ANY-1 Instruction Set

Instruction Formats

Immediate Format:

63 32	3130	2928	27 24	23 16	15 8	7 0
Constant$_{32}$	~$_2$	U$_2$	Sz$_4$	Ra$_8$	Rt$_8$	09h$_8$

Register Format:

SR1 (one source register)

63 61	6058	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	01h$_8$	U$_2$	Sz$_4$	m$_3$	z	~$_8$	Func$_8$	Ra$_8$	Rt$_8$	03h$_8$

SR2 (two source register)

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	01h$_8$	U$_2$	Sz$_4$	m$_3$	z	Func$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

SR3 (three source register)

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	Func$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

SR4 (four source register)

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	Rd$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

z: 1 = zero vector element if mask bit clear, 0 = vector element unchanged (ignored for scalar ops)
m$_3$: vector mask register (ignored for scalar operations).
Rm$_3$: rounding mode
If any of Rt, Ra, Rb, Rc are vector registers, then the instruction is a vector instruction.

Rn$_8$	
0 to 63	scalar registers
64 to 127	vector registers
128 to 255	Rn is a seven bit constant

U$_2$	Execution Unit	Qualifier
0	Integer	.int
1	Floating-point	.fp
2	Decimal floating-point	.dfp
3	Posit	.pos

Sz$_4$	Size	Qualifier	Alt Qualifier
0	byte	.b	
1	wyde	.w	
2	tetra	.t	.s (single)
3	octa	.o	.d (double)
4	hexi	.h	.q (quad)

8	SIMD byte	.bp	
9	SIMD wyde	.wp	
10	SIMD tetra	.tp	.sp
11	SIMD octa	.op	.dp
12	SIMD hexi	.hp	.qp

Example Instruction

```
add.int.o  x1,x2,x3,x0    ; scalar add of integers x2,x3

add.int.o v1,v2,v3,v0     ; vector add of integers v2,v3

add.int.o v1,v2,v0,x4     ; vector add scalar integers v2,x4

add.fp.o v1,v2,v3,v0      ; vector add float-point double v2,v3
```

Instructions

Arithmetic / Logical

ABS – Absolute Value

Description:

This instruction takes the absolute value of a register and places the result in a target register.

Instruction Format: SR1

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$01h_8$	U_2	Sz_4	m_3	z	$\sim_8$	4_8	Ra_8	Rt_8	$03h_8$

Operation:

```
If Ra < 0
   Rt = -Ra
else
   Rt = Ra
```

Vector Operation

```
for x = 0 to VL - 1
          if (Vm[x]) Rt[x] = Ra[x] < 0 ? -Ra[x] : Ra[x]
```

Exceptions: none

Notes:

For sign-magnitude formats this instruction simply clears the MSB of the number. No rounding occurs.

ADD - Addition

Description:

Add two values. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Operation:

Rt = Ra + Imm

or

Rt = Ra + Rb + Rc

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = Va[x] + Vb[x] + Vc[x]
    else if (z) Vt[x] = 0
```

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
Constant$_{32}$	~$_2$	U$_2$	Sz$_4$	Ra$_8$	Rt$_8$	04h$_8$

Register Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	4$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

Exceptions: none

ADDIS – Add Immediate Shifted

Description:

Perform an addition operation between operands. The immediate constant is shifted left by a multiple of 32 bits and sign extended to the left and zero extended to the right before use.

Immediate Instruction Format

63 32	31 28	27 24	23 16	15 8	7 0
Constant$_{32}$	Fn$_4$	Sh$_4$	Ra$_8$	Rt$_8$	Opcode$_8$

63 32	31 28	27 24	23 16	15 8	7 0
Constant$_{32}$	4$_4$	Sh$_4$	Ra$_8$	Rt$_8$	Opcode$_8$

Operation

Rt = Ra + (Immediate << (32 * Sh))

Exceptions: none

AND – Bitwise And

Description:

Perform a bitwise 'and' operation between operands. The first operand must be in a register. The second operand may be in a register of may be an immediate value specified in the instruction. A third source operand must be in a register. The immediate constant is one extended before use.

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
$Constant_{32}$	$\sim_2$	$\sim_2$	Sz_4	Ra_8	Rt_8	$08h_8$

Register Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	8_8	$\sim_2$	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$03h_8$

Operation:

Rt = Ra & Imm

or

Rt = Ra & Rb & Rc

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = Va[x] & Vb[x] & Vc[x]
    else if (z) Vt[x] = 0
```

