



---

**l.add**

**Add Signed**

**l.add**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x0	opcode 0x0
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4 bits	4 bits

**Format:**

`l.add rD,rA,rB`

**Description:**

The contents of general-purpose register rA are added to the contents of general-purpose register rB to form the result. The result is placed into general-purpose register rD.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] + rB[31:0]
SR[CY] <- carry
SR[OV] <- overflow
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] + rB[63:0]
SR[CY] <- carry
SR[OV] <- overflow
```

**Exceptions:**

Range Exception

Instruction Class	Implementation
ORBIS32 I	Required

**l.addc**

## Add Signed and Carry

**l.addc**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x1	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

### Format:

```
l.addc rD,rA,rB
```

### Description:

The contents of general-purpose register rA are added to the contents of general-purpose register rB and carry SR[CY] to form the result. The result is placed into general-purpose register rD.

### 32-bit Implementation:

```
rD[31:0] <- rA[31:0] + rB[31:0] + SR[CY]  
SR[CY] <- carry  
SR[OV] <- overflow
```

### 64-bit Implementation:

```
rD[63:0] <- rA[63:0] + rB[63:0] + SR[CY]  
SR[CY] <- carry  
SR[OV] <- overflow
```

### Exceptions:

Range Exception

Instruction Class	Implementation
ORBIS32 I	Required



---

**l.and**

**And**

**l.and**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x3	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

l.and rD,rA,rB

**Description:**

The contents of general-purpose register rA are combined with the contents of general-purpose register rB in a bit-wise logical AND operation. The result is placed into general-purpose register rD.

**32-bit Implementation:**

rD[31:0] <- rA[31:0] AND rB[31:0]

**64-bit Implementation:**

rD[63:0] <- rA[63:0] AND rB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.andi**

**And with Immediate Half Word**

**l.andi**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x29	D	A	K
6 bits	5 bits	5 bits	16bits

**Format:**

`l.andi rD,rA,K`

**Description:**

The immediate value is zero-extended and combined with the contents of general-purpose register rA in a bit-wise logical AND operation. The result is placed into general-purpose register rD.

**32-bit Implementation:**

`rD[31:0] <- rA[31:0] AND extz(Immediate)`

**64-bit Implementation:**

`rD[63:0] <- rA[63:0] AND extz(Immediate)`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.bf**

**Branch if Flag**

**l.bf**

31 . . . . . 26	25 . . . . . 0
opcode 0x4	N
6 bits	26bits

**Format:**

l.bf N

**Description:**

The immediate value is shifted left two bits, sign-extended to program counter width and then added to the address of the delay slot. The result is the effective address of the branch. If the flag is set, then the program branches to EA with a delay of one instruction.

**32-bit Implementation:**

EA <- (Immediate << 2) + DelayInsnAddr  
PC <- EA if SR[F] set

**64-bit Implementation:**

EA <- (Immediate << 2) + DelayInsnAddr  
PC <- EA if SR[F] set

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

l.bnf

## Branch if No Flag

l.bnf

31 . . . . . 26	25 . . . . . 0
opcode 0x3	N
6 bits	26bits

### Format:

l.bnf N

### Description:

The immediate value is shifted left two bits, sign-extended to program counter width and then added to the address of the delay slot. The result is the effective address of the branch. If the flag is cleared, then the program branches to EA with a delay of one instruction.

### 32-bit Implementation:

EA <- (Immediate << 2) + DelayInsnAddr  
 PC <- EA if SR[F] cleared

### 64-bit Implementation:

EA <- (Immediate << 2) + DelayInsnAddr  
 PC <- EA if SR[F] cleared

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required



**l.cmov**

**Conditional Move**

**l.cmov**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0xe	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.cmov rD,rA,rB`

**Description:**

If SR[F] is set, general-purpose register rA is placed in general-purpose register rD. If SR[F] is cleared, general-purpose register rB is placed in general-purpose register rD.

**32-bit Implementation:**

`rD[31:0] <- SR[F] ? rA[31:0] : rB[31:0]`

**64-bit Implementation:**

`rD[63:0] <- SR[F] ? rA[63:0] : rB[63:0]`

**Exceptions:**

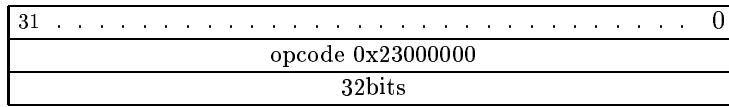
None

Instruction Class	Implementation
ORBIS32 II	Optional

**l.csync**

**Context Synchronization**

**l.csync**



**Format:**

l.csync

**Description:**

Execution of context synchronization instruction results in completion of all operations inside the processor and a flush of the instruction pipelines. When all operations are complete, The RISC core resumes with an empty instruction pipeline and fresh context in all units (MMU for example).

**32-bit Implementation:**

context-synchronization

**64-bit Implementation:**

context-synchronization

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust1

**Reserved for ORBIS32/64 Custom Instructions**

l.cust1

31 . . . . . 26	25 . . . . . 0
opcode 0x1c	reserved
6 bits	26bits

**Format:**

l.cust1

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust2

Reserved for ORBIS32/64 Custom Instructions

l.cust2

31 . . . . . 26	25 . . . . . 0
opcode 0x1d	reserved
6 bits	26bits

**Format:**

l.cust2

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust3

Reserved for ORBIS32/64 Custom Instructions

l.cust3

31 . . . . . 26	25 . . . . . 0
opcode 0x1e	reserved
6 bits	26bits

**Format:**

l.cust3

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust4

**Reserved for ORBIS32/64 Custom Instructions**

l.cust4

31 . . . . . 26	25 . . . . . 0
opcode 0x1f	reserved
6 bits	26bits

**Format:**

l.cust4

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust5

Reserved for ORBIS32/64 Custom Instructions

l.cust5

31 . . . . . 26	25 . . . . . 0
opcode 0x3c	reserved
6 bits	26bits

**Format:**

l.cust5

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust6

Reserved for ORBIS32/64 Custom Instructions

l.cust6

31 . . . . . 26	25 . . . . . 0
opcode 0x3d	reserved
6 bits	26bits

**Format:**

l.cust6

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional



l.cust7

Reserved for ORBIS32/64 Custom Instructions

l.cust7

31 . . . . . 26	25 . . . . . 0
opcode 0x3e	reserved
6 bits	26bits

**Format:**

l.cust7

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

l.cust8

Reserved for ORBIS32/64 Custom Instructions

l.cust8

31 . . . . . 26	25 . . . . . 0
opcode 0x3f	reserved
6 bits	26bits

**Format:**

l.cust8

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but rather by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORBIS32 II	Optional

**l.div**

**Divide Signed**

**l.div**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x9	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.div rD,rA,rB`

**Description:**

The content of general-purpose register rA are divided by the content of general-purpose register rB and the result is placed into general-purpose register rD. Both operands are treated as signed integers. A carry flag is set when the divisor is zero.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] / rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] / rB[63:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

**l.divu**

**Divide Unsigned**

**l.divu**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0xa	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.divu rD,rA,rB`

**Description:**

The content of general-purpose register rA are divided by the content of general-purpose register rA and the result is placed into general-purpose register rD. Both operands are treated as unsigned integers. A carry flag is set when the divisor is zero.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] / rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] / rB[63:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.extbs

### Extend Byte with Sign

l.extbs

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x1	reserved	opcode 0xc
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

#### Format:

l.extbs rD,rA,rB

#### Description:

Bit 7 of general-purpose register rA is placed in high-order bits of general-purpose register rD. The low-order eight bits of general-purpose register rA are copied into the low-order eight bits of general-purpose register rD.

#### 32-bit Implementation:

rD[31:8] <- rA[7]  
rD[7:0] <- rA[7:0]

#### 64-bit Implementation:

rD[63:8] <- rA[7]  
rD[7:0] <- rA[7:0]

#### Exceptions:

None

Instruction Class	Implementation
ORBIS32 II	Optional

**l.extbz**

**Extend Byte with Zero**

**l.extbz**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0xc
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.extbz rD,rA,rB`

**Description:**

Zero is placed in high-order bits of general-purpose register rD. The low-order eight bits of general-purpose register rA are copied into the low-order eight bits of general-purpose register rD.

**32-bit Implementation:**

`rD[31:8] <- 0`  
`rD[7:0] <- rA[7:0]`

**64-bit Implementation:**

`rD[63:8] <- 0`  
`rD[7:0] <- rA[7:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.exths

### Extend Half Word with Sign

l.exths

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0xc
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

#### Format:

l.exths rD,rA,rB

#### Description:

Bit 15 of general-purpose register rA is placed in high-order bits of general-purpose register rD. The low-order 16 bits of general-purpose register rA are copied into the low-order 16 bits of general-purpose register rD.

#### 32-bit Implementation:

rD[31:16] <- rA[15]  
rD[15:0] <- rA[15:0]

#### 64-bit Implementation:

rD[63:16] <- rA[15]  
rD[15:0] <- rA[15:0]

#### Exceptions:

None

Instruction Class	Implementation
ORBIS32 II	Optional

---

**l.exthz**

**Extend Half Word with Zero**

**l.exthz**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x2	reserved	opcode 0xc
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.exthz rD,rA,rB`

**Description:**

Zero is placed in high-order bits of general-purpose register rD. The low-order 16 bits of general-purpose register rA are copied into the low-order 16 bits of general-purpose register rD.

**32-bit Implementation:**

`rD[31:16] <- 0`  
`rD[15:0] <- rA[15:0]`

**64-bit Implementation:**

`rD[63:16] <- 0`  
`rD[15:0] <- rA[15:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional



l.extws

### Extend Word with Sign

l.extws

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0xd
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

#### Format:

l.extws rD,rA,rB

#### Description:

Bit 31 of general-purpose register rA is placed in high-order bits of general-purpose register rD. The low-order 32 bits of general-purpose register rA are copied from low-order 32 bits of general-purpose register rD.

#### 32-bit Implementation:

rD[31:0] <- rA[31:0]

#### 64-bit Implementation:

rD[63:32] <- rA[31]  
rD[31:0] <- rA[31:0]

#### Exceptions:

None

Instruction Class	Implementation
ORBIS64 II	Optional

---

**l.extwz**

**Extend Word with Zero**

**l.extwz**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x1	reserved	opcode 0xd
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.extwz rD,rA,rB`

**Description:**

Zero is placed in high-order bits of general-purpose register rD. The low-order 32 bits of general-purpose register rA are copied into the low-order 32 bits of general-purpose register rD.

**32-bit Implementation:**

`rD[31:0] <- rA[31:0]`

**64-bit Implementation:**

`rD[63:32] <- 0`  
`rD[31:0] <- rA[31:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS64 II	Optional

---

**l.ff1**

**Find First 1**

**l.ff1**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0xf	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.ff1 rD,rA,rB`

**Description:**

Position of the first '1' bit is written into general-purpose register rD. Checking for bit '1' starts with MSB and counting is decremented for every zero bit. If first '1' bit is discovered in LSB, one is written into rD. If there is no '1' bits, zero is written in rD.

**32-bit Implementation:**

`rD[31:0] <- rA[31] ? 32 : rA[30] ? 31 ... rA[0] ? 1 : 0`

**64-bit Implementation:**

`rD[63:0] <- rA[63] ? 64 : rA[62] ? 63 ... rA[0] ? 1 : 0`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.j

## Jump

l.j

31 . . . . . 26	25 . . . . . 0
opcode 0x0	N
6 bits	26bits

### Format:

l.j N

### Description:

The immediate value is shifted left two bits, sign-extended to program counter width and then added to the address of the delay slot. The result is the effective address of the jump. The program unconditionally jumps to EA with a delay of one instruction.

### 32-bit Implementation:

PC <- (Immediate << 2) + DelayInsnAddr  
 LR <- DelayInsnAddr + 4

### 64-bit Implementation:

PC <- (Immediate << 2) + DelayInsnAddr  
 LR <- DelayInsnAddr + 4

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required

l.jal

## Jump and Link

l.jal

31 . . . . . 26	25 . . . . . 0
opcode 0x1	N
6 bits	26bits

### Format:

l.jal N

### Description:

The immediate value is shifted left two bits, sign-extended to program counter width and then added to the address of the delay slot. The result is the effective address of the jump. The program unconditionally jumps to EA with a delay of one instruction. The address of the instruction after the delay slot is placed in the link register.

### 32-bit Implementation:

PC <- (Immediate << 2) + DelayInsnAddr  
 LR <- DelayInsnAddr + 4

### 64-bit Implementation:

PC <- (Immediate << 2) + DelayInsnAddr  
 LR <- DelayInsnAddr + 4

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required

l.jalr

### Jump and Link Register

l.jalr

31 . . . . . 26	25 . . . . . 16	15 . . . . . 11	10 . . . . . 0
opcode 0x12	reserved	B	reserved
6 bits	10 bits	5 bits	11bits

**Format:**

l.jalr rB

**Description:**

The contents of general-purpose register rB is the effective address of the jump. The program unconditionally jumps to EA with a delay of one instruction. The address of the instruction after the delay slot is placed in the link register.

**32-bit Implementation:**

PC <- rB  
 LR <- DelayInsnAddr + 4

**64-bit Implementation:**

PC <- rB  
 LR <- DelayInsnAddr + 4

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

l.jr

## Jump Register

l.jr

31 . . . . . 26	25 . . . . . 16	15 . . . . . 11	10 . . . . . 0
opcode 0x11	reserved	B	reserved
6 bits	10 bits	5 bits	11bits

### Format:

l.jr rB

### Description:

The contents of general-purpose register rB is the effective address of the jump. The program unconditionally jumps to EA with a delay of one instruction.

### 32-bit Implementation:

PC <- rB

### 64-bit Implementation:

PC <- rB

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required

1.lbs

### Load Byte and Extend with Sign

1.lbs

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x24	D	A	I
6 bits	5 bits	5 bits	16bits

#### Format:

1.lbs rD,I(rA)

#### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The byte in memory addressed by EA is loaded into the low-order eight bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with bit 7 of the loaded value.

#### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[7:0] <- (EA)[7:0]
rD[31:8] <- rA[7]
```

#### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[7:0] <- (EA)[7:0]
rD[63:8] <- rA[7]
```

#### Exceptions:

- TLB miss
- Page fault
- Bus error
- Alignment

Instruction Class	Implementation
ORBIS32 I	Required



1.lbz

## Load Byte and Extend with Zero

1.lbz

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x23	D	A	I
6 bits	5 bits	5 bits	16bits

### Format:

1.lbz rD,I(rA)

### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The byte in memory addressed by EA is loaded into the low-order eight bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with zero.

### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[7:0] <- (EA)[7:0]
rD[31:8] <- 0
```

### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[7:0] <- (EA)[7:0]
rD[63:8] <- 0
```

### Exceptions:

TLB miss  
Page fault  
Bus error  
Alignment

Instruction Class	Implementation
ORBIS32 I	Required



l.lhs

## Load Half Word and Extend with Sign

l.lhs

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x26	D	A	I
6 bits	5 bits	5 bits	16bits

### Format:

l.lhs rD,I(rA)

### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The half word in memory addressed by EA is loaded into the low-order 16 bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with bit 15 of the loaded value.

### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[15:0] <- (EA)[15:0]
rD[31:16] <- rA[15]
```

### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[15:0] <- (EA)[15:0]
rD[63:16] <- rA[15]
```

### Exceptions:

TLB miss  
Page fault  
Bus error  
Alignment

Instruction Class	Implementation
ORBIS32 I	Required

1.lhz

### Load Half Word and Extend with Zero

1.lhz

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x25	D	A	I
6 bits	5 bits	5 bits	16bits

#### Format:

1.lhz rD,I(rA)

#### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The half word in memory addressed by EA is loaded into the low-order 16 bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with zero.

#### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[15:0] <- (EA)[15:0]
rD[31:16] <- 0
```

#### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[15:0] <- (EA)[15:0]
rD[63:16] <- 0
```

#### Exceptions:

TLB miss  
Page fault  
Bus error  
Alignment

Instruction Class	Implementation
ORBIS32 I	Required

l.lws

## Load Single Word and Extend with Sign

l.lws

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x22	D	A	I
6 bits	5 bits	5 bits	16bits

### Format:

```
l.lws rD,I(rA)
```

### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The single word in memory addressed by EA is loaded into the low-order 32 bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with bit 31 of the loaded value.

### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[31:0] <- (EA)[31:0]
```

### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[31:0] <- (EA)[31:0]
rD[63:32] <- rA[31]
```

### Exceptions:

TLB miss  
Page fault  
Bus error  
Alignment

Instruction Class	Implementation
ORBIS32 I	Required

l.lwz

## Load Single Word and Extend with Zero

l.lwz

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x21	D	A	I
6 bits	5 bits	5 bits	16bits

### Format:

l.lwz rD,I(rA)

### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The single word in memory addressed by EA is loaded into the low-order 32 bits of general-purpose register rD. High-order bits of general-purpose register rD are replaced with zero.

### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
rD[31:0] <- (EA)[31:0]
```

### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
rD[31:0] <- (EA)[31:0]
rD[63:32] <- 0
```

### Exceptions:

TLB miss  
Page fault  
Bus error  
Alignment

Instruction Class	Implementation
ORBIS32 I	Required

**l.mac**

## Multiply Signed and Accumulate

**l.mac**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 7	6 . . . . 4	3 . . . . 0
opcode 0x38	reserved	A	B	reserved	opcode 0x0	reserved	opcode 0x7
6 bits	5 bits	5 bits	5 bits	1 bits	3 bits	3 bits	4bits

### Format:

`l.mac rA,rB`

### Description:

The contents of general-purpose register rA and the contents of general-purpose register rB are multiplied and the result truncated to 32 bits and added to the special-purpose registers MACHI and MACLO. All operands are treated as signed integers.

### 32-bit Implementation:

```
temp[31:0] <- rA[31:0] * rB[31:0]
MACHI[31:0]MACLO[31:0] <- temp[31:0] + MACHI[31:0]MACLO[31:0]
```

### 64-bit Implementation:

```
temp[31:0] <- rA[63:0] * rB[63:0]
MACHI[31:0]MACLO[31:0] <- temp[31:0] + MACHI[31:0]MACLO[31:0]
```

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.maci

## Multiply Immediate Signed and Accumulate

l.maci

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x2d	reserved	A	I
6 bits	5 bits	5 bits	16bits

### Format:

l.maci rA,I

### Description:

Immediate and the contents of general-purpose register rA are multiplied and the result is truncated to 32 bits and added to the special-purpose registers MACHI and MACLO. All operands are treated as signed integers.

### 32-bit Implementation:

```
temp[31:0] <- rA[31:0] * Immediate
MACHI[31:0]MACLO[31:0] <- temp[31:0] + MACHI[31:0]MACLO[31:0]
```

### 64-bit Implementation:

```
temp[31:0] <- rA[63:0] * Immediate
MACHI[31:0]MACLO[31:0] <- temp[31:0] + MACHI[31:0]MACLO[31:0]
```

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 II	Optional



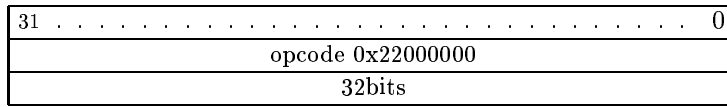




**l.msync**

## Memory Synchronization

**l.msync**



**Format:**

l.msync

**Description:**

Execution of the memory synchronization instruction results in completion of all load/store operations before the RISC core continues.

**32-bit Implementation:**

memory-synchronization

**64-bit Implementation:**

memory-synchronization

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

**l.mtspr**

**Move To Special-Purpose Register**

**l.mtspr**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x10	K	A	B	K
6 bits	5 bits	5 bits	5 bits	11bits

**Format:**

```
l.mtspr rA,rB,K
```

**Description:**

The contents of general-purpose register rB are moved into special register identified by the sum of general-purpose register rA and the immediate value.

**32-bit Implementation:**

```
spr(rB+Immediate) <- rA[31:0]
```

**64-bit Implementation:**

```
spr(rB+Immediate) <- rA[31:0]
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.mul**

**Multiply Signed**

**l.mul**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0x6	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.mul rD,rA,rB`

**Description:**

The contents of general-purpose register rA and the contents of general-purpose register rB are multiplied and the result is truncated to destination register width and placed into general-purpose register rD. Both operands are treated as signed integers.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] * rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] * rB[63:0]
SR[OV] <- overflow
SR[CY] <- carry
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.muli**

**Multiply Immediate Signed**

**l.muli**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x2c	D	A	I
6 bits	5 bits	5 bits	16bits

**Format:**

```
l.muli rD,rA,I
```

**Description:**

Immediate and the contents of general-purpose register rA are multiplied and the result is truncated to destination register width and placed into general-purpose register rD.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] * Immediate  
SR[OV] <- overflow  
SR[CY] <- carry
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] * Immediate  
SR[OV] <- overflow  
SR[CY] <- carry
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.mulu**

**Multiply Unsigned**

**l.mulu**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0xb	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits		4 bits	4bits

**Format:**

`l.mulu rD,rA,rB`

**Description:**

The contents of general-purpose register rA and the contents of general-purpose register rB are multiplied and the result is truncated to destination register width and placed into general-purpose register rD. Both operands are treated as unsigned integers.

**32-bit Implementation:**

```

rD[31:0] <- rA[31:0] * rB[31:0]
SR[OV] <- overflow
SR[CY] <- carry

```

**64-bit Implementation:**

```

rD[63:0] <- rA[63:0] * rB[63:0]
SR[OV] <- overflow
SR[CY] <- carry

```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.nop**

**No Operation**

**l.nop**

31 . . . . . 24	23 . . . . . 0
opcode 0x15	reserved
8 bits	24bits

**Format:**

l.nop

**Description:**

This instruction does not do anything except it takes at least one clock cycle to complete. It is often used to fill delay slot gaps.Immediate can be used for simulation purposes.

**32-bit Implementation:**

**64-bit Implementation:**

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



---

**l.or**

**Or**

**l.or**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x4	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.or rD,rA,rB`

**Description:**

The contents of general-purpose register rA are combined with the contents of general-purpose register rB in a bit-wise logical OR operation. The result is placed into general-purpose register rD.

**32-bit Implementation:**

`rD[31:0] <- rA[31:0] OR rB[31:0]`

**64-bit Implementation:**

`rD[63:0] <- rA[63:0] OR rB[63:0]`

**Exceptions:**

None

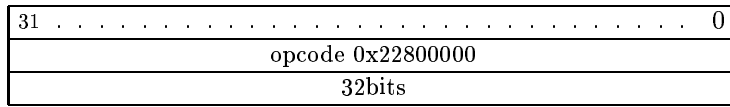
Instruction Class	Implementation
ORBIS32 I	Required



**l.psync**

**Pipeline Synchronization**

**l.psync**



**Format:**

l.psync

**Description:**

Execution of pipeline synchronization instruction results in completion of all instructions that were fetched before l.psync instruction. Once all instructions are completed, instructions fetched after l.psync are flushed from the pipeline and fetched again.

**32-bit Implementation:**

pipeline-synchronization

**64-bit Implementation:**

pipeline-synchronization

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.rfe

## Return From Exception

l.rfe

31 . . . . . 26	25 . . . . . 0
opcode 0x9	reserved
6 bits	26bits

### Format:

l.rfe

### Description:

Execution of this instruction partially restores the state of the processor prior to the exception. This instruction does not have a delay slot.

### 32-bit Implementation:

PC <- EPCR  
ESR <- SR

### 64-bit Implementation:

PC <- EPCR  
ESR <- SR

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.ror**

**Rotate Right**

**l.ror**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x3	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.ror rD,rA,rB`

**Description:**

General-purpose register `rB` specifies the number of bit positions the contents of general-purpose register `rA` are rotated right. The result is written into general-purpose register `rD`.

**32-bit Implementation:**

```
rD[31-rB[5:0]:0] <- rA[31:rB]
rD[31:32-rB[5:0]] <- rA[rB[5:0]-1:0]
```

**64-bit Implementation:**

```
rD[63-rB[5:0]:0] <- rA[63:rB]
rD[63:64-rB[5:0]] <- rA[rB[5:0]-1:0]
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

---

**l.rori**

**Rotate Right with Immediate**

**l.rori**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 6	5 . . . . . 0
opcode 0x2e	D	A	reserved	opcode 0x3	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

**Format:**

`l.rori rD,rA,L`

**Description:**

The 6-bit immediate value specifies the number of bit positions the contents of general-purpose register rA are rotated right. The result is written into general-purpose register rD.

**32-bit Implementation:**

```
rD[31-L:0] <- rA[31:L]
rD[31:32-L] <- rA[L-1:0]
```

**64-bit Implementation:**

```
rD[63-L:0] <- rA[63:L]
rD[63:64-L] <- rA[L-1:0]
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

l.sb

### Store Byte

l.sb

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x36	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

#### Format:

l.sb I(rA),rB

#### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The low-order 8 bits of general-purpose register rB are stored to memory location addressed by EA.

#### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
(EA)[7:0] <- rB[7:0]
```

#### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
(EA)[7:0] <- rB[7:0]
```

#### Exceptions:

```
TLB miss
Page fault
Bus error
Alignment
```

Instruction Class	Implementation
ORBIS32 I	Required

l.sd

### Store Double Word

l.sd

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x34	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

#### Format:

l.sd I(rA),rB

#### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The double word in general-purpose register rB is stored to memory location addressed by EA.

