0.1. ORBIS32/64

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
6 bits	5 bits	5 bits	5 bits	1 bits	2 bits	4 bits	4bits

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] <- 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.addi Add Immediate Signed l.addi

31 26	25 . . . 21	20 . . . 16	15 0
opcode 0x27	D	A	I
6 bits	5 bits	5 bits	16bits

Format:

```
l.addi rD,rA,I
```

Description:

The immediate value is signed-extended and added to the contents of general-purposeregister rA to form the result. The result is placed into general-purposeregister rD.

32-bit Implementation:

```
rD[31:0] <- rA[31:0] + exts(Immediate)
SR[CY] <- carry
SR[OV] <- overflow
```

64-bit Implementation:

```
rD[63:0] <- rA[63:0] + exts(Immediate)
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 rB in a bit-wise logical AND operation. The result is placed into general-purpose register rD.

32-bit Implementation:

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

64-bit Implementation:

```
rD[63:0] <- rB[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 compare 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 compare 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[CF] is set, general-purpose register rA is placed in general-purpose register rD. If SR[CF] is cleared, general-purpose register rB is placed in general-purpose register rD.

32-bit Implementation:

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

64-bit Implementation:

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

Exceptions:

```
None
```

Instruction Class	Implementation
ORBIS32 II&Optional	

l.csync — Context Syncronization — l.csync

31 0
opcode 0x23000000
32bits

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 divide by zero flag is set when the divisor is zero.

32-bit Implementation:

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

64-bit Implementation:

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

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 divide by zero flag is set when the divisor is zero.

32-bit Implementation:

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

64-bit Implementation:

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

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	