Exceptions: none

ANDIS – Bitwise And Immediate Shifted

Description:

Perform a bitwise and operation between operands. The immediate constant is shifted left a multiple of 32 bits and one extended to the left and right before use.

Immediate Instruction Format

63 32	31 28	27 24	23 16	15 8	7 0
Constant$_{32}$	8_4	Sh_4	Ra_8	Rt_8	$Opcode_8$

Operation

Rt = Ra & ((Immediate << (32*Sh2)) | 0xFFFFFFFF)

Exceptions: none

AISPC – Add Immediate Shifted to PC

Description:

This instruction forms the sum of the program counter and an immediate value shifted left a multiple of 32 times. The result is then placed in the target register. The low order 32 bits of the target register are zeroed out.

Instruction Format

63 32	31 28	27 24	23 16	15 8	7 0
Constant$_{32}$	F$_4$	Sh$_4$	63$_8$	Rt$_8$	Opcode$_8$

Exceptions: none

BMM – Bit Matrix Multiply

BMM Rt, Ra, Rb

Description:

The BMM instruction treats the bits of register Ra and register Rb as an 8x8 matrix and performs a bit matrix multiply of the two registers and stores the result in the target register. An alternate mnemonic for this instruction is MOR.

Instruction Format: S2

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Fn_3	Rm_3	$03h_8$	U_2	Sz_4	m_3	z	$\sim_8$	Rb_8	Ra_8	Rt_8	$03h_8$

Fn_3	Function
0	MOR
1	MXOR
2	MORT (MOR transpose)
3	MXORT (MXOR transpose)
4 to 7	reserved

Operation:

```
for I = 0 to 7
        for j = 0 to 7
                Rt.bit[i][j] = (Ra[i][0]&Rb[0][j]) | (Ra[i][1]&Rb[1][j]) | … | (Ra[i][15]&Rb[15][j])
```

Clock Cycles: 1

Execution Units: Integer ALU

Exceptions: none

Notes:

The bits are numbered with bit 63 of a register representing I,j = 0,0 and bit 0 of the register representing I,j = 7,7.

BYTNDX – Byte Index

Description:

This instruction searches Ra, which is treated as an array of eight bytes, for a byte value specified by Rb or an immediate value and places the index of the byte into the target register Rt. If the byte is not found -1 is placed in the target register. A common use would be to search for a null byte. The index result may vary from -1 to +7. The index of the first found byte is returned (closest to zero).

Instruction Format: SR2

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
0_3	Rm_3	$\sim_8$	0_2	Sz_4	m_3	z	$\sim_8$	Rb_8	Ra_8	Rt_8	$1Ah_8$

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
1_3	Rm_3	$\sim_8$	0_2	Sz_4	m_3	z	$\sim_8$	Imm_8	Ra_8	Rt_8	$1Ah_8$

R2 Supported Formats: .w, .t, .o

Clock Cycles: 1

Execution Units: Integer ALU

Operation:

Rt = Index of (Rb in Ra)

Exceptions: none

CNTPOP – Count Population

Description:

Count the number of ones and place the count in the target register.

