfsux</td>
<td>QRS,RA,RB</td>
<td>(X=0)</td>
</tr>
<tr>
<td>qvstfsuxa</td>
<td>QRS,RA,RB</td>
<td>(X=1)</td>
</tr>
</tbody>
</table>

#### qvstfsux

- 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, 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)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>645</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>677</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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]
QPX Architecture

Quad-Vector STore Floating-point Double indXed X-form
qvstfdx QRS,RA,RB (X=0)
qvstfdxa QRS,RA,RB (X=1)

<table>
<thead>
<tr>
<th>31</th>
<th>8</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>711</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QRS</td>
<td>RA</td>
<td>RB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>31</th>
<th>8</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>743</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QRS</td>
<td>RA</td>
<td>RB</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 X-form

\( \text{qvstfdxi} \quad \text{QRS,RA,RB} \quad (X=0) \)
\( \text{qvstfdxia} \quad \text{QRS,RA,RB} \quad (X=1) \)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>709</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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

if (SNEE = 1) then
  if (isNaN (QRS) OR isNaN (QRS) OR isNaN (QRS) OR isNaN (QRS)) then
    ESR[AP] ← 1
if (SIEE = 1) then
  if (isInf (QRS) OR isInf (QRS) OR isInf (QRS) OR isInf (QRS)) 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

\( \text{qvstfduxi} \quad \text{QRS,RA,RB} \quad (X=0) \)
\( \text{qvstfduxia} \quad \text{QRS,RA,RB} \quad (X=1) \)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>741</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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

if (SNEE = 1) then
  if (isNaN (QRS) OR isNaN (QRS) OR isNaN (QRS) OR isNaN (QRS)) then
    ESR[AP] ← 1
if (SIEE = 1) then
  if (isInf (QRS) OR isInf (QRS) OR isInf (QRS) OR isInf (QRS)) 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]
QPX Architecture

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

\[
\text{qvstfcsx} \quad \text{QRS,RA,RB} \\
\text{qvstfcsxa} \quad \text{QRS,RA,RB}
\]

(X=0)  (X=1)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>135</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>135</td>
</tr>
</tbody>
</table>

If \( RA = 0 \) then \( b \leftarrow 0 \)  
else \( b \leftarrow (RA) \)  
\( EA \leftarrow (b + (RB)) \& 0xFFFFFFFFFFFFFFF8 \)  
\( \text{MEM}(EA, 8) \leftarrow \text{SINGLE}(\text{QRS}^0) \vert \vert \text{SINGLE}(\text{QRS}^1) \)

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 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**

\[
\text{qvstfcdx} \quad \text{QRS,RA,RB} \\
\text{qvstfcdxa} \quad \text{QRS,RA,RB}
\]

(X=0)  (X=1)

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>199</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>199</td>
</tr>
</tbody>
</table>

If \( RA = 0 \) then \( b \leftarrow 0 \)  
else \( b \leftarrow (RA) \)  
\( EA \leftarrow (b + (RB)) \& 0xFFFFFFFFFFFFFFFO \)  
\( \text{MEM}(EA, 16) \leftarrow \text{QR}^0 \vert \vert \text{QR}^1 \)

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 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

$qvstfcsi_x$ QRS,RA,RB  
$qvstfcsx_{ia}$ QRS,RA,RB  

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>133</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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

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

if (SIEE = 1) then
  if (isInf(QRS) OR
       isInf(QRS)) 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.

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  
$qvstfcdxia$ QRS,RA,RB  

<table>
<thead>
<tr>
<th>31</th>
<th>QRS</th>
<th>RA</th>
<th>RB</th>
<th>197</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

if RA = 0 then b ← 0
else            b ← (RA)
EA ← (b + (RB)) & 0FFFFFFFFFFFFFFFF0
MEM(EA, 16) ← QRS  ||  QRS

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

if (SIEE = 1) then
  if (isInf(QRS) OR
       isInf(QRS)) 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.