#### 32-bit Implementation:

N/A

#### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
(EA)[63:0] <- rB[63:0]
```

#### Exceptions:

- TLB miss
- Page fault
- Bus error
- Alignment

Instruction Class	Implementation
ORBIS64 I	Required



l.sfeq

**Set Flag if Equal**

l.sfeq

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x720	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

l.sfeq rA,rB

**Description:**

The contents of general-purpose registers rA and rB are compared. If the contents are equal, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

SR[F] <- rA[31:0] == rB[31:0]

**64-bit Implementation:**

SR[F] <- rA[63:0] == rB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



l.sfges

### Set Flag if Greater or Equal Than Signed

l.sfges

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x72b	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

l.sfges rA,rB

**Description:**

The contents of general-purpose registers rA and rB are compared as signed integers. If the contents of the first register are greater or equal than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

SR[F] <- rA[31:0] >= rB[31:0]

**64-bit Implementation:**

SR[F] <- rA[63:0] >= rB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



l.sfgeu

**Set Flag if Greater or Equal Than Unsigned**

l.sfgeu

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x723	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

l.sfgeu rA,rB

**Description:**

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the contents of the first register are greater or equal than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

SR[F] <- rA[31:0] >= rB[31:0]

**64-bit Implementation:**

SR[F] <- rA[63:0] >= rB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.sfgeui      Set Flag if Greater or Equal Than Immediate Unsigned      l.sfgeui**

31 . . . . . 21	20 . . . . 16	15 . . . . . . . . . . 0
opcode 0x5e3	A	I
11 bits	5 bits	16bits

**Format:**

`l.sfgeui rA,I`

**Description:**

The contents of general-purpose register rA and the zero-extended immediate value are compared as unsigned integers. If the contents of the first register are greater than or equal to the immediate value then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- rA[31:0] >= extz(Immediate)`

**64-bit Implementation:**

`SR[F] <- rA[63:0] >= extz(Immediate)`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.sfgts

### Set Flag if Greater Than Signed

l.sfgts

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x72a	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

l.sfgts rA,rB

**Description:**

The contents of general-purpose registers rA and rB are compared as signed integers. If the contents of the first register are greater than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

SR[F] <- rA[31:0] > rB[31:0]

**64-bit Implementation:**

SR[F] <- rA[63:0] > rB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required





**l.sfgtu**

**Set Flag if Greater Than Unsigned**

**l.sfgtu**

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x722	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

`l.sfgtu rA,rB`

**Description:**

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the contents of the first register are greater than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

$SR[F] \leftarrow rA[31:0] > rB[31:0]$

**64-bit Implementation:**

$SR[F] \leftarrow rA[63:0] > rB[63:0]$

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



l.sfles

### Set Flag if Less or Equal Than Signed

l.sfles

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x72d	A	B	reserved
11 bits	5 bits	5 bits	11bits

#### Format:

l.sfles rA,rB

#### Description:

The contents of general-purpose registers rA and rB are compared as signed integers. If the contents of the first register are less or equal than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

#### 32-bit Implementation:

SR[F] <- rA[31:0] <= rB[31:0]

#### 64-bit Implementation:

SR[F] <- rA[63:0] <= rB[63:0]

#### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required

---

l.sflesi

## Set Flag if Less or Equal Than Immediate Signed

l.sflesi

31 . . . . . 21	20 . . . . 16	15 . . . . . . . . . . 0
opcode 0x5ed	A	I
11 bits	5 bits	16bits

### Format:

l.sflesi rA,I

### Description:

The contents of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the contents of the first register are less than or equal to the immediate value then the compare flag is set; otherwise the compare flag is cleared.

### 32-bit Implementation:

SR[F] <- rA[31:0] <= exts(Immediate)

### 64-bit Implementation:

SR[F] <- rA[63:0] <= exts(Immediate)

### Exceptions:

None

Instruction Class	Implementation
ORBIS32 II	Optional

**l.sfleu**

**Set Flag if Less or Equal Than Unsigned**

**l.sfleu**

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x725	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

`l.sfleu rA,rB`

**Description:**

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the contents of the first register are less or equal than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

$SR[F] \leftarrow rA[31:0] \leq rB[31:0]$

**64-bit Implementation:**

$SR[F] \leftarrow rA[63:0] \leq rB[63:0]$

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**l.sfleui      Set Flag if Less or Equal Than Immediate Unsigned      l.sfleui**

31 . . . . . 21	20 . . . . 16	15 . . . . . . . . . . 0
opcode 0x5e5	A	I
11 bits	5 bits	16bits

**Format:**

`l.sfleui rA,I`

**Description:**

The contents of general-purpose register rA and the zero-extended immediate value are compared as unsigned integers. If the contents of the first register are less than or equal to the immediate value then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- rA[31:0] <= extz(Immediate)`

**64-bit Implementation:**

`SR[F] <- rA[63:0] <= extz(Immediate)`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional

l.sflts

### Set Flag if Less Than Signed

l.sflts

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x72c	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

```
l.sflts rA,rB
```

**Description:**

The contents of general-purpose registers rA and rB are compared as signed integers. If the contents of the first register are less than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

$$SR[F] \leftarrow rA[31:0] < rB[31:0]$$

**64-bit Implementation:**

$$SR[F] \leftarrow rA[63:0] < rB[63:0]$$

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

l.sfltsi

### Set Flag if Less Than Immediate Signed

l.sfltsi

31 . . . . . 21	20 . . . . 16	15 . . . . . . . . . . 0
opcode 0x5ec	A	I
11 bits	5 bits	16bits

**Format:**

```
l.sfltsi rA,I
```

**Description:**

The contents of general-purpose register rA and the sign-extended immediate value are compared as signed integers. If the contents of the first register are less than the immediate value then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

$$SR[F] \leftarrow rA[31:0] < \text{exts}(\text{Immediate})$$

**64-bit Implementation:**

$$SR[F] \leftarrow rA[63:0] < \text{exts}(\text{Immediate})$$

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 II	Optional



**l.sftu**

**Set Flag if Less Than Unsigned**

**l.sftu**

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x724	A	B	reserved
11 bits	5 bits	5 bits	11bits

**Format:**

`l.sftu rA,rB`

**Description:**

The contents of general-purpose registers rA and rB are compared as unsigned integers. If the contents of the first register are less than the contents of the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

$SR[F] \leftarrow rA[31:0] < rB[31:0]$

**64-bit Implementation:**

$SR[F] \leftarrow rA[63:0] < rB[63:0]$

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



l.sfne

### Set Flag if Not Equal

l.sfne

31 . . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x721	A	B	reserved
11 bits	5 bits	5 bits	11bits

#### Format:

l.sfne rA,rB

#### Description:

The contents of general-purpose registers rA and rB are compared. If the contents are not equal, then the compare flag is set; otherwise the compare flag is cleared.

#### 32-bit Implementation:

SR[F] <- rA[31:0] != rB[31:0]

#### 64-bit Implementation:

SR[F] <- rA[63:0] != rB[63:0]

#### Exceptions:

None

Instruction Class	Implementation
ORBIS32 I	Required



**l.sh**

**Store Half Word**

**l.sh**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x37	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

**Format:**

`l.sh I(rA),rB`

**Description:**

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The low-order 16 bits of general-purpose register rB are stored to memory location addressed by EA.

**32-bit Implementation:**

```
EA <- exts(Immediate) + rA[31:0]
(EA)[15:0] <- rB[15:0]
```

**64-bit Implementation:**

```
EA <- exts(Immediate) + rA[63:0]
(EA)[15:0] <- rB[15:0]
```

**Exceptions:**

- TLB miss
- Page fault
- Bus error
- Alignment

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.sll**

**Shift Left Logical**

**l.sll**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.sll rD,rA,rB`

**Description:**

General-purpose register `rB` specifies the number of bit positions the contents of general-purpose register `rA` are shifted left, inserting zeros into the low-order bits. The result is written into general-purpose `rD`.

**32-bit Implementation:**

```
rD[31:rB] <- rA[31-rB:0]
rD[rB-1:0] <- 0
```

**64-bit Implementation:**

```
rD[63:rB] <- rA[63-rB:0]
rD[rB-1:0] <- 0
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.slli**

**Shift Left Logical with Immediate**

**l.slli**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 6	5 . . . . . 0
opcode 0x2e	D	A	reserved	opcode 0x0	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

**Format:**

`l.slli rD,rA,L`

**Description:**

The 6-bit immediate value specifies the number of bit positions the contents of general-purpose register rA are shifted left, inserting zeros into the low-order bits. The result is written into general-purpose register rD.

**32-bit Implementation:**

`rD[31:L] <- rA[31-L:0]`  
`rD[L-1:0] <- 0`

**64-bit Implementation:**

`rD[63:L] <- rA[63-L:0]`  
`rD[L-1:0] <- 0`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.sra**

**Shift Right Arithmetic**

**l.sra**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x2	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.sra rD,rA,rB`

**Description:**

General-purpose register `rB` specifies the number of bit positions the contents of general-purpose register `rA` are shifted right, sign-extending the high-order bits. The result is written into general-purpose register `rD`.

**32-bit Implementation:**

```
rD[31-rB[5:0]:0] <- rA[31:rB[5:0]]
rD[31:32-rB[5:0]] <- rA[31]
```

**64-bit Implementation:**

```
rD[63-rB[5:0]:0] <- rA[63:rB[5:0]]
rD[63:64-rB[5:0]] <- rA[63]
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



---

**l.srai**

**Shift Right Arithmetic with Immediate**

**l.srai**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 6	5 . . . . . 0
opcode 0x2e	D	A	reserved	opcode 0x2	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

**Format:**

`l.srai rD,rA,L`

**Description:**

The 6-bit immediate value specifies the number of bit positions the contents of general-purpose register rA are shifted right, sign-extending the high-order bits. The result is written into general-purpose register rD.

**32-bit Implementation:**

`rD[31-L:0] <- rA[31:L]`  
`rD[31:32-L] <- rA[31]`

**64-bit Implementation:**

`rD[63-L:0] <- rA[63:L]`  
`rD[63:64-L] <- rA[63]`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.srl**

**Shift Right Logical**

**l.srl**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9 . . . . 6	5 . . . . 4	3 . . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x1	reserved	opcode 0x8
6 bits	5 bits	5 bits	5 bits	1 bits	4 bits	2 bits	4bits

**Format:**

`l.srl rD,rA,rB`

**Description:**

General-purpose register `rB` specifies the number of bit positions the contents of general-purpose register `rA` are shifted right, inserting zeros into the high-order bits. The result is written into general-purpose register `rD`.

**32-bit Implementation:**

```
rD[31-rB[5:0]:0] <- rA[31:rB[5:0]]  
rD[31:32-rB[5:0]] <- 0
```

**64-bit Implementation:**

```
rD[63-rB[5:0]:0] <- rA[63:rB[5:0]]  
rD[63:64-rB[5:0]] <- 0
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.srli**

**Shift Right Logical with Immediate**

**l.srli**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 6	5 . . . . . 0
opcode 0x2e	D	A	reserved	opcode 0x1	L
6 bits	5 bits	5 bits	8 bits	2 bits	6bits

**Format:**

`l.srli rD,rA,L`

**Description:**

The 6-bit immediate value specifies the number of bit positions the contents of general-purpose register rA are shifted right, inserting zeros into the high-order bits. The result is written into general-purpose register rD.

**32-bit Implementation:**

`rD[31-L:0] <- rA[31:L]`  
`rD[31:32-L] <- 0`

**64-bit Implementation:**

`rD[63-L:0] <- rA[63:L]`  
`rD[63:64-L] <- 0`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

---

**l.sub**

**Subtract Signed**

**l.sub**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x2	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

```
l.sub rD,rA,rB
```

**Description:**

The contents of general-purpose register rB are subtracted from the contents of general-purpose register rA to form the result. The result is placed into general-purpose register rD.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] - rB[31:0]  
SR[CY] <- carry  
SR[OV] <- overflow
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] - rB[63:0]  
SR[CY] <- carry  
SR[OV] <- overflow
```

**Exceptions:**

Range Exception

Instruction Class	Implementation
ORBIS32 I	Required

**l.sw**

## Store Single Word

**l.sw**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . . . . . . . 0
opcode 0x35	I	A	B	I
6 bits	5 bits	5 bits	5 bits	11bits

### Format:

`l.sw I(rA),rB`

### Description:

The offset is sign-extended and added to the contents of general-purpose register rA. The sum represents an effective address. The low-order 32 bits of general-purpose register rB are stored to memory location addressed by EA.

### 32-bit Implementation:

```
EA <- exts(Immediate) + rA[31:0]
(EA)[31:0] <- rB[31:0]
```

### 64-bit Implementation:

```
EA <- exts(Immediate) + rA[63:0]
(EA)[31:0] <- rB[31:0]
```

### Exceptions:

- TLB miss
- Page fault
- Bus error
- Alignment

Instruction Class	Implementation
ORBIS32 I	Required

l.sys

System Call

l.sys

31 . . . . . 16	15 . . . . . 0
opcode 0x2000	K
16 bits	16bits

**Format:**

l.sys K

**Description:**

Execution of the system call instruction results in the system call exception. The system calls exception is a request to the operating system to provide operating system services. The immediate value specifies which system service is required.

**32-bit Implementation:**

system-call-exception(K)

**64-bit Implementation:**

system-call-exception(K)

**Exceptions:**

System Call

Instruction Class	Implementation
ORBIS32 I	Required

**l.trap**

**Trap**

**l.trap**

31 . . . . . 16	15 . . . . . 0
opcode 0x2100	K
16 bits	16bits

**Format:**

l.trap K

**Description:**

Execution of trap instruction results in the trap exception if specified flag is set. Trap eception is request to operating systemor to the debug facilityto execute certain debug services. Immediate is used to select which SR bit is tested by trap instruction.

**32-bit Implementation:**

if SR[K] = 1 then breakpoint-exception(I)

**64-bit Implementation:**

if SR[K] = 1 then breakpoint-exception(I)

**Exceptions:**

Trap exception

Instruction Class	Implementation
ORBIS32 II	Optional

---

**l.xor**

**Exclusive Or**

**l.xor**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10	9	8	7 . . . 4	3 . . . 0
opcode 0x38	D	A	B	reserved	opcode 0x0	reserved	opcode 0x5	
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits	

**Format:**

`l.xor rD,rA,rB`

**Description:**

The contents of general-purpose register rA are combined with the contents of general-purpose register rB in a bit-wise logical XOR operation. The result is placed into general-purpose register rD.

**32-bit Implementation:**

`rD[31:0] <- rA[31:0] XOR rB[31:0]`

**64-bit Implementation:**

`rD[63:0] <- rA[63:0] XOR rB[63:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required



**l.xori**

**Exclusive Or with Immediate Half Word**

**l.xori**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 0
opcode 0x2b	D	A	I
6 bits	5 bits	5 bits	16bits

**Format:**

```
l.xori rD,rA,I
```

**Description:**

The immediate value is zero-extended and combined with the contents of general-purpose register rA in a bit-wise logical XOR operation. The result is placed into general-purpose register rD.

**32-bit Implementation:**

```
rD[31:0] <- rA[31:0] XOR extz(Immediate)
```

**64-bit Implementation:**

```
rD[63:0] <- rA[63:0] XOR extz(Immediate)
```

**Exceptions:**

None

Instruction Class	Implementation
ORBIS32 I	Required

**0.2. ORFPX32/64**

lf.add.d

## Add Floating-Point Double-Precision

lf.add.d

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x10
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.add.d rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are added to the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] + vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required

lf.add.s

### Add Floating-Point Single-Precision

lf.add.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x10
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lf.add.s rD,rA,rB

**Description:**

The contents of vector/floating-point register vfrA is added to the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

**32-bit Implementation:**

vfrD[31:0] <- vfrA[31:0] + vfrB[31:0]

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required



lf.cust1.s

Reserved for ORFPX32 Custom Instructions

lf.cust1.s

31 . . . . . 26	25 . . . . . 8	7 . . . . . 4	3 . . . . . 0
opcode 0xb	reserved	opcode 0xe	reserved
6 bits	18 bits	4 bits	4bits

**Format:**

lf.cust1.s

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but instead by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORFPX32 II	Optional

---

lf.div.d

## Divide Floating-Point Double-Precision

lf.div.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x13
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.div.d rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are divided by the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] / vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 II	Optional

lf.div.s

## Divide Floating-Point Single-Precision

lf.div.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x13
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.div.s rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA is divided by the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

vfrD[31:0] <- vfrA[31:0] / vfrB[31:0]

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 II	Optional

lf.ftoi.d

## Floating-Point Double-Precision To Integer

lf.ftoi.d

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x15
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.ftoi.d rD,rA

### Description:

The contents of vector/floating-point register vfrA are converted to an integer and stored in general-purpose register rD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

rD[63:0] <- ftoi(vfrA[63:0])

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required



lf.ftoi.s

## Floating-Point Single-Precision To Integer

lf.ftoi.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x15
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.ftoi.s rD,rA

### Description:

The contents of vector/floating-point register vfrA are converted to an integer and stored into general-purpose register rD.

### 32-bit Implementation:

rD[31:0] <- ftoi(vfrA[31:0])

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 I	Required

lf.itof.d

## Integer To Floating-Point Double-Precision

lf.itof.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x14
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.itof.d rD,rA

### Description:

The contents of general-purpose register rA are converted to a Double-precision floating-point number and stored in vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- itof(rA[63:0])

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required

lf.itof.s

## Integer To Floating-Point Single-Precision

lf.itof.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x14
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.itof.s rD,rA

### Description:

The contents of general-purpose register rA are converted to a single-precision floating-point number and stored into vector/floating-point register vfrD.

### 32-bit Implementation:

vfrD[31:0] <- itof(rA[31:0])

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 I	Required

lf.madd.d

## Multiply and Add Floating-Point Double-Precision

lf.madd.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x17
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.madd.d rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are multiplied by the contents of vector/floating-point register vfrB and added to special-purpose register FPMADDLO/FPMADDHI.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

FPMADDHI[31:0]FPMADDLO[31:0] <- vfrA[63:0] \* vfrB[63:0] + FPMADDHI[31:0]FPMADDLO[31:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 II	Optional

lf.madd.s

## Multiply and Add Floating-Point Single-Precision

lf.madd.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x17
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.madd.s rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA is multiplied by the contents of vector/floating-point register vfrB and added to special-purpose register FPMADDLO/FPMADDHI.

### 32-bit Implementation:

FPMADDHI[31:0]FPMADDLO[31:0] <- vfrA[31:0] \* vfrB[31:0] + FPMADDHI[31:0]FPMADDLO[31:0]

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 II	Optional

---

lf.mul.d

## Multiply Floating-Point Double-Precision

lf.mul.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x12
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lf.mul.d rD,rA,rB
```

### Description:

The contents of vector/floating-point register vfrA are multiplied by the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[63:0] <- vfrA[63:0] * vfrB[63:0]
```

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required

lf.mul.s

## Multiply Floating-Point Single-Precision

lf.mul.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x12
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.mul.s rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA is multiplied by the contents of vector/floating-point register vfrB to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

vfrD[31:0] <- vfrA[31:0] \* vfrB[31:0]

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 I	Required

---

lf.rem.d

## Remainder Floating-Point Double-Precision

lf.rem.d

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x16
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.rem.d rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are divided by the contents of vector/floating-point register vfrB and remainder is used as the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] % vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 II	Optional



lf.rem.s

## Remainder Floating-Point Single-Precision

lf.rem.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x16
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.rem.s rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA is divided by the contents of vector/floating-point register vfrB and remainder is used as the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

vfrD[31:0] <- vfrA[31:0] % vfrB[31:0]

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 II	Optional

---

lf.sfeq.d

**Set Flag if Equal Floating-Point Double-Precision**

lf.sfeq.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x18
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lf.sfeq.d rA,rB

**Description:**

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If the two registers are equal, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

SR[F] <- vfrA[63:0] == vfrB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORFPX64 I	Required

---

lf.sfeq.s

**Set Flag if Equal Floating-Point Single-Precision**

lf.sfeq.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0x18
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lf.sfeq.s rA,rB

**Description:**

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If the two registers are equal, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

SR[F] <- vfrA[31:0] == vfrB[31:0]

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

---

## lf.sfge.d Set Flag if Greater or Equal Than Floating-Point Double-Precision lf.sfge.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x1b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.sfge.d rA,rB

### Description:

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If first register is greater than or equal to the second register, then the compare flag is set; otherwise the compare flag is cleared.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

SR[F] <- vfrA[63:0] >= vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required

**lf.sfge.s Set Flag if Greater or Equal Than Floating-Point Single-Precision lf.sfge.s**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0x1b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfge.s rA,rB`

**Description:**

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If first register is greater than or equal to the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- vfrA[31:0] >= vfrB[31:0]`

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

**lf.sfgt.d    Set Flag if Greater Than Floating-Point Double-Precision    lf.sfgt.d**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x1a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfgt.d rA,rB`

**Description:**

The contents of vector/floating-point register `vfrA` and the contents of vector/floating-point register `vfrB` are compared. If first register is greater than the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

`SR[F] <- vfrA[63:0] > vfrB[63:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORFPX64 I	Required

**lf.sfgt.s      Set Flag if Greater Than Floating-Point Single-Precision      lf.sfgt.s**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0x1a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfgt.s rA,rB`

**Description:**

The contents of vector/floating-point register `vfrA` and the contents of vector/floating-point register `vfrB` are compared. If the first register is greater than the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- vfrA[31:0] > vfrB[31:0]`

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

---

## lf.sfle.d Set Flag if Less or Equal Than Floating-Point Double-Precision lf.sfle.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x1d
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.sfle.d rA,rB

### Description:

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If first register is less than or equal to the second register, then the compare flag is set; otherwise the compare flag is cleared.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

SR[F] <- vfrA[363:0] <= vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required



**lf.sfle.s Set Flag if Less or Equal Than Floating-Point Single-Precision lf.sfle.s**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0xd
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfle.s rA,rB`

**Description:**

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If first register is less than or equal to the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- vfrA[31:0] <= vfrB[31:0]`

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

**lf.sflt.d      Set Flag if Less Than Floating-Point Double-Precision      lf.sflt.d**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x1c
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sflt.d rA,rB`

**Description:**

The contents of vector/floating-point register `vfrA` and the contents of vector/floating-point register `vfrB` are compared. If first register is less than the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

`SR[F] <- vfrA[63:0] < vfrB[63:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORFPX64 I	Required

**lf.sflt.s      Set Flag if Less Than Floating-Point Single-Precision      lf.sflt.s**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0x1c
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sflt.s rA,rB`

**Description:**

The contents of vector/floating-point register vfrA and the contents of vector/floating-point register vfrB are compared. If first register is less than the second register, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- vfrA[31:0] < vfrB[31:0]`

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

**lf.sfne.d      Set Flag if Not Equal Floating-Point Double-Precision      lf.sfne.d**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	reserved	A	B	reserved	opcode 0x19
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfne.d rA,rB`

**Description:**

The contents of vector/floating-point register `vfrA` and the contents of vector/floating-point register `vfrB` are compared. If the two registers are not equal, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

`SR[F] <- vfrA[63:0] != vfrB[63:0]`

**Exceptions:**

None

Instruction Class	Implementation
ORFPX64 I	Required

**lf.sfne.s      Set Flag if Not Equal Floating-Point Single-Precision      lf.sfne.s**

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	reserved	A	B	reserved	opcode 0x19
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lf.sfne.s rA,rB`

**Description:**

The contents of vector/floating-point register `vfrA` and the contents of vector/floating-point register `vfrB` are compared. If the two registers are not equal, then the compare flag is set; otherwise the compare flag is cleared.

**32-bit Implementation:**

`SR[F] <- vfrA[31:0] != vfrB[31:0]`

**64-bit Implementation:**

N/A

**Exceptions:**

None

Instruction Class	Implementation
ORFPX32 I	Required

lf.sub.d

## Subtract Floating-Point Double-Precision

lf.sub.d

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xc	D	A	B	reserved	opcode 0x11
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.sub.d rD,rA,rB

### Description:

The contents of vector/floating-point register vfrB are subtracted from the contents of vector/floating-point register vfrA to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] - vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORFPX64 I	Required

lf.sub.s

## Subtract Floating-Point Single-Precision

lf.sub.s

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xb	D	A	B	reserved	opcode 0x11
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lf.sub.s rD,rA,rB

### Description:

The contents of vector/floating-point register vfrB is subtracted from the contents of vector/floating-point register vfrA to form the result. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

vfrD[31:0] <- vfrA[31:0] - vfrB[31:0]

### 64-bit Implementation:

N/A

### Exceptions:

None

Instruction Class	Implementation
ORFPX32 I	Required

---

lvf.ld

Load Vector/Floating-Point Double Word

lvf.ld

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 0
opcode 0xd	D	A	reserved	opcode 0x0
6 bits	5 bits	5 bits	8 bits	8bits

**Format:**

lvf.ld rD,0(rA)

**Description:**

The contents of vector/floating-point register vfrA is used as an effective address. The double word in memory addressed by EA is loaded into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
EA <- vfrA[63:0]
vfrD[63:0] <- (EA)[63:0]
```

**Exceptions:**

TLB miss  
Page fault  
Bus error

Instruction Class	Implementation
ORFPX64 I	Required



---

lvf.lw

## Load Vector/Floating-Point Single Word

lvf.lw

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 8	7 . . . . . 0
opcode 0xd	D	A	reserved	opcode 0x1
6 bits	5 bits	5 bits	8 bits	8bits

### Format:

```
lvf.lw rD,0(rA)
```

### Description:

The contents of vector/floating-point register vfrA is used as an effective address. The double word in memory addressed by EA is loaded into vector/floating-point register vfrD.