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = popcnt(Va[x])
```

Instruction Format:

63 61	6058	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	1_8	0_2	Sz_4	m_3	z	$\sim_8$	2_8	Ra_8	Rt_8	$03h_8$

Execution Units: integer ALU

Exceptions: none

DEP – Deposit

Description:

Insert to a bitfield. Rc specifies the bitfield offset, Rd specifies the width of the bitfield. Rb specifies the data to insert. Ra contains the original source data. The least significant Rd minus one bits of Rb are inserted into Ra at the position specified by Rc. The final result is placed into Rt.

This instruction may also be used to perform a left shift of a single register by specifying x0 for Ra.

Formats Supported: SR3

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	Rd_8	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$1Dh_8$

Operation Size: .o, .t, .w, .b

Execution Units: integer ALU

Exceptions: none

Example:

EXT –Extract Bitfield

Description:

A bitfield is extracted from the source by shifting the source to the right and ‘and’ masking. The result is sign extended to the width of the machine. This instruction may be used to sign extend a value from an arbitrary bit position. The width specified should be one less than the desired width. The source is value is contained in the register pair Ra, Rb. The field width is specified by Rc and field offset by Rd.

Instruction Format: SR4

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	Rd_8	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$2Ch_8$

Execution Units: Integer ALU

Exceptions: none

Notes:

EXTU –Extract Bitfield Unsigned

Description:

A bitfield is extracted from the source by shifting the source to the right and ‘and’ masking. The result is zero extended to the width of the machine. This instruction may be used to zero extend a value from an arbitrary bit position. The width specified should be one less than the desired width.

Instruction Format: SR4

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	Rd$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	24h$_8$

Execution Units: Integer ALU

Exceptions: none

Notes:

FDP – Fused Dot Product

Description:

Calculate the dot product x = (a * b) + (c * d). The operations are fused together meaning no rounding occurs until the final product is produced.

Instruction Format: SR4

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	Rd$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	37h$_8$

MAX – Maximum Value

Description:

Determines the maximum of three values in registers Ra, Rb, Rc and places the result in the target register Rt.

Instruction Format

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$Func_8$	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$03h_8$

Operation:

```
IF Ra > Rb and Ra > Rc
        Rt = Ra
else if Rb > Rc
        Rt = Rb
else
        Rt = Rc
```

MIN – Minimum Value

Description:

Determines the minimum of three values in registers Ra, Rb, Rc and places the result in the target register Rt.

Instruction Format

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$Func_8$	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$03h_8$

Operation:

```
IF Ra < Rb and Ra < Rc
        Rt = Ra
else if Rb < Rc
        Rt = Rb
else
        Rt = Rc
```

MUL – Signed Multiply

Description:

Multiply two values. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction. Both the operands are treated as signed values, the result is a signed result.

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = Va[x] * Vb[x]
```

Exceptions: multiply overflow, if enabled

MULF – Fast Unsigned Multiply

Description:

Multiply two values. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction. Both the operands are treated as unsigned values. The result is an unsigned result. The fast multiply multiplies only the low order 24 bits of the first operand times the low order 16 bits of the second. The result is a 40-bit unsigned product.

Exceptions: none

MUX – Multiplex

Description:

The MUX instruction performs a bit-by-bit copy of a bit of Rb to the target register if the corresponding bit in Ra is set, or a copy of a bit from Rc if the corresponding bit in Ra is clear.

Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$1Bh_8$	0_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$03h_8$

Exceptions: none

Execution Units: integer ALU

NEG - Negate

Description:

This is an alternate mnemonic for the SUB instruction where the first register operand is R0.

Vector Operation

```
for x = 0 to VL-1
    if (Vm[x]) Vt[x] = R0 - Va[x]
```

NOT – Logical Not

Description:

This instruction takes the logical ‘not’ value of a register and places the result in a target register. If the source register contains a non-zero value, then a zero is loaded into the target. Otherwise, if the source register contains a zero a one is loaded into the target register.

Register Instruction Format

47 42	4140	39 36	35 33	32	31 26	25 20	19 14	13 8	7	6 0
$01h_6$	0_2	Sz_4	$\sim_3$	~	$\sim_6$	$05h_6$	Ra_6	Rt_6	0	$03h_7$

Operation:

Rt = !Ra

Exceptions: none

OR – Bitwise Or

Description:

Perform a bitwise or operation between operands. The immediate constant is zero extended before use.