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]
**QPX Architecture**

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

qvstfcux $QRS,RA,RB$ \( (X=0) \)
qvstfcuxa $QRS,RA,RB$ \( (X=1) \)

$\begin{array}{cccccc}
0 & 31 & 6 & 11 & 16 & 21 & 167 & X \\
\hline
& & & & & & & \\
\end{array}$

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 $QRS,RA,RB$ \( (X=0) \)
qvstfcuxa $QRS,RA,RB$ \( (X=1) \)

$\begin{array}{cccccc}
0 & 31 & 6 & 11 & 16 & 21 & 231 & X \\
\hline
& & & & & & & \\
\end{array}$

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 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 X-form

\[ \text{qvstfcuxi} \quad \text{QRS,RA,RB} \quad (X=0) \]
\[ \text{qvstfcuxia} \quad \text{QRS,RA,RB} \quad (X=1) \]

\[
\begin{array}{cccccccc}
31 & 6 & 11 & 16 & 21 & 185 & X \\
0 & \text{QRS} & \text{RA} & \text{RB} & & & \\
\end{array}
\]

\[ \text{EA} \leftarrow \text{(RA) + (RB)} \& \ 0xFFFFFFFFFFFFFFF8 \]
\[ \text{MEM}(\text{EA}, 8) \leftarrow \text{SINGLE(QRS}^0) || \text{SINGLE(QRS}^1) \]
\[ \text{RA} \leftarrow \text{EA} \]

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

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

\[ \text{qvstfcdux} \quad \text{QRS,RA,RB} \quad (X=0) \]
\[ \text{qvstfcduxia} \quad \text{QRS,RA,RB} \quad (X=1) \]

\[
\begin{array}{cccccccc}
31 & 6 & 11 & 16 & 21 & 229 & X \\
0 & \text{QRS} & \text{RA} & \text{RB} & & & \\
\end{array}
\]

\[ \text{EA} \leftarrow \text{(RA) + (RB)} \& \ 0xFFFFFFFFFFF0 \]
\[ \text{MEM}(\text{EA}, 16) \leftarrow \text{QRS}^0 || \text{QRS}^1 \]
\[ \text{RA} \leftarrow \text{EA} \]

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

if (SIEE = 1) then
  if (isInf (QRS\(^0\)) OR isInf (QRS\(^1\))) then
    ESR[AP] \leftarrow 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 indXed X-form

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

if RA = 0 then b ← 0
else           b ← (RA)
EA ← (b + (RB)) & 0xFFFFFFFFFFFF0

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
```

<table>
<thead>
<tr>
<th></th>
<th>QRT</th>
<th>/</th>
<th>QRB</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

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

**Special Registers Altered:**
None

**Quad-Vector Floating-point NEGateway**

**X-form**

```
qvfneg  QRT, QRB
```

<table>
<thead>
<tr>
<th></th>
<th>QRT</th>
<th>/</th>
<th>QRB</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

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
```

<table>
<thead>
<tr>
<th></th>
<th>QRT</th>
<th>/</th>
<th>QRB</th>
<th>254</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

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**

```
qvfabsn  QRT, QRB
```

<table>
<thead>
<tr>
<th></th>
<th>QRT</th>
<th>/</th>
<th>QRB</th>
<th>136</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>11</td>
<td>16</td>
<td>21</td>
</tr>
</tbody>
</table>

For each vector element, the contents of register QRB, with bit 0 set to one, 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

\[
\begin{array}{cccccccc}
\text{qvfadd} & \text{QRT, QRA, QRB} \\
0 & 4 & 6 & 11 & 16 & 21 & \| & 26 & 31 \\
\end{array}
\]

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

\[
\begin{array}{cccccccc}
\text{qvfsub} & \text{QRT, QRA, QRB} \\
0 & 4 & 6 & 11 & 16 & 21 & \| & 26 & 31 \\
\end{array}
\]

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

```
0 6 11 16 21 26 31
```
qvfmuls QRT,QRA,QRC

```
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

```
0 6 11 16 21 24 31
```
qvfres QRT,QRB

```
0 6 11 16 21 24 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.,

\[
\text{ABS}(\text{estimate} - \frac{1}{x}) \leq \frac{1}{16384}
\]

where \( x \) is the initial value in QRB.