### 32-bit Implementation:

```
EA <- vfrA[31:0]  
vfrD[31:0] <- (EA)[31:0]
```

### 64-bit Implementation:

```
EA <- vfrA[31:0]  
vfrD[31:0] <- (EA)[31:0]
```

### Exceptions:

TLB miss  
Page fault  
Bus error

Instruction Class	Implementation
ORFPX32 I	Required

---

lvf.sd

## Store Vector/Floating-Point Double Word

lvf.sd

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xd	reserved	A	B	reserved	opcode 0x10
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lvf.sd 0(rA),rB

### Description:

The contents of vector/floating-point register vfrA is used as an effective address. The double word in vector/floating-point register vfrB is stored to the memory location addressed by EA.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
EA <- vfrA[63:0]
vfrD[63:0] <- (EA)[63:0]
```

### Exceptions:

TLB miss  
Page fault  
Bus error

Instruction Class	Implementation
ORFPX64 I	Required

---

lvf.sw

## Store Vector/Floating-Point Single Word

lvf.sw

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xd	reserved	A	B	reserved	opcode 0x11
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lvf.sw 0(rA),rB
```

### Description:

The contents of vector/floating-point register vfrA is used as an effective address. The single word in vector/floating-point register vfrB is stored to the memory location addressed by EA.

### 32-bit Implementation:

```
EA <- vfrA[31:0]  
vfrD[31:0] <- (EA)[31:0]
```

### 64-bit Implementation:

```
EA <- vfrA[31:0]  
vfrD[31:0] <- (EA)[31:0]
```

### Exceptions:

```
TLB miss  
Page fault  
Bus error
```

Instruction Class	Implementation
ORFPX32 I	Required

### 0.3. ORVDX64

lv.add.b

### Vector Byte Elements Add Signed

lv.add.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x30
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.add.b rD,rA,rB

**Description:**

The byte elements of vector/floating-point register vfrA are added to the byte elements of vector/floating-point register vfrB to form the result elements. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```

vfrD[7:0] <- vfrA[7:0] + vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] + vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] + vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] + vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] + vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] + vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] + vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] + vfrB[63:56]

```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x31
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.add.h rD,rA,rB
```

**Description:**

The half-word elements of vector/floating-point register vfrA are added to the half-word elements of vector/floating-point register vfrB to form the result elements. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- vfrA[15:0] + vfrB[15:0]
vfrD[31:16] <- vfrA[31:16] + vfrB[31:16]
vfrD[47:32] <- vfrA[47:32] + vfrB[47:32]
vfrD[63:48] <- vfrA[63:48] + vfrB[63:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.adds.b

## Vector Byte Elements Add Signed Saturated

lv.adds.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x32
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.adds.b rD,rA,rB

### Description:

The byte elements of vector/floating-point register vfrA are added to the byte elements of vector/floating-point register vfrB to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- sat8s(vfrA[7:0] + vfrB[7:0])
vfrD[15:8] <- sat8s(vfrA[15:8] + vfrB[15:8])
vfrD[23:16] <- sat8s(vfrA[23:16] + vfrB[23:16])
vfrD[31:24] <- sat8s(vfrA[31:24] + vfrB[31:24])
vfrD[39:32] <- sat8s(vfrA[39:32] + vfrB[39:32])
vfrD[47:40] <- sat8s(vfrA[47:40] + vfrB[47:40])
vfrD[55:48] <- sat8s(vfrA[55:48] + vfrB[55:48])
vfrD[63:56] <- sat8s(vfrA[63:56] + vfrB[63:56])
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x33
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.adds.h rD,rA,rB
```

**Description:**

The half-word elements of vector/floating-point register `vfrA` are added to the half-word elements of vector/floating-point register `vfrB` to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat16s(vfrA[15:0] + vfrB[15:0])
vfrD[31:16] <- sat16s(vfrA[31:16] + vfrB[31:16])
vfrD[47:32] <- sat16s(vfrA[47:32] + vfrB[47:32])
vfrD[63:48] <- sat16s(vfrA[63:48] + vfrB[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.addu.b

## Vector Byte Elements Add Unsigned

lv.addu.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x34
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.addu.b rD,rA,rB

### Description:

The unsigned byte elements of vector/floating-point register vfrA are added to the unsigned byte elements of vector/floating-point register vfrB to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] + vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] + vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] + vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] + vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] + vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] + vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] + vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] + vfrB[63:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.addu.h

## Vector Half-Word Elements Add Unsigned

lv.addu.h

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x35
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.addu.h rD,rA,rB

### Description:

The unsigned half-word elements of vector/floating-point register vfrA are added to the unsigned half-word elements of vector/floating-point register vfrB to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] + vfrB[15:0]
vfrD[31:16] <- vfrA[31:16] + vfrB[31:16]
vfrD[47:32] <- vfrA[47:32] + vfrB[47:32]
vfrD[63:48] <- vfrA[63:48] + vfrB[63:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.addus.b

## Vector Byte Elements Add Unsigned Saturated

lv.addus.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x36
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.addus.b rD,rA,rB

### Description:

The unsigned byte elements of vector/floating-point register vfrA are added to the unsigned byte elements of vector/floating-point register vfrB to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- sat8u(vfrA[7:0] + vfrB[7:0])
vfrD[15:8] <- sat8u(vfrA[15:8] + vfrB[15:8])
vfrD[23:16] <- sat8u(vfrA[23:16] + vfrB[23:16])
vfrD[31:24] <- sat8u(vfrA[31:24] + vfrB[31:24])
vfrD[39:32] <- sat8u(vfrA[39:32] + vfrB[39:32])
vfrD[47:40] <- sat8u(vfrA[47:40] + vfrB[47:40])
vfrD[55:48] <- sat8u(vfrA[55:48] + vfrB[55:48])
vfrD[63:56] <- sat8u(vfrA[63:56] + vfrB[63:56])
```

### Exceptions:

None

Instruction Class	Implementation
ORV DX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x37
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.addus.h rD,rA,rB
```

**Description:**

The unsigned half-word elements of vector/floating-point register vfrA are added to the unsigned half-word elements of vector/floating-point register vfrB to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat16s(vfrA[15:0] + vfrB[15:0])
vfrD[31:16] <- sat16s(vfrA[31:16] + vfrB[31:16])
vfrD[47:32] <- sat16s(vfrA[47:32] + vfrB[47:32])
vfrD[63:48] <- sat16s(vfrA[63:48] + vfrB[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x10
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_eq.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all corresponding elements are equal; otherwise the compare flag is cleared.  
The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] == vfrB[7:0]
vfrA[15:8] == vfrB[15:8] &&
vfrA[23:16] == vfrB[23:16] &&
vfrA[31:24] == vfrB[31:24] &&
vfrA[39:32] == vfrB[39:32] &&
vfrA[47:40] == vfrB[47:40] &&
vfrA[55:48] == vfrB[55:48] &&
vfrA[63:56] == vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x11
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_eq.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if all corresponding elements are equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] == vfrB[15:0] &&
      vfrA[31:16] == vfrB[31:16] &&
      vfrA[47:32] == vfrB[47:32] &&
      vfrA[63:48] == vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.all\_ge.b

## Vector Byte Elements All Greater Than or Equal To

lv.all\_ge.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x12
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lv.all_ge.b rD,rA,rB
```

### Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are greater than or equal to the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
flag <- vfrA[7:0] >= vfrB[7:0] &&
vfrA[15:8] >= vfrB[15:8] &&
vfrA[23:16] >= vfrB[23:16] &&
vfrA[31:24] >= vfrB[31:24] &&
vfrA[39:32] >= vfrB[39:32] &&
vfrA[47:40] >= vfrB[47:40] &&
vfrA[55:48] >= vfrB[55:48] &&
vfrA[63:56] >= vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.all\_ge.h** Vector Half-Word Elements All Greater Than or Equal To **lv.all\_ge.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x13
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.all_ge.h rD,rA,rB`

**Description:**

All half-word elements of vector/floating-point register `vfrA` are compared to the half-word elements of vector/floating-point register `vfrB`. The compare flag is set if all elements of `vfrA` are greater than or equal to the elements of `vfrB`; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] >= vfrB[15:0] &&
vfrA[31:16] >= vfrB[31:16] &&
vfrA[47:32] >= vfrB[47:32] &&
vfrA[63:48] >= vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.all\_gt.b

## Vector Byte Elements All Greater Than

lv.all\_gt.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x14
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lv.all_gt.b rD,rA,rB
```

### Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are greater than the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
flag <- vfrA[7:0] > vfrB[7:0] &&  
vfrA[15:8] > vfrB[15:8] &&  
vfrA[23:16] > vfrB[23:16] &&  
vfrA[31:24] > vfrB[31:24] &&  
vfrA[39:32] > vfrB[39:32] &&  
vfrA[47:40] > vfrB[47:40] &&  
vfrA[55:48] > vfrB[55:48] &&  
vfrA[63:56] > vfrB[63:56]  
vfrD[63:0] <- repl(flag)
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x15
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_gt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are greater than the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] > vfrB[15:0] &&
      vfrA[31:16] > vfrB[31:16] &&
      vfrA[47:32] > vfrB[47:32] &&
      vfrA[63:48] > vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.all\_le.b

Vector Byte Elements All Less Than or Equal To

lv.all\_le.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x16
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

Format:

```
lv.all_le.b rD,rA,rB
```

Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are less than or equal to the elements of vfrB;otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

32-bit Implementation:

N/A

64-bit Implementation:

```
flag <- vfrA[7:0] <= vfrB[7:0] &&
vfrA[15:8] <= vfrB[15:8] &&
vfrA[23:16] <= vfrB[23:16] &&
vfrA[31:24] <= vfrB[31:24] &&
vfrA[39:32] <= vfrB[39:32] &&
vfrA[47:40] <= vfrB[47:40] &&
vfrA[55:48] <= vfrB[55:48] &&
vfrA[63:56] <= vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.all\_le.h**      **Vector Half-Word Elements All Less Than or Equal To**      **lv.all\_le.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x17
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_le.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are less than or equal to the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] <= vfrB[15:0] &&
vfrA[31:16] <= vfrB[31:16] &&
vfrA[47:32] <= vfrB[47:32] &&
vfrA[63:48] <= vfrB[63:48] vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x18
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_lt.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are less than the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] < vfrB[7:0] &&
vfrA[15:8] < vfrB[15:8] &&
vfrA[23:16] < vfrB[23:16] &&
vfrA[31:24] < vfrB[31:24] &&
vfrA[39:32] < vfrB[39:32] &&
vfrA[47:40] < vfrB[47:40] &&
vfrA[55:48] < vfrB[55:48] &&
vfrA[63:56] < vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x19
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_lt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if all elements of vfrA are less than the elements of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] < vfrB[15:0] &&
      vfrA[31:16] < vfrB[31:16] &&
      vfrA[47:32] < vfrB[47:32] &&
      vfrA[63:48] < vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x1a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_ne.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if all corresponding elements are not equal; otherwise the compare flag is cleared.

The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] != vfrB[7:0] &&
vfrA[15:8] != vfrB[15:8] &&
vfrA[23:16] != vfrB[23:16] &&
vfrA[31:24] != vfrB[31:24] &&
vfrA[39:32] != vfrB[39:32] &&
vfrA[47:40] != vfrB[47:40] &&
vfrA[55:48] != vfrB[55:48] &&
vfrA[63:56] != vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x1b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.all_ne.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if all corresponding elements are not equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] != vfrB[15:0] &&
      vfrA[31:16] != vfrB[31:16] &&
      vfrA[47:32] != vfrB[47:32] &&
      vfrA[63:48] != vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.and

## Vector And

lv.and

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x38
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.and rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are combined with the contents of vector/floating-point register vfrB in a bit-wise logical AND operation. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] AND vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



lv.any\_eq.b

### Vector Byte Elements Any Equal

lv.any\_eq.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x20
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_eq.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any two corresponding elements are equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] == vfrB[7:0] ||
vfrA[15:8] == vfrB[15:8] ||
vfrA[23:16] == vfrB[23:16] ||
vfrA[31:24] == vfrB[31:24] ||
vfrA[39:32] == vfrB[39:32] ||
vfrA[47:40] == vfrB[47:40] ||
vfrA[55:48] == vfrB[55:48] ||
vfrA[63:56] == vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x21
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_eq.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if any two corresponding elements are equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] == vfrB[15:0] ||
vfrA[31:16] == vfrB[31:16] ||
vfrA[47:32] == vfrB[47:32] ||
vfrA[63:48] == vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.any\_ge.b      Vector Byte Elements Any Greater Than or Equal To      lv.any\_ge.b**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x22
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.any\_ge.b rD,rA,rB

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is greater than or equal to the corresponding element of vfrB;otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] >= vfrB[7:0] ||
vfrA[15:8] >= vfrB[15:8] ||
vfrA[23:16] >= vfrB[23:16] ||
vfrA[31:24] >= vfrB[31:24] ||
vfrA[39:32] >= vfrB[39:32] ||
vfrA[47:40] >= vfrB[47:40] ||
vfrA[55:48] >= vfrB[55:48] ||
vfrA[63:56] >= vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.any\_ge.h** Vector Half-Word Elements Any Greater Than or Equal To **lv.any\_ge.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x23
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.any_ge.h rD,rA,rB`

**Description:**

All half-word elements of vector/floating-point register `vfrA` are compared to the half-word elements of vector/floating-point register `vfrB`. The compare flag is set if any element of `vfrA` is greater than or equal to the corresponding element of `vfrB`; otherwise the compare flag is cleared.