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
Constant$_{32}$	~$_2$	0$_2$	Sz$_4$	Ra$_8$	Rt$_8$	09h$_8$

Register Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	09h$_8$	0$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

Operation

Rt = Ra | Immediate

OR

Rt = Ra | Rb | Rc

Vector Operation

```
for x = 0 to VL-1
    if (Vm[x]) Vt[x] = Va[x] | Vb[x] | Vc[x]
```

Exceptions: none

ORIS – Bitwise Or Immediate Shifted

Description:

Perform a bitwise or operation between operands. The immediate constant is shifted left a multiple of 32 bits and zero extended to the left and right before use.

Immediate Instruction Format

63 32	31 28	27 24	23 16	15 8	7 0
Constant$_{32}$	9_4	Sh$_4$	Ra$_8$	Rt$_8$	Opcode$_8$

Operation

Rt = Ra | (Immediate << (32 * Sh2))

Exceptions: none

PERM – Permute Bytes

Description:

This instruction allows any combination of bytes in a source register to be copied to a target register. The low order twenty-four bits of register Rb or a 12-bit immediate constant are used to identify which source bytes are copied to the destination. The twenty-four-bit value is composed of eight three-bit fields. Field S0 indicates the source byte for target byte position 0. S1 indicates the source byte for target byte position 1. S2 to S7 work similarly for the remaining target bytes. There are many interesting possibilities with this instruction. A single source byte could be copied to all target byte positions for instance. Or the order of bytes in a word could be reversed.

Instruction Format: SR2, PERM

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
0_3	Rm$_3$	~$_8$	0_2	Sz$_4$	m$_3$	z	~$_8$	Rb$_8$	Ra$_8$	Rt$_8$	17h$_8$

63 61	60 58	57 50	49 48	47 44	43 41	40	39 24	23 16	15 8	7 0
1_3	Rm$_3$	Imm$_{23..16}$	0_2	Sz$_4$	m$_3$	z	Imm$_{15..0}$	Ra$_8$	Rt$_8$	17h$_8$

Execution Units: integer ALU

Clock Cycles: 1

Exceptions: none

SEQ – Set if Equal

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is equal to a second operand in register (Rb) or an immediate constant then the target register is set to a one, otherwise the target register is set to a zero.

For floating-point operations positive and negative zero are considered equal.

If a vector operation is taking place then the target register is one of the vector mask registers.

Immediate Instruction Format

63 32	3130	2928	27 24	23 16	15 8	7 0
Constant_{32}	$\sim_2$	U_2	Sz_4	Ra_8	Rt_8	$26h_8$

Register Instruction Format

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$26h_8$	U_2	Sz_4	m_3	z	$\sim_8$	Rb_8	Ra_8	Rt_8	$03h_8$

SGE – Set if Greater Than or Equal

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is greater than or equal to a second operand in register (Rb) then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

There is no immediate form to this instruction. An immediate equivalent may be achieved using the SGT instruction and adjusting the constant by one.

SGEU – Set if Greater Than or Equal Unsigned

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is greater than or equal to a second operand in register (Rb) then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

There is no immediate form to this instruction. An immediate equivalent may be achieved using the SGTU instruction and adjusting the constant by one.

SGT – Set if Greater Than

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is greater than a second operand which is a constant supplied in the instruction, then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

There is no register form of this instruction. The register equivalent operation may be performed using the SLT instruction and swapping the registers.

SGTU – Set if Greater Than Unsigned

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is greater than a second operand which is a constant supplied in the instruction, then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

There is no register form of this instruction. The register equivalent operation may be performed using the SLTU instruction and swapping the registers.

SIGN – Sign

Synopsis

Take sign of value. Rt = Ra < 0 ? –1 : Ra = 0 ? 0 : 1

Description

The sign of a register is placed in the target register Rt.