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

<table>
<thead>
<tr>
<th>Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-\infty)</td>
<td>(-\infty)</td>
</tr>
<tr>
<td>(-0)</td>
<td>(-0)</td>
</tr>
<tr>
<td>(+0)</td>
<td>(+\infty)</td>
</tr>
<tr>
<td>(+\infty)</td>
<td>(+0)</td>
</tr>
<tr>
<td>SNaN</td>
<td>QNaN</td>
</tr>
<tr>
<td>QNaN</td>
<td>QNaN</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>QRT</th>
<th>///</th>
<th>16</th>
<th>21</th>
<th>26</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

qvfrsqrtes QRT,QRB

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>QRT</th>
<th>///</th>
<th>16</th>
<th>21</th>
<th>26</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.,

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

where x is the initial value in QRB.

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

<table>
<thead>
<tr>
<th>Operand</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-\infty$</td>
<td>QNaN</td>
</tr>
<tr>
<td>&lt; 0</td>
<td>QNaN</td>
</tr>
<tr>
<td>0</td>
<td>$-\infty$</td>
</tr>
<tr>
<td>+0</td>
<td>$+\infty$</td>
</tr>
<tr>
<td>$+\infty$</td>
<td>+0</td>
</tr>
<tr>
<td>SNaN</td>
<td>QNaN</td>
</tr>
<tr>
<td>QNaN</td>
<td>QNaN</td>
</tr>
</tbody>
</table>

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

| 0 | 5 | 11 | 16 | 21 | 26 | 29 | / |

qvfmadd s QRT,QRA,QRC,QRB

| 0 | 5 | 11 | 16 | 21 | 26 | 29 | / |

The operations

QRT<sub>0</sub> ← [(QRA<sub>0</sub>)×(QRC<sub>0</sub])] + (QRB<sub>0</sub>)
QRT<sub>1</sub> ← [(QRA<sub>1</sub>)×(QRC<sub>1</sub])] + (QRB<sub>1</sub>)
QRT<sub>2</sub> ← [(QRA<sub>2</sub>)×(QRC<sub>2</sub])] + (QRB<sub>2</sub>)
QRT<sub>3</sub> ← [(QRA<sub>3</sub>)×(QRC<sub>3</sub])] + (QRB<sub>3</sub>)

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

| 0 | 5 | 11 | 16 | 21 | 26 | 28 | / |

qvfmsubs QRT,QRA,QRC,QRB

| 0 | 5 | 11 | 16 | 21 | 26 | 28 | / |

The operations

QRT<sub>0</sub> ← [(QRA<sub>0</sub>)×(QRC<sub>0</sub])] − (QRB<sub>0</sub>)
QRT<sub>1</sub> ← [(QRA<sub>1</sub>)×(QRC<sub>1</sub])] − (QRB<sub>1</sub>)
QRT<sub>2</sub> ← [(QRA<sub>2</sub>)×(QRC<sub>2</sub])] − (QRB<sub>2</sub>)
QRT<sub>3</sub> ← [(QRA<sub>3</sub>)×(QRC<sub>3</sub])] − (QRB<sub>3</sub>)

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 | /
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
</tr>
</tbody>
</table>

qvfnmadds QRT,QRA,QRC,QRB

| 0 | QRT | QRA | QRB | QRC | 31 | /
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
</tr>
</tbody>
</table>

The operations

\[
\begin{align*}
QRT_0 & \leftarrow - \left( \left[ (QRA_0) \times (QRC_0) \right] + (QRB_0) \right) \\
QRT_1 & \leftarrow - \left( \left[ (QRA_1) \times (QRC_1) \right] + (QRB_1) \right) \\
QRT_2 & \leftarrow - \left( \left[ (QRA_2) \times (QRC_2) \right] + (QRB_2) \right) \\
QRT_3 & \leftarrow - \left( \left[ (QRA_3) \times (QRC_3) \right] + (QRB_3) \right)
\end{align*}
\]

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 `qvfnmadd` 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 | /
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
</tr>
</tbody>
</table>

qvfnmsubs QRT,QRA,QRC,QRB

| 0 | QRT | QRA | QRB | QRC | 30 | /
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
</tr>
</tbody>
</table>