The compare flag is replicated into all bit positions of vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] >= vfrB[15:0] ||
vfrA[31:16] >= vfrB[31:16] ||
vfrA[47:32] >= vfrB[47:32] ||
vfrA[63:48] >= vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.any\_gt.b

### Vector Byte Elements Any Greater Than

lv.any\_gt.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x24
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

#### Format:

lv.any\_gt.b rD,rA,rB

#### Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is greater than the corresponding element of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

#### 32-bit Implementation:

N/A

#### 64-bit Implementation:

```
flag <- vfrA[7:0] > vfrB[7:0] ||
vfrA[15:8] > vfrB[15:8] ||
vfrA[23:16] > vfrB[23:16] ||
vfrA[31:24] > vfrB[31:24] ||
vfrA[39:32] > vfrB[39:32] ||
vfrA[47:40] > vfrB[47:40] ||
vfrA[55:48] > vfrB[55:48] ||
vfrA[63:56] > vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

#### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x25
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_gt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is greater than the corresponding element of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] > vfrB[15:0] ||
      vfrA[31:16] > vfrB[31:16] ||
      vfrA[47:32] > vfrB[47:32] ||
      vfrA[63:48] > vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.any\_le.b

### Vector Byte Elements Any Less Than or Equal To

lv.any\_le.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x26
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

#### Format:

lv.any\_le.b rD,rA,rB

#### Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is less than or equal to the corresponding element of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

#### 32-bit Implementation:

N/A

#### 64-bit Implementation:

```

flag <- vfrA[7:0] <= vfrB[7:0] ||
vfrA[15:8] <= vfrB[15:8] ||
vfrA[23:16] <= vfrB[23:16] ||
vfrA[31:24] <= vfrB[31:24] ||
vfrA[39:32] <= vfrB[39:32] ||
vfrA[47:40] <= vfrB[47:40] ||
vfrA[55:48] <= vfrB[55:48] ||
vfrA[63:56] <= vfrB[63:56]
vfrD[63:0] <- repl(flag)

```

#### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.any\_le.h**      **Vector Half-Word Elements Any Less Than or Equal To**      **lv.any\_le.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x27
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.any_le.h rD,rA,rB`

**Description:**

All half-word elements of vector/floating-point register `vfrA` are compared to the half-word elements of vector/floating-point register `vfrB`. The compare flag is set if any element of `vfrA` is less than or equal to the corresponding element of `vfrB`; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] ,= vfrB[15:0] ||
vfrA[31:16] <= vfrB[31:16] ||
vfrA[47:32] <= vfrB[47:32] ||
vfrA[63:48] <= vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x28
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_lt.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is less than the corresponding element of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] < vfrB[7:0] ||
vfrA[15:8] < vfrB[15:8] ||
vfrA[23:16] < vfrB[23:16] ||
vfrA[31:24] < vfrB[31:24] ||
vfrA[39:32] < vfrB[39:32] ||
vfrA[47:40] < vfrB[47:40] ||
vfrA[55:48] < vfrB[55:48] ||
vfrA[63:56] < vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x29
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_lt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if any element of vfrA is less than the corresponding element of vfrB; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] < vfrB[15:0] ||
vfrA[31:16] < vfrB[31:16] ||
vfrA[47:32] < vfrB[47:32] ||
vfrA[63:48] < vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.any\_ne.b

### Vector Byte Elements Any Not Equal

lv.any\_ne.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x2a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_ne.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. The compare flag is set if any two corresponding elements are not equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[7:0] != vfrB[7:0] ||
vfrA[15:8] != vfrB[15:8] ||
vfrA[23:16] != vfrB[23:16] ||
vfrA[31:24] != vfrB[31:24] ||
vfrA[39:32] != vfrB[39:32] ||
vfrA[47:40] != vfrB[47:40] ||
vfrA[55:48] != vfrB[55:48] ||
vfrA[63:56] != vfrB[63:56]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x2b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.any_ne.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. The compare flag is set if any two corresponding elements are not equal; otherwise the compare flag is cleared. The compare flag is replicated into all bit positions of vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
flag <- vfrA[15:0] != vfrB[15:0] ||
vfrA[31:16] != vfrB[31:16] ||
vfrA[47:32] != vfrB[47:32] ||
vfrA[63:48] != vfrB[63:48]
vfrD[63:0] <- repl(flag)
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.avg.b

### Vector Byte Elements Average

lv.avg.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x39
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.avg.b rD,rA,rB

**Description:**

The byte elements of vector/floating-point register vfrA are added to the byte elements of vector/floating-point register vfrB and the sum is shifted right by one to form the result elements. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```

vfrD[7:0] <- (vfrA[7:0] + vfrB[7:0]) » 1
vfrD[15:8] <- (vfrA[15:8] + vfrB[15:8]) » 1
vfrD[23:16] <- (vfrA[23:16] + vfrB[23:16]) » 1
vfrD[31:24] <- (vfrA[31:24] + vfrB[31:24]) » 1
vfrD[39:32] <- (vfrA[39:32] + vfrB[39:32]) » 1
vfrD[47:40] <- (vfrA[47:40] + vfrB[47:40]) » 1
vfrD[55:48] <- (vfrA[55:48] + vfrB[55:48]) » 1
vfrD[63:56] <- (vfrA[63:56] + vfrB[63:56]) » 1

```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x3a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.avg.h rD,rA,rB
```

**Description:**

The half-word elements of vector/floating-point register vfrA are added to the half-word elements of vector/floating-point register vfrB and the sum is shifted right by one to form the result elements. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- (vfrA[15:0] + vfrB[15:0]) » 1
vfrD[31:16] <- (vfrA[31:16] + vfrB[31:16]) » 1
vfrD[47:32] <- (vfrA[47:32] + vfrB[47:32]) » 1
vfrD[63:48] <- (vfrA[63:48] + vfrB[63:48]) » 1
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.cmp\_eq.b

### Vector Byte Elements Compare Equal

lv.cmp\_eq.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x40
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

#### Format:

lv.cmp\_eq.b rD,rA,rB

#### Description:

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the two corresponding compared elements are equal; otherwise the element bits are cleared.

#### 32-bit Implementation:

N/A

#### 64-bit Implementation:

```

vfrD[7:0] <- repl(vfrA[7:0] == vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] == vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] == vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] == vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] == vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] == vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] == vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] == vfrB[63:56])

```

#### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x41
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_eq.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the two corresponding compared elements are equal; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] == vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] == vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] == vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] == vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



**lv.cmp\_ge.b** Vector Byte Elements Compare Greater Than or Equal To **lv.cmp\_ge.b**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x42
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.cmp_ge.b rD,rA,rB`

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is greater than or equal to the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- repl(vfrA[7:0] >= vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] >= vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] >= vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] >= vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] >= vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] >= vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] >= vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] >= vfrB[63:56])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.cmp\_ge.h Vector Half-Word Elements Compare Greater Than or Equal To lv.cmp\_ge.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x43
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.cmp\_ge.h rD,rA,rB

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is greater than or equal to the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] >= vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] >= vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] >= vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] >= vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.cmp\_gt.b

### Vector Byte Elements Compare Greater Than

lv.cmp\_gt.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x44
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_gt.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is greater than the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```

vfrD[7:0] <- repl(vfrA[7:0] > vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] > vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] > vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] > vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] > vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] > vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] > vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] > vfrB[63:56])

```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x45
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_gt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is greater than the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] > vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] > vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] > vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] > vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.cmp\_le.b      Vector Byte Elements Compare Less Than or Equal To      lv.cmp\_le.b**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x46
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.cmp_le.b rD,rA,rB`

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is less than or equal to the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- repl(vfrA[7:0] <= vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] <= vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] <= vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] <= vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] <= vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] <= vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] <= vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] <= vfrB[63:56])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.cmp\_le.h Vector Half-Word Elements Compare Less Than or Equal To lv.cmp\_le.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x47
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.cmp\_le.h rD,rA,rB

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is less than or equal to the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] <= vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] <= vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] <= vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] <= vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x48
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_lt.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is less than the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- repl(vfrA[7:0] <= vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] <= vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] <= vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] <= vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] <= vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] <= vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] <= vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] <= vfrB[63:56])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x49
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_lt.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the element in vfrA is less than the element in vfrB; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] <= vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] <= vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] <= vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] <= vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



lv.cmp\_ne.b

### Vector Byte Elements Compare Not Equal

lv.cmp\_ne.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x4a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_ne.b rD,rA,rB
```

**Description:**

All byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the two corresponding compared elements are not equal; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```

vfrD[7:0] <- repl(vfrA[7:0] != vfrB[7:0])
vfrD[15:8] <- repl(vfrA[15:8] != vfrB[15:8])
vfrD[23:16] <- repl(vfrA[23:16] != vfrB[23:16])
vfrD[31:24] <- repl(vfrA[31:24] != vfrB[31:24])
vfrD[39:32] <- repl(vfrA[39:32] != vfrB[39:32])
vfrD[47:40] <- repl(vfrA[47:40] != vfrB[47:40])
vfrD[55:48] <- repl(vfrA[55:48] != vfrB[55:48])
vfrD[63:56] <- repl(vfrA[63:56] != vfrB[63:56])

```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x4b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.cmp_ne.h rD,rA,rB
```

**Description:**

All half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB. Bits of the element in vector/floating-point register vfrD are set if the two corresponding compared elements are not equal; otherwise the element bits are cleared.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- repl(vfrA[7:0] != vfrB[7:0])
vfrD[31:16] <- repl(vfrA[23:16] != vfrB[23:16])
vfrD[47:32] <- repl(vfrA[39:32] != vfrB[39:32])
vfrD[63:48] <- repl(vfrA[55:48] != vfrB[55:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.cust2

Reserved for Custom Vector Instructions

lv.cust2

31 . . . . . 26	25 . . . . . 8	7 . . . . . 4	3 . . . . . 0
opcode 0xa	reserved	opcode 0xd	reserved
6 bits	18 bits	4 bits	4bits

**Format:**

lv.cust2

**Description:**

This fake instruction only allocates instruction set space for custom instructions. Custom instructions are those that are not defined by the architecture, but instead by the implementation itself.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

N/A

**Exceptions:**

N/A

Instruction Class	Implementation
ORVDX64 II	Optional





**lv.madds.h Vector Half-Word Elements Multiply Add Signed Saturated lv.madds.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x54
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.madds.h rD,rA,rB

**Description:**

The signed half-word elements of vector/floating-point register vfrA are multiplied by the signed half-word elements of vector/floating-point register vfrB to form intermediate results. They are then added to the signed half-word VMAC elements to form the final results that are placed again in the VMAC registers. The intermediate result is placed into vector/floating-point register vfrD. If any of the final results exceeds the min/max value, it is saturated.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat32s(vfrA[15:0] * vfrB[15:0] + VMACLO[31:0])
vfrD[31:16] <- sat32s(vfrA[31:16] * vfrB[31:16] + VMACLO[63:32])
vfrD[47:32] <- sat32s(vfrA[47:32] * vfrB[47:32] + VMACHI[31:0])
vfrD[63:48] <- sat32s(vfrA[63:48] * vfrB[63:48] + VMACHI[63:32])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.max.b

## Vector Byte Elements Maximum

lv.max.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x55
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.max.b rD,rA,rB

### Description:

The byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB and the larger elements are selected to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] > vfrB[7:0] ? vfrA[7:0] : vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] > vfrB[15:8] ? vfrA[15:8] : vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] > vfrB[23:16] ? vfrA[23:16] : vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] > vfrB[31:24] ? vfrA[31:24] : vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] > vfrB[39:32] ? vfrA[39:32] : vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] > vfrB[47:40] ? vfrA[47:40] : vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] > vfrB[55:48] ? vfrA[55:48] : vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] > vfrB[63:56] ? vfrA[63:56] : vfrB[63:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.max.h

## Vector Half-Word Elements Maximum

lv.max.h

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x56
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.max.h rD,rA,rB

### Description:

The half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB and the larger elements are selected to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] > vfrB[15:0] ? vfrA[15:0] : vfrB[15:0]
vfrD[31:16] <- vfrA[31:16] > vfrB[31:16] ? vfrA[31:16] : vfrB[31:16]
vfrD[47:32] <- vfrA[47:32] > vfrB[47:32] ? vfrA[47:32] : vfrB[47:32]
vfrD[63:48] <- vfrA[63:48] > vfrB[63:48] ? vfrA[63:48] : vfrB[63:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.merge.b

## Vector Byte Elements Merge

lv.merge.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x57
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.merge.b rD,rA,rB

### Description:

The byte elements of the lower half of the vector/floating-point register vfrA are combined with the byte elements of the lower half of vector/floating-point register vfrB in such a way that lowest element is from vfrB, the second element from vfrA, the third again from vfrB etc. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrB[7:0]
vfrD[15:8] <- vfrA[15:8]
vfrD[23:16] <- vfrB[23:16]
vfrD[31:24] <- vfrA[31:24]
vfrD[39:32] <- vfrB[39:32]
vfrD[47:40] <- vfrA[47:40]
vfrD[55:48] <- vfrB[55:48]
vfrD[63:56] <- vfrA[63:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORV DX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x58
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.merge.h rD,rA,rB
```