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = Va[x] < 0 ? –1 : Va[x]=0 ? 0 : 1
```

SLLP –Shift Left Logical Pair

Description:

Left shift a pair of operand values by an operand value and place the result in the target register. The upper 64 bits of the result are placed in the target register. Zeros are shifted into the least significant bits. The operand pair must be in registers specified by the Ra and Rb field of the instruction. The third operand may be either a register specified by the Rc field of the instruction, or an immediate value.

This instruction may also be used to perform a left rotate of a single register by specifying the same register for Ra and Rb.

Formats Supported: SR3

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm_3	$_8$	0_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	Rt_8	$03h_8$

Operation Size: .o, .t, .w, .b

Execution Units: integer ALU

Exceptions: none

Example:

SLT – Set if Less Than

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is less than a second operand in either a register (Rb) or a constant supplied in the instruction, then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

The register form of the instruction may also be used to test for greater than by swapping the operands around.

SLE – Set if Less Than or Equal

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is less than or equal to a second operand in register (Rb) then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as signed values.

There is no immediate form to this instruction. An immediate equivalent may be achieved using the SLT instruction and adjusting the constant by one.

SLEU – Set if Less Than or Equal

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is less than or equal to a second operand in register (Rb) then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as unsigned values.

There is no immediate form to this instruction. An immediate equivalent may be achieved using the SLTU instruction and adjusting the constant by one.

SLTU – Set if Less Than Unsigned

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is less than a second operand in either a register (Rb) or a constant supplied in the instruction, then the target register is set to a one, otherwise the target register is set to a zero. The operands are treated as unsigned values.

The register form of the instruction may also be used to test for greater than by swapping the operands around.

SNE – Set if Not Equal

Description:

The set instruction places a 1 or 0 in the target register based on the relationship between the two source operands. If operand Ra is not equal to a second operand in register (Rb) or an immediate constant then the target register is set to a one, otherwise the target register is set to a zero.

For floating-point operations positive and negative zero are considered equal.

SRAP –Shift Right Arithmetic Pair

Description:

This is an alternate mnemonic for the signed field extract EXT instruction.

Right shift a pair of operand values by an operand value and place the result in the target register. The lower 64 bits of the result are placed in the target register. The sign bits is shifted into the most significant bits. The operand pair must be in registers specified by the Ra and Rb field of the instruction. The third operand may be either a register specified by the Rc field of the instruction, or an immediate value.

Instruction Format: SR4

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm$_3$	BFh$_8$	0$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	2Ch$_8$

Operation Size: .o, .t, .w, .b

Execution Units: integer ALU

Exceptions: none

Example:

SRLP –Shift Right Logical Pair

Description:

This is an alternate mnemonic for the unsigned field extract EXTU instruction.

Right shift a pair of operand values by an operand value and place the result in the target register. The lower 64 bits of the result are placed in the target register. Zeros are shifted into the most significant bits. The operand pair must be in registers specified by the Ra and Rb field of the instruction. The third operand may be either a register specified by the Rc field of the instruction, or an immediate value.

This instruction may also be used to perform a right rotate of a single register by specifying the same register for Ra and Rb.

Instruction Format: SR4

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm$_3$	BFh$_8$	0$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	24h$_8$

Operation Size: .o, .t, .w, .b

Execution Units: integer ALU

Exceptions: none

Example:

SUB - Subtract

Description:

Subtract two values. Both operands must be in a register or small immediates.

Instruction Format: SR2

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	1Ch$_8$	U$_2$	Sz$_4$	m$_3$	z	5$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

Scalar Operation

Rt = Ra - Rb

Vector Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Vt[x] = Va[x] - Vb[x]
    else if (z) Vt[x] = 0
    else Vt[x] = Vt[x]
```

SUBF – Subtract From

Description:

Subtract two values. The first operand must be in a register. The second operand must be an immediate value specified in the instruction. There is no register form for this instruction.

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
Constant_{32}	$\sim_2$	U_2	Sz_4	Ra_8	Rt_8	$05h_8$

Operation:

Rt = Imm - Ra

Exceptions: none

U21NDX – UTF21 Index

Description:

This instruction searches Ra, which is treated as an array of three UTF21 values, for a value specified by Rb or an immediate value and places the index of the value into the target register Rt. If the UTF21 value is not found -1 is placed in the target register. A common use would be to search for a null. The index result may vary from -1 to +2. The index of the first found value is returned (closest to zero).