The operations

\[
\begin{align*}
QRT_0 & \leftarrow - \left( \left[ (QRA_0) \times (QRC_0) \right] - (QRB_0) \right) \\
QRT_1 & \leftarrow - \left( \left[ (QRA_1) \times (QRC_1) \right] - (QRB_1) \right) \\
QRT_2 & \leftarrow - \left( \left[ (QRA_2) \times (QRC_2) \right] - (QRB_2) \right) \\
QRT_3 & \leftarrow - \left( \left[ (QRA_3) \times (QRC_3) \right] - (QRB_3) \right)
\end{align*}
\]

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 `qvfnmsub` 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  

\[ \text{qvxmadds} \quad \text{QRT, QRA, QRC, QRB} \]

0 6 11 16 21 26 31
0 6 11 16 21 26 31

The operations

\[ QRT^4 \leftarrow (QRA^0 \times QRC^0) + (QRB^0) \]
\[ QRT^1 \leftarrow (QRA^0 \times QRC^1) + (QRB^1) \]
\[ QRT^2 \leftarrow (QRA^2 \times QRC^2) + (QRB^2) \]
\[ QRT^3 \leftarrow (QRA^3 \times QRC^3) + (QRB^3) \]

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 qvfxnppmadd.

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

A-form

\[ \text{qvfxnppmadds} \quad \text{QRT, QRA, QRC, QRB} \]

0 6 11 16 21 26 31
0 6 11 16 21 26 31

The operations

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

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.

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

\[ qvfxxcpnadd \quad QRT, QRA, QRC, QRB \]

| 0 | 4 | QRT 11 16 21 26 | 3 |
|---|---|----------------|

\[ qvfxxcpnaddsd \quad QRT, QRA, QRC, QRB \]

<table>
<thead>
<tr>
<th>0</th>
<th>6</th>
<th>QRT 11 16 21 26</th>
</tr>
</thead>
</table>

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 cross (X) Multiply [Single] A-form

qvfxmul QRT,QRA,QRC

<table>
<thead>
<tr>
<th>4</th>
<th>QRT</th>
<th>QRA</th>
<th>QRC</th>
<th>17</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
</tr>
</tbody>
</table>

qvfxmuls QRT,QRA,QRC

<table>
<thead>
<tr>
<th>0</th>
<th>QRT</th>
<th>QRA</th>
<th>QRC</th>
<th>17</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>26</td>
<td>31</td>
</tr>
</tbody>
</table>

The operations

QRT<sup>0</sup> ← (QRA<sup>0</sup>) × (QRC<sup>0</sup>)
QRT<sup>1</sup> ← (QRA<sup>0</sup>) × (QRC<sup>1</sup>)
QRT<sup>2</sup> ← (QRA<sup>2</sup>) × (QRC<sup>2</sup>)
QRT<sup>3</sup> ← (QRA<sup>2</sup>) × (QRC<sup>3</sup>)

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*

\[ \text{qvfrsp} \quad \text{QRT,QRB} \]

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>31</th>
</tr>
</thead>
</table>

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

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

<table>
<thead>
<tr>
<th>qvcltid</th>
<th>QRT, QRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>QRT /// QRB 814 /</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>qvcltdu</th>
<th>QRT, QRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>QRT /// QRB 942 /</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>qvcltdz</th>
<th>QRT, QRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>QRT /// QRB 815 /</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>qvcltdzu</th>
<th>QRT, QRB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>QRT /// QRB 943 /</td>
</tr>
</tbody>
</table>

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

\[
\text{qvfctiw} \quad \text{QRT},\text{QRB}
\]

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>QRT</th>
<th>///</th>
<th>16</th>
<th>21</th>
<th>/</th>
</tr>
</thead>
</table>

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\(_{0:31}\) ← 0x7FF80000

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

\[
\text{qvfctiwu} \quad \text{QRT},\text{QRB}
\]

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>QRT</th>
<th>///</th>
<th>16</th>
<th>21</th>
<th>142</th>
<th>/</th>
</tr>
</thead>
</table>

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\(_{0:31}\) ← 0x7FF80000
Quad-Vector Floating-point Convert To Integer Word with round toward Zero

X-form

$qvftiwz\ \ QRT,QRB$

<table>
<thead>
<tr>
<th>4</th>
<th>QRT</th>
<th>11 //</th>
<th>QRB</th>
<th>15</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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