**Description:**

The half-word elements of the lower half of the vector/floating-point register vfrA are combined with the half-word elements of the lower half of vector/floating-point register vfrB in such a way that lowest element is from vfrB, the second element from vfrA, the third again from vfrB etc. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- vfrB[15:0]
vfrD[31:16] <- vfrA[31:16]
vfrD[47:32] <- vfrB[47:32]
vfrD[63:48] <- vfrA[63:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.min.b

## Vector Byte Elements Minimum

lv.min.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x59
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.min.b rD,rA,rB

### Description:

The byte elements of vector/floating-point register vfrA are compared to the byte elements of vector/floating-point register vfrB and the smaller elements are selected to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] < vfrB[7:0] ? vfrA[7:0] : vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] < vfrB[15:8] ? vfrA[15:8] : vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] < vfrB[23:16] ? vfrA[23:16] : vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] < vfrB[31:24] ? vfrA[31:24] : vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] < vfrB[39:32] ? vfrA[39:32] : vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] < vfrB[47:40] ? vfrA[47:40] : vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] < vfrB[55:48] ? vfrA[55:48] : vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] < vfrB[63:56] ? vfrA[63:56] : vfrB[63:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.min.h rD,rA,rB
```

**Description:**

The half-word elements of vector/floating-point register vfrA are compared to the half-word elements of vector/floating-point register vfrB and the smaller elements are selected to form the result elements. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- vfrA[15:0] < vfrB[15:0] ? vfrA[15:0] : vfrB[15:0]
vfrD[31:16] <- vfrA[31:16] < vfrB[31:16] ? vfrA[31:16] : vfrB[31:16]
vfrD[47:32] <- vfrA[47:32] < vfrB[47:32] ? vfrA[47:32] : vfrB[47:32]
vfrD[63:48] <- vfrA[63:48] < vfrB[63:48] ? vfrA[63:48] : vfrB[63:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.msubs.h Vector Half-Word Elements Multiply Subtract Signed Saturated lv.msubs.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.msubs.h rD,rA,rB

**Description:**

The signed half-word elements of vector/floating-point register vfrA are multiplied by the signed half-word elements of vector/floating-point register vfrB to form intermediate results. They are then subtracted from the signed half-word VMAC elements to form the final results that are placed again in the VMAC registers. The intermediate result is placed into vector/floating-point register vfrD. If any of the final results exceeds the min/max value, it is saturated.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat32s(VMACLO[31:0] - vfrA[15:0] * vfrB[15:0])
vfrD[31:16] <- sat32s(VMACLO[63:32] - vfrA[31:16] * vfrB[31:16])
vfrD[47:32] <- sat32s(VMACHI[31:0] - vfrA[47:32] * vfrB[47:32])
vfrD[63:48] <- sat32s(VMACHI[63:32] - vfrA[63:48] * vfrB[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.muls.h

## Vector Half-Word Elements Multiply Signed Saturated

lv.muls.h

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5c
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lv.muls.h rD,rA,rB
```

### Description:

The signed half-word elements of vector/floating-point register vfrA are multiplied by the signed half-word elements of vector/floating-point register vfrB to form the results. The result is placed into vector/floating-point register vfrD. If any of the final results exceeds the min/max value, it is saturated.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- sat32s(vfrA[15:0] * vfrB[15:0])  
vfrD[31:16] <- sat32s(vfrA[31:16] * vfrB[31:16])  
vfrD[47:32] <- sat32s(vfrA[47:32] * vfrB[47:32])  
vfrD[63:48] <- sat32s(vfrA[63:48] * vfrB[63:48])
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 II	Optional

---

lv.nand

Vector Not And

lv.nand

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5d
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.nand rD,rA,rB

**Description:**

The contents of vector/floating-point register vfrA are combined with the contents of vector/floating-point register vfrB in a bit-wise logical NAND operation. The result is placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

vfrD[63:0] <- vfrA[63:0] NAND vfrB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.nor

## Vector Not Or

lv.nor

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5e
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.nor rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are combined with the contents of vector/floating-point register vfrB in a bit-wise logical NOR operation. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] NOR vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.or

**Vector Or**

lv.or

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x5f
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

lv.or rD,rA,rB

**Description:**

The contents of vector/floating-point register vfrA are combined with the contents of vector/floating-point register vfrB in a bit-wise logical OR operation. The result is placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

vfrD[63:0] <- vfrA[63:0] OR vfrB[63:0]

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x60
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.pack.b rD,rA,rB
```

**Description:**

The lower half of the byte elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the byte truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[3:0] <- vfrB[3:0]
vfrD[7:4] <- vfrB[11:8]
vfrD[11:8] <- vfrB[19:16]
vfrD[15:12] <- vfrB[27:24]
vfrD[19:16] <- vfrB[35:32]
vfrD[23:20] <- vfrB[43:40]
vfrD[27:24] <- vfrB[51:48]
vfrD[31:28] <- vfrB[59:56]
vfrD[35:32] <- vfrA[3:0]
vfrD[39:36] <- vfrA[11:8]
vfrD[43:40] <- vfrA[19:16]
vfrD[47:44] <- vfrA[27:24]
vfrD[51:48] <- vfrA[35:32]
vfrD[55:52] <- vfrA[43:40]
vfrD[59:56] <- vfrA[51:48]
vfrD[63:60] <- vfrA[59:56]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x61
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.pack.h rD,rA,rB
```

**Description:**

The lower half of the half-word elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the half-word truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- vfrB[15:0]
vfrD[15:8] <- vfrB[31:16]
vfrD[23:16] <- vfrB[47:32]
vfrD[31:24] <- vfrB[63:48]
vfrD[39:32] <- vfrA[15:0]
vfrD[47:40] <- vfrA[31:16]
vfrD[55:48] <- vfrA[47:32]
vfrD[63:56] <- vfrA[63:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x62
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.packs.b rD,rA,rB
```

**Description:**

The lower half of the signed byte elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the signed byte truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. If any truncated element exceeds a signed 4-bit value, it is saturated. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[3:0] <- sat4s(vfrB[7:0])
vfrD[7:4] <- sat4s(vfrB[15:8])
vfrD[11:8] <- sat4s(vfrB[23:16])
vfrD[15:12] <- sat4s(vfrB[31:24])
vfrD[19:16] <- sat4s(vfrB[39:32])
vfrD[23:20] <- sat4s(vfrB[47:40])
vfrD[27:24] <- sat4s(vfrB[55:48])
vfrD[31:28] <- sat4s(vfrB[63:56])
vfrD[35:32] <- sat4s(vfrA[7:0])
vfrD[39:36] <- sat4s(vfrA[15:8])
vfrD[43:40] <- sat4s(vfrA[23:16])
vfrD[47:44] <- sat4s(vfrA[31:24])
vfrD[51:48] <- sat4s(vfrA[39:32])
vfrD[55:52] <- sat4s(vfrA[47:40])
vfrD[59:56] <- sat4s(vfrA[55:48])
vfrD[63:60] <- sat4s(vfrA[63:56])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x63
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.packs.h rD,rA,rB
```

**Description:**

The lower half of the signed halfword elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the signed half-word truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. If any truncated element exceeds a signed 8-bit value, it is saturated. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- sat8s(vfrB[15:0])
vfrD[15:8] <- sat8s(vfrB[31:16])
vfrD[23:16] <- sat8s(vfrB[47:32])
vfrD[31:24] <- sat8s(vfrB[63:48])
vfrD[39:32] <- sat8s(vfrA[15:0])
vfrD[47:40] <- sat8s(vfrA[31:16])
vfrD[55:48] <- sat8s(vfrA[47:32])
vfrD[63:56] <- sat8s(vfrA[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x64
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.packus.b rD,rA,rB
```

**Description:**

The lower half of the unsigned byte elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the unsigned byte truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. If any truncated element exceeds a unsigned 4-bit value, it is saturated. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[3:0] <- sat4u(vfrB[7:0])
vfrD[7:4] <- sat4u(vfrB[15:8])
vfrD[11:8] <- sat4u(vfrB[23:16])
vfrD[15:12] <- sat4u(vfrB[31:24])
vfrD[19:16] <- sat4u(vfrB[39:32])
vfrD[23:20] <- sat4u(vfrB[47:40])
vfrD[27:24] <- sat4u(vfrB[55:48])
vfrD[31:28] <- sat4u(vfrB[63:56])
vfrD[35:32] <- sat4u(vfrA[7:0])
vfrD[39:36] <- sat4u(vfrA[15:8])
vfrD[43:40] <- sat4u(vfrA[23:16])
vfrD[47:44] <- sat4u(vfrA[31:24])
vfrD[51:48] <- sat4u(vfrA[39:32])
vfrD[55:52] <- sat4u(vfrA[47:40])
vfrD[59:56] <- sat4u(vfrA[55:48])
vfrD[63:60] <- sat4u(vfrA[63:56])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x65
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.packus.h rD,rA,rB
```

**Description:**

The lower half of the unsigned halfword elements of the vector/floating-point register vfrA are truncated and combined with the lower half of the unsigned half-word truncated elements of the vector/floating-point register vfrB in such a way that lowest elements are from vfrB and highest elements from vfrA. If any truncated element exceeds an unsigned 8-bit value, it is saturated. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- sat8u(vfrB[15:0])
vfrD[15:8] <- sat8u(vfrB[31:16])
vfrD[23:16] <- sat8u(vfrB[47:32])
vfrD[31:24] <- sat8u(vfrB[63:48])
vfrD[39:32] <- sat8u(vfrA[15:0])
vfrD[47:40] <- sat8u(vfrA[31:16])
vfrD[55:48] <- sat8u(vfrA[47:32])
vfrD[63:56] <- sat8u(vfrA[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.perm.n

## Vector Nibble Elements Permute

lv.perm.n

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x66
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.perm.n rD,rA,rB

### Description:

The 4-bit elements of vector/floating-point register vfrA are permuted according to corresponding 4-bit values in vector/floating-point register vfrB. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[3:0] <- vfrA[vfrB[3:0]*4+3:vfrB[3:0]*4]
vfrD[7:4] <- vfrA[vfrB[7:4]*4+3:vfrB[7:4]*4]
vfrD[11:8] <- vfrA[vfrB[11:8]*4+3:vfrB[11:8]*4]
vfrD[15:12] <- vfrA[vfrB[15:12]*4+3:vfrB[15:12]*4]
vfrD[19:16] <- vfrA[vfrB[19:16]*4+3:vfrB[19:16]*4]
vfrD[23:20] <- vfrA[vfrB[23:20]*4+3:vfrB[23:20]*4]
vfrD[27:24] <- vfrA[vfrB[27:24]*4+3:vfrB[27:24]*4]
vfrD[31:28] <- vfrA[vfrB[31:28]*4+3:vfrB[31:28]*4]
vfrD[35:32] <- vfrA[vfrB[35:32]*4+3:vfrB[35:32]*4]
vfrD[39:36] <- vfrA[vfrB[39:36]*4+3:vfrB[39:36]*4]
vfrD[43:40] <- vfrA[vfrB[43:40]*4+3:vfrB[43:40]*4]
vfrD[47:44] <- vfrA[vfrB[47:44]*4+3:vfrB[47:44]*4]
vfrD[51:48] <- vfrA[vfrB[51:48]*4+3:vfrB[51:48]*4]
vfrD[55:52] <- vfrA[vfrB[55:52]*4+3:vfrB[55:52]*4]
vfrD[59:56] <- vfrA[vfrB[59:56]*4+3:vfrB[59:56]*4]
vfrD[63:60] <- vfrA[vfrB[63:60]*4+3:vfrB[63:60]*4]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.rl.b

## Vector Byte Elements Rotate Left

lv.rl.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x67
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.rl.b rD,rA,rB

### Description:

The contents of byte elements of vector/floating-point register vfrA are rotated left by the number of bits specified in lower 3 bits in each byte element of vector/floating-point register vfrB. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] r1 vfrB[2:0]
vfrD[15:8] <- vfrA[15:8] r1 vfrB[10:8]
vfrD[23:16] <- vfrA[23:16] r1 vfrB[18:16]
vfrD[31:24] <- vfrA[31:24] r1 vfrB[26:24]
vfrD[39:32] <- vfrA[39:32] r1 vfrB[34:32]
vfrD[47:40] <- vfrA[47:40] r1 vfrB[42:40]
vfrD[55:48] <- vfrA[55:48] r1 vfrB[50:48]
vfrD[63:56] <- vfrA[63:56] r1 vfrB[58:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.rl.h

## Vector Half-Word Elements Rotate Left

lv.rl.h

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x68
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.rl.h rD,rA,rB

### Description:

The contents of half-word elements of vector/floating-point register vfrA are rotated left by the number of bits specified in lower 4 bits in each half-word element of vector/floating-point register vfrB. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] r1 vfrB[3:0]
vfrD[31:16] <- vfrA[31:16] r1 vfrB[19:16]
vfrD[47:32] <- vfrA[47:32] r1 vfrB[35:32]
vfrD[63:48] <- vfrA[63:48] r1 vfrB[51:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.sll