Instruction Format: SR2

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
0_3	Rm_3	$\sim_8$	0_2	Sz_4	m_3	z	$\sim_8$	Rb_8	Ra_8	Rt_8	$23h_8$

63 61	60 58	57 50	4948	47 44	4341	40	39 24	23 16	15 8	7 0
1_3	Rm_3	$Imm_{23..16}$	0_2	Sz_4	m_3	z	$Imm_{15..0}$	Ra_8	Rt_8	$23h_8$

R2 Supported Formats: .t, .o

Clock Cycles: 1

Execution Units: Integer ALU

Operation:

Rt = Index of (Rb in Ra)

Exceptions: none

WYDNDX – Wyde Index

Description:

This instruction searches Ra, which is treated as an array of four wydes, for a wyde value specified by Rb or an immediate value and places the index of the wyde into the target register Rt. If the wyde is not found -1 is placed in the target register. A common use would be to search for a null wyde. The index result may vary from -1 to +3. The index of the first found wyde is returned (closest to zero).

Instruction Format: SR2

63 61	60 58	57 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
0_3	Rm_3	$\sim_8$	0_2	Sz_4	m_3	z	$\sim_8$	Rb_8	Ra_8	Rt_8	$1Bh_8$

63 61	60 58	57 50	4948	47 44	4341	40	39 24	23 16	15 8	7 0
1_3	Rm_3	$\sim_8$	0_2	Sz_4	m_3	z	Imm_{16}	Ra_8	Rt_8	$1Bh_8$

R2 Supported Formats: .t, .o

Clock Cycles: 1

Execution Units: Integer ALU

Operation:

Rt = Index of (Rb in Ra)

Exceptions: none

XOR – Bitwise Exclusive Or

Description:

Perform a bitwise exclusive or operation between operands. The first operand must be in a register. The second operand may be a register or immediate value. A third operand must be in a register. The immediate constant is zero extended before use.

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
Constant$_{32}$	~$_2$	0$_2$	Sz$_4$	Ra$_8$	Rt$_8$	0Ah$_8$

Register Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
~$_3$	Rm$_3$	0Ah$_8$	0$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	Rt$_8$	03h$_8$

Operation

Rt = Ra ^ Immediate

OR

Rt = Ra ^ Rb ^ Rc

Vector Operation

```
for x = 0 to VL-1
    if (Vm[x]) Vt[x] = Va[x] ^ Vb[x] ^ Vc[x]
    else if (z) Vt[x] = 0
    else Vt[x] = Vt[x]
```

Exceptions: none

ZXB –Zero Extend Byte

Description:

This is an alternate mnemonic for the bitfield extract (EXTU) operation.

Instruction Format: EXT

A bitfield in the source specified by Ra is extracted, the result is copied to the target register. Rc specifies the bit offset. Rd specifies the bit width.

Clock Cycles: 1

Execution Units: Integer ALU

Exceptions: none

Notes:

ZXW –Zero Extend Wyde

Description:

This is an alternate mnemonic for the bitfield extract (EXTU) operation.

Instruction Format: BFI

A bitfield in the source specified by Ra is extracted, the result is copied to the target register. Rc specifies the bit offset. Rd specifies the bit width.

Clock Cycles: 1

Execution Units: Integer ALU

Exceptions: none

Notes:

ZXT –Zero Extend Tetra

Description:

This is an alternate mnemonic for the bitfield extract (EXTU) operation.

Instruction Format: EXT

A bitfield in the source specified by Ra is extracted, the result is copied to the target register. Rc specifies the bit offset. Rd specifies the bit width.

Clock Cycles: 1

Execution Units: Integer ALU

Exceptions: none

Notes:

Memory Operations

CEA – Compute Effective Address

Description:

This instruction computes the effective address for a load/store operation.