$qvftiwuz\ \ QRT,QRB$

<table>
<thead>
<tr>
<th>4</th>
<th>QRT</th>
<th>11 //</th>
<th>QRB</th>
<th>143</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

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, $QRT_{0:31} \leftarrow 0x7FF80000$
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

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

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
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 | 5 | 11 | 16 | 21 | 392 | / |

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 | 5 | 11 | 16 | 21 | 456 | / |

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 Minus X-form**

qvfrim QRT,QRB

| 4 | 5 | 11 | 16 | 21 | 488 | / |

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

```plaintext
qvftstnan QRT, QRA, QRB
```

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>64</th>
</tr>
</thead>
</table>
| if isNaN(QRA) OR isNaN(QRB)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if isNaN(QRA) OR isNaN(QRB)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if isNaN(QRA) OR isNaN(QRB)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if isNaN(QRA) OR isNaN(QRB)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 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

```plaintext
qvfcmpgt QRT, QRA, QRB
```

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>32</th>
</tr>
</thead>
</table>
| if (QRA^2) > (QRB^2)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if (QRA^2) > (QRB^2)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if (QRA^2) > (QRB^2)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 0xBFF0_0000_0000_0000
| if (QRA^2) > (QRB^2)
  then QRT ← 0x3FF0_0000_0000_0000
  else QRT ← 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**

\[
qvfcmplt \quad QRT,QRA,QRB
\]

<table>
<thead>
<tr>
<th>4</th>
<th>96</th>
<th>0</th>
<th>6</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

if \[(QRA)^0 < (QRB)^0\]
then \[QRT^0 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^0 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^1 < (QRB)^1\]
then \[QRT^1 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^1 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^2 < (QRB)^2\]
then \[QRT^2 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^2 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^3 < (QRB)^3\]
then \[QRT^3 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^3 \leftarrow 0xBFF0\_0000\_0000\_0000\]

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**

\[
qvfcmpeq \quad QRT,QRA,QRB
\]

<table>
<thead>
<tr>
<th>4</th>
<th>96</th>
<th>0</th>
<th>6</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>6</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>31</td>
</tr>
</tbody>
</table>

if \[(QRA)^0 = (QRB)^0\]
then \[QRT^0 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^0 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^1 = (QRB)^1\]
then \[QRT^1 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^1 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^2 = (QRB)^2\]
then \[QRT^2 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^2 \leftarrow 0xBFF0\_0000\_0000\_0000\]
if \[(QRA)^3 = (QRB)^3\]
then \[QRT^3 \leftarrow 0x3FF0\_0000\_0000\_0000\]
else \[QRT^3 \leftarrow 0xBFF0\_0000\_0000\_0000\]