## Vector Shift Left Logical

lv.sll

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.sll rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are shifted left by the number of bits specified in lower 4 bits in each byte element of vector/floating-point register vfrB, inserting zeros into the low-order bits of vfrD. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] << vfrB[2:0]

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.sll.b

## Vector Byte Elements Shift Left Logical

lv.sll.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x69
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.sll.b rD,rA,rB

### Description:

The contents of byte elements of vector/floating-point register vfrA are shifted left by the number of bits specified in lower 3 bits in each byte element of vector/floating-point register vfrB, inserting zeros into the low-order bits. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] << vfrB[2:0]
vfrD[15:8] <- vfrA[15:8] << vfrB[10:8]
vfrD[23:16] <- vfrA[23:16] << vfrB[18:16]
vfrD[31:24] <- vfrA[31:24] << vfrB[26:24]
vfrD[39:32] <- vfrA[39:32] << vfrB[34:32]
vfrD[47:40] <- vfrA[47:40] << vfrB[42:40]
vfrD[55:48] <- vfrA[55:48] << vfrB[50:48]
vfrD[63:56] <- vfrA[63:56] << vfrB[58:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.sll.h rD,rA,rB
```

**Description:**

The contents of half-word elements of vector/floating-point register vfrA are shifted left by the number of bits specified in lower 4 bits in each half-word element of vector/floating-point register vfrB, inserting zeros into the low-order bits. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- vfrA[15:0] << vfrB[3:0]
vfrD[31:16] <- vfrA[31:16] << vfrB[19:16]
vfrD[47:32] <- vfrA[47:32] << vfrB[35:32]
vfrD[63:48] <- vfrA[63:48] << vfrB[51:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.sra.b

## Vector Byte Elements Shift Right Arithmetic

lv.sra.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6e
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.sra.b rD,rA,rB

### Description:

The contents of byte elements of vector/floating-point register vfrA are shifted right by the number of bits specified in lower 3 bits in each byte element of vector/floating-point register vfrB, inserting the most significant bit of each element into the high-order bits. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] sra vfrB[2:0]
vfrD[15:8] <- vfrA[15:8] sra vfrB[10:8]
vfrD[23:16] <- vfrA[23:16] sra vfrB[18:16]
vfrD[31:24] <- vfrA[31:24] sra vfrB[26:24]
vfrD[39:32] <- vfrA[39:32] sra vfrB[34:32]
vfrD[47:40] <- vfrA[47:40] sra vfrB[42:40]
vfrD[55:48] <- vfrA[55:48] sra vfrB[50:48]
vfrD[63:56] <- vfrA[63:56] sra vfrB[58:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORV DX64 I	Required

---

lv.sra.h

## Vector Half-Word Elements Shift Right Arithmetic

lv.sra.h

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6f
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lv.sra.h rD,rA,rB
```

### Description:

The contents of half-word elements of vector/floating-point register vfrA are shifted right by the number of bits specified in lower 4 bits in each half-word element of vector/floating-point register vfrB, inserting the most significant bit of each element into the high-order bits. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] sra vfrB[3:0]  
vfrD[31:16] <- vfrA[31:16] sra vfrB[19:16]  
vfrD[47:32] <- vfrA[47:32] sra vfrB[35:32]  
vfrD[63:48] <- vfrA[63:48] sra vfrB[51:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.srl

## Vector Shift Right Logical

lv.srl

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x70
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.srl rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are shifted right by the number of bits specified in lower 4 bits in each byte element of vector/floating-point register vfrB, inserting zeros into the high-order bits of vfrD. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] > vfrB[2:0]

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.srl.b

## Vector Byte Elements Shift Right Logical

lv.srl.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6c
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.srl.b rD,rA,rB

### Description:

The contents of byte elements of vector/floating-point register vfrA are shifted right by the number of bits specified in lower 3 bits in each byte element of vector/floating-point register vfrB, inserting zeros into the high-order bits. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] > vfrB[2:0]
vfrD[15:8] <- vfrA[15:8] > vfrB[10:8]
vfrD[23:16] <- vfrA[23:16] > vfrB[18:16]
vfrD[31:24] <- vfrA[31:24] > vfrB[26:24]
vfrD[39:32] <- vfrA[39:32] > vfrB[34:32]
vfrD[47:40] <- vfrA[47:40] > vfrB[42:40]
vfrD[55:48] <- vfrA[55:48] > vfrB[50:48]
vfrD[63:56] <- vfrA[63:56] > vfrB[58:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x6d
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.srl.h rD,rA,rB
```

**Description:**

The contents of half-word elements of vector/floating-point register vfrA are shifted right by the number of bits specified in lower 4 bits in each half-word element of vector/floating-point register vfrB, inserting zeros into the high-order bits. The result elements are placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- vfrA[15:0] » vfrB[3:0]
vfrD[31:16] <- vfrA[31:16] » vfrB[19:16]
vfrD[47:32] <- vfrA[47:32] » vfrB[35:32]
vfrD[63:48] <- vfrA[63:48] » vfrB[51:48]
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.sub.b

## Vector Byte Elements Subtract Signed

lv.sub.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x71
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.sub.b rD,rA,rB

### Description:

The byte elements of vector/floating-point register vfrB are subtracted from the byte elements of vector/floating-point register vfrA to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```

vfrD[7:0] <- vfrA[7:0] - vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] - vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] - vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] - vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] - vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] - vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] - vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] - vfrB[63:56]

```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required



---

lv.sub.h

## Vector Half-Word Elements Subtract Signed

lv.sub.h

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x72
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.sub.h rD,rA,rB

### Description:

The half-word elements of vector/floating-point register vfrB are subtracted from the half-word elements of vector/floating-point register vfrA to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] - vfrB[15:0]
vfrD[31:16] <- vfrA[31:16] - vfrB[31:16]
vfrD[47:32] <- vfrA[47:32] - vfrB[47:32]
vfrD[63:48] <- vfrA[63:48] - vfrB[63:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

lv.subs.b

## Vector Byte Elements Subtract Signed Saturated

lv.subs.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x73
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.subs.b rD,rA,rB

### Description:

The byte elements of vector/floating-point register vfrB are subtracted from the byte elements of vector/floating-point register vfrA to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```

vfrD[7:0] <- sat8s(vfrA[7:0] + vfrB[7:0])
vfrD[15:8] <- sat8s(vfrA[15:8] + vfrB[15:8])
vfrD[23:16] <- sat8s(vfrA[23:16] + vfrB[23:16])
vfrD[31:24] <- sat8s(vfrA[31:24] + vfrB[31:24])
vfrD[39:32] <- sat8s(vfrA[39:32] + vfrB[39:32])
vfrD[47:40] <- sat8s(vfrA[47:40] + vfrB[47:40])
vfrD[55:48] <- sat8s(vfrA[55:48] + vfrB[55:48])
vfrD[63:56] <- sat8s(vfrA[63:56] + vfrB[63:56])

```

### Exceptions:

None

Instruction Class	Implementation
ORV DX64 I	Required

**lv.subs.h**      **Vector Half-Word Elements Subtract Signed Saturated**      **lv.subs.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x74
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.subs.h rD,rA,rB`

**Description:**

The half-word elements of vector/floating-point register `vfrB` are subtracted from the half-word elements of vector/floating-point register `vfrA` to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat16s(vfrA[15:0] - vfrB[15:0])
vfrD[31:16] <- sat16s(vfrA[31:16] - vfrB[31:16])
vfrD[47:32] <- sat16s(vfrA[47:32] - vfrB[47:32])
vfrD[63:48] <- sat16s(vfrA[63:48] - vfrB[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.subu.b

## Vector Byte Elements Subtract Unsigned

lv.subu.b

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x75
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.subu.b rD,rA,rB

### Description:

The unsigned byte elements of vector/floating-point register vfrB are subtracted from the unsigned byte elements of vector/floating-point register vfrA to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- vfrA[7:0] - vfrB[7:0]
vfrD[15:8] <- vfrA[15:8] - vfrB[15:8]
vfrD[23:16] <- vfrA[23:16] - vfrB[23:16]
vfrD[31:24] <- vfrA[31:24] - vfrB[31:24]
vfrD[39:32] <- vfrA[39:32] - vfrB[39:32]
vfrD[47:40] <- vfrA[47:40] - vfrB[47:40]
vfrD[55:48] <- vfrA[55:48] - vfrB[55:48]
vfrD[63:56] <- vfrA[63:56] - vfrB[63:56]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.subu.h

## Vector Half-Word Elements Subtract Unsigned

lv.subu.h

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x76
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

```
lv.subu.h rD,rA,rB
```

### Description:

The unsigned half-word elements of vector/floating-point register vfrB are subtracted from the unsigned half-word elements of vector/floating-point register vfrA to form the result elements. The result elements are placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[15:0] <- vfrA[15:0] - vfrB[15:0]  
vfrD[31:16] <- vfrA[31:16] - vfrB[31:16]  
vfrD[47:32] <- vfrA[47:32] - vfrB[47:32]  
vfrD[63:48] <- vfrA[63:48] - vfrB[63:48]
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.subus.b

## Vector Byte Elements Subtract Unsigned Saturated

lv.subus.b

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x77
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.subus.b rD,rA,rB

### Description:

The unsigned byte elements of vector/floating-point register vfrB are subtracted from the unsigned byte elements of vector/floating-point register vfrA to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

```
vfrD[7:0] <- sat8u(vfrA[7:0] + vfrB[7:0])
vfrD[15:8] <- sat8u(vfrA[15:8] + vfrB[15:8])
vfrD[23:16] <- sat8u(vfrA[23:16] + vfrB[23:16])
vfrD[31:24] <- sat8u(vfrA[31:24] + vfrB[31:24])
vfrD[39:32] <- sat8u(vfrA[39:32] + vfrB[39:32])
vfrD[47:40] <- sat8u(vfrA[47:40] + vfrB[47:40])
vfrD[55:48] <- sat8u(vfrA[55:48] + vfrB[55:48])
vfrD[63:56] <- sat8u(vfrA[63:56] + vfrB[63:56])
```

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required

**lv.subus.h    Vector Half-Word Elements Subtract Unsigned Saturated    lv.subus.h**

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x78
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

`lv.subus.h rD,rA,rB`

**Description:**

The unsigned half-word elements of vector/floating-point register `vfrB` are subtracted from the unsigned half-word elements of vector/floating-point register `vfrA` to form the result elements. If the result exceeds the min/max value for the destination data type, it is saturated to the min/max value and placed into vector/floating-point register `vfrD`.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- sat16u(vfrA[15:0] - vfrB[15:0])
vfrD[31:16] <- sat16u(vfrA[31:16] - vfrB[31:16])
vfrD[47:32] <- sat16u(vfrA[47:32] - vfrB[47:32])
vfrD[63:48] <- sat16u(vfrA[63:48] - vfrB[63:48])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x79
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.unpack.b rD,rA,rB
```

**Description:**

The lower half of the 4-bit elements in vector/floating-point register vfrA are sign-extended and placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[7:0] <- exts(vfrA[3:0])
vfrD[15:8] <- exts(vfrA[7:4])
vfrD[23:16] <- exts(vfrA[11:8])
vfrD[31:24] <- exts(vfrA[15:12])
vfrD[39:32] <- exts(vfrA[19:16])
vfrD[47:40] <- exts(vfrA[23:20])
vfrD[55:48] <- exts(vfrA[27:24])
vfrD[63:56] <- exts(vfrA[31:28])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required



31 . . . . 26	25 . . . . 21	20 . . . . 16	15 . . . . 11	10 . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x7a
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

**Format:**

```
lv.unpack.h rD,rA,rB
```

**Description:**

The lower half of the 8-bit elements in vector/floating-point register vfrA are sign-extended and placed into vector/floating-point register vfrD.

**32-bit Implementation:**

N/A

**64-bit Implementation:**

```
vfrD[15:0] <- exts(vfrA[7:0])
vfrD[31:16] <- exts(vfrA[15:8])
vfrD[47:32] <- exts(vfrA[23:16])
vfrD[63:48] <- exts(vfrA[31:24])
```

**Exceptions:**

None

Instruction Class	Implementation
ORVDX64 I	Required

---

lv.xor

## Vector Exclusive Or

lv.xor

31 . . . . . 26	25 . . . . . 21	20 . . . . . 16	15 . . . . . 11	10 . . . . . 8	7 . . . . . 0
opcode 0xa	D	A	B	reserved	opcode 0x7b
6 bits	5 bits	5 bits	5 bits	3 bits	8bits

### Format:

lv.xor rD,rA,rB

### Description:

The contents of vector/floating-point register vfrA are combined with the contents of vector/floating-point register vfrB in a bit-wise logical XOR operation. The result is placed into vector/floating-point register vfrD.

### 32-bit Implementation:

N/A

### 64-bit Implementation:

vfrD[63:0] <- vfrA[63:0] XOR vfrB[63:0]

### Exceptions:

None

Instruction Class	Implementation
ORVDX64 I	Required