Formats Supported:

Scalar Indexed Form (LD)

The effective address (EA) is calculated as the sum of Ra plus Rb multiplied by a scale and a constant and placed in target register Rt.

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Const$_{21..8}$	U$_2$	Sz$_4$	Sc$_3$	z	Const$_{7..0}$	Rb$_8$	Ra$_8$	Rt$_8$	68h$_8$

z: 1= zero extend, 0 = sign extend

Sc$_3$	Multiplier
0	1
1	2
2	4
3	8
4	16

Operation:

Rt = d + Ra + Rb * Sc

Vector forms

Stridden Form (LDS)

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Const$_{21..8}$	U$_2$	Sz$_4$	m$_3$	z	Const$_{7..0}$	Rb$_8$	Ra$_8$	Rt$_8$	69h$_8$

Vm[x]	z	Result
0	0	Vt[x] = Vt[x] (unchanged)
0	1	Vt[x] = 0 (set to zero)
1	0	Vt[x] = memory address
1	1	Vt[x] = memory address

U$_2$	Unit
0	integer
1	floating-point

2	decimal-float
3	posit

Sz_4	Operation Size
0	byte
1	wyde
2	tetra
3	octa
4	hexi

Operation:

```
for x = 0 to vector length
        if (Vm[x])
                Vt[x] = d + Ra + Rb * x
        else
                Vt[x] = z ? 0 : Vt[x]
```

Indexed Form

63 48	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$Const_{23..8}$	Sz_4	m_3	z	$Const_{7..0}$	Vb_8	Ra_8	Rt_8	$6Ah_8$

Operation:

```
n = 0
for x = 0 to vector length
        if (Vm[x])
                Vt[x] = d + Ra + Vb[x]
        else
                Vt[x] = z ? 0 : Vt[x]
```

Exceptions: none

LDx – Load

Description:

Load a value from memory into a register.

Formats Supported:

Scalar Indexed Form (LD)

The effective address (EA) is calculated as the sum of Ra plus Rb multiplied by a scale and a constant.

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$Const_{21..8}$	U_2	Sz_4	Sc_3	z	$Const_{7..0}$	Rb_8	Ra_8	Rt_8	$60h_8$

z: 1= zero extend, 0 = sign extend

Sc_3	Multiplier
0	1
1	2
2	4
3	8
4	16

Operation:

Rt = Memory[d + Ra + Rb * Sc]

Vector forms

Stridden Form (LDS)

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$Const_{21..8}$	U_2	Sz_4	m_3	z	$Const_{7..0}$	Rb_8	Ra_8	Rt_8	$61h_8$

Data is loaded from memory addresses separated by the stride amount specified by register field Rb, beginning with the sum of Ra and an immediate value. If the vector mask bit is clear and the ‘z’ bit is set in the instruction then the corresponding element of the vector register is loaded with zero. If the vector mask bit is clear and the ‘z’ bit is clear in the instruction then the corresponding element of the vector register is left unchanged (no value is loaded from memory).

Elements are loaded only up to the length specified in the vector length register.

Vm[x]	z	Result
0	0	Vt[x] = Vt[x] (unchanged)
0	1	Vt[x] = 0 (set to zero)
1	0	Vt[x] = memory, sign extended
1	1	Vt[x] = memory, zero extended

U_2	Unit
0	integer
1	floating-point
2	decimal-float
3	posit

Sz_4	Operation Size
0	byte
1	wyde
2	tetra
3	octa
4	hexi

Operation:

```
for x = 0 to vector length
    if (Vm[x])
        Vt[x] = Memory[d+Ra + Rb * x]
    else
        Vt[x] = z ? 0 : Vt[x]
```

Indexed Form

Data is loaded from memory addresses beginning with the sum of Ra and a vector element from Vb.