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
```

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

$$qvalignisi, QRA, QRB, VD$$

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>23</th>
<th>5</th>
<th>/</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QRT</td>
<td>QRA</td>
<td>QRB</td>
<td>VD</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if $VD = 00$ then  
$QRT \leftarrow (QRA)$

else if $VD = 01$ then  
$QRT \leftarrow (QRA^1) \|| (QRA^2) \|| (QRA^3) \|| (QRB^0)$

else if $VD = 10$ then  
$QRT \leftarrow (QRA^2) \|| (QRA^3) \|| (QRB^0) \|| (QRB^1)$

else if $VD = 11$ then  
$QRT \leftarrow (QRA^3) \|| (QRB^0) \|| (QRB^1) \|| (QRB^2)$

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

$$qvperm, QRT, QRA, QRB, QRC$$

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
<th>11</th>
<th>16</th>
<th>21</th>
<th>26</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>QRT</td>
<td>QRA</td>
<td>QRB</td>
<td>QRC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For each vector element,  
if $QRC_{1:11} = 0x400$ then  
case $QRC_{12:14}$:

- $QRT \leftarrow (QRA^0)$ when 000
- $QRT \leftarrow (QRA^1)$ when 001
- $QRT \leftarrow (QRA^2)$ when 010
- $QRT \leftarrow (QRA^3)$ when 011
- $QRT \leftarrow (QRB^0)$ when 100
- $QRT \leftarrow (QRB^1)$ when 101
- $QRT \leftarrow (QRB^2)$ when 110
- $QRT \leftarrow (QRB^3)$ when 111

else $QRT \leftarrow$ 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

$\text{QRT}, \text{QRA}, \text{VD}$

if $\text{VD} = 00$ then
\[ \text{QRT} \leftarrow (\text{QRA}^{0}) || (\text{QRA}^{0}) || (\text{QRA}^{0}) || (\text{QRA}^{0}) \]
else if $\text{VD} = 01$ then
\[ \text{QRT} \leftarrow (\text{QRA}^{1}) || (\text{QRA}^{1}) || (\text{QRA}^{1}) || (\text{QRA}^{1}) \]
else if $\text{VD} = 10$ then
\[ \text{QRT} \leftarrow (\text{QRA}^{2}) || (\text{QRA}^{2}) || (\text{QRA}^{2}) || (\text{QRA}^{2}) \]
else if $\text{VD} = 11$ then
\[ \text{QRT} \leftarrow (\text{QRA}^{3}) || (\text{QRA}^{3}) || (\text{QRA}^{3}) || (\text{QRA}^{3}) \]

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

$\text{QRT}, \text{GPC}$

\[ \text{QRT}^0 \leftarrow 0x400 || \text{GPC}^0:2 \]
\[ \text{QRT}^1 \leftarrow 0x400 || \text{GPC}^3:5 \]
\[ \text{QRT}^2 \leftarrow 0x400 || \text{GPC}^6:8 \]
\[ \text{QRT}^3 \leftarrow 0x400 || \text{GPC}^9:11 \]

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

<table>
<thead>
<tr>
<th></th>
<th>QRT</th>
<th>QRA</th>
<th>QRB</th>
<th>TT</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>9</td>
<td>11</td>
<td>16</td>
<td>21</td>
<td>25</td>
<td>31</td>
</tr>
</tbody>
</table>

For each vector element,

if [(QRA) < 0.0 OR isnan(QRA)] AND
   [(QRB) < 0.0 OR isnan(QRB)] then
   if TT = 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 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 = 1 then QRT ← 0x3FF0_0000_0000_0000
     else QRT ← 0xBFF0_0000_0000_0000
if [(QRA) ≥ 0.0] AND
   [(QRB) ≥ 0.0] then
   if TT = 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.

<table>
<thead>
<tr>
<th>Extended Mnemonic</th>
<th>Equivalent</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>qvfclrc QRT</td>
<td>qvfllogical QRT,QRT,QRT,0</td>
<td>clear (set as FALSE)</td>
</tr>
<tr>
<td>qvfandc QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRB,4</td>
<td>and complement B</td>
</tr>
<tr>
<td>qvfand QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRB,1</td>
<td>and</td>
</tr>
<tr>
<td>qvfcomp QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRB,8</td>
<td>and complement B</td>
</tr>
<tr>
<td>qvfctfa QRT,QRA</td>
<td>qvfllogical QRT,QRA,QRA,5</td>
<td>convert to float-boolean A</td>
</tr>
<tr>
<td>qvfnotc QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRB,13</td>
<td>or complement B</td>
</tr>
<tr>
<td>qvfset QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRB,15</td>
<td>or complement B</td>
</tr>
<tr>
<td>qvfandb QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,6</td>
<td>xor</td>
</tr>
<tr>
<td>qvfnor QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,7</td>
<td>or</td>
</tr>
<tr>
<td>qvfnor QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,9</td>
<td>or complement B</td>
</tr>
<tr>
<td>qvfequ QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,10</td>
<td>Boolean equivalent (XNOR)</td>
</tr>
<tr>
<td>qvfandb QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,11</td>
<td>or complement B</td>
</tr>
<tr>
<td>qvfset QRT,QRA,QRB</td>
<td>qvfllogical QRT,QRA,QRA,14</td>
<td>or complement B</td>
</tr>
<tr>
<td>qvfset QRT</td>
<td>qvfllogical QRT,QRT,QRT,15</td>
<td>set (set as TRUE)</td>
</tr>
</tbody>
</table>