63 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Const$_{23..8}$	Sz$_4$	m$_3$	z	Const$_{7..0}$	Vb$_8$	Ra$_8$	Rt$_8$	62h$_8$

Operation:

```
n = 0
for x = 0 to vector length
    if (Vm[x])
        Vt[x] = Memory[d + Ra + Vb[x]]
    else
        Vt[x] = z ? 0 : Vt[x]
```

Exceptions: none

LDB – Load Byte (8 bits)

Description:

Data is loaded from the memory address which is the sum of an immediate value and the sum of Ra and Rb times a scale. The value loaded is sign extended from bit 7 to the machine width.

Formats Supported: LD

Operation:

$Rd = Memory_8[d + Ra + Rb*Sc]$

Exceptions: none

LDBZ – Load Byte, Zero Extend (8 bits)

Description:

Data is loaded from the memory address which is the sum of an immediate value and the sum of Ra and Rb times a scale. The value loaded is zero extended from bit 8 to the machine width.

Formats Supported: LD

Operation:

$Rd = Memory_8[d + Ra + Rb*Sc]$

Exceptions: none

LDO – Load Octa (64 bits)

Description:

Data is loaded into Rt from the memory address which is the sum of an immediate value and the sum of Ra and Rb scaled.

Formats Supported: RR,RI

Operation:

Rt = Memory_{64}[d + Ra + Rb*Sc]

Execution Units: Mem

Exceptions: none

LDT – Load Tetra (32 bits)

Description:

Data is loaded from the memory address which is the sum of Ra and an immediate value or the sum of Ra and Rb scaled. The value loaded is sign extended from bit 31 to the machine width.

Formats Supported: RR,RI

Operation:

Rt = Memory_{32}[d + Ra + Rb*Sc]

Execution Units: Mem

Exceptions: none

LDTZ – Load Tetra, Zero Extend (32 bits)

Description:

Data is loaded from the memory address which is the sum of Ra and an immediate value or the sum of Ra and Rb scaled. The value loaded is zero extended from bit 8 to the machine width.

Formats Supported: RR,RI

Operation:

Rt = Memory_{32}[d + Ra + Rb*Sc]

Execution Units: Mem

Exceptions: none

LDW – Load Wyde (16 bits)

Description:

Data is loaded from the memory address which is the sum of Ra and an immediate value or the sum of Ra and Rb scaled. The value loaded is sign extended from bit 15 to the machine width.

Formats Supported: LD

Operation:

$Rt = Memory_{16}[d + Ra + Rb*Sc]$

Execution Units: Mem

Exceptions: none

LDWZ – Load Wyde, Zero Extend (16 bits)

Description:

Data is loaded from the memory address which is the sum of Ra and an immediate value or the sum of Ra and Rb scaled. The value loaded is zero extended from bit 16 to the machine width.

Formats Supported: LD

Operation:

$Rt = Memory_{16}[d + Ra + Rb*Sc]$

Execution Units: Mem

Exceptions: none

STx – Store

Description:

Store values to memory. Either the contents of a scalar or vector register or a seven-bit immediate constant may be stored. Both scalar and vector store operations are possible.

Formats Supported:

Scalar Indexed Form (ST)

The effective address (EA) is calculated as the sum of Ra plus Rb multiplied by a scale and a constant.

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$Const_{21..8}$	U_2	Sz_4	Sc_3	z	$Const_{7..0}$	Rb_8	Ra_8	Rs_8	$70h_8$

z: 1= zero extend, 0 = sign extend

Sc_3	Multiplier
0	1
1	2
2	4
3	8
4	16

Operation:

Memory[d+Ra + Rb * Sc] = Rs

Vector forms

Stridden Form (STS)

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$Const_{21..8}$	U_2	Sz_4	m_3	z	$Const_{7..0}$	Rb_8	Ra_8	Rs_8	$71h_8$

Data is stored to memory addresses separated by the stride amount specified by register field Rb, beginning with the sum of Ra and an immediate value. If the vector mask bit is clear and the ‘z’ bit is set in the instruction then memory for the corresponding element of the vector register is stored with zero. If the vector mask bit is clear and the ‘z’ bit is clear in the instruction then memory corresponding to the element of the vector register is left unchanged (no value is stored to memory).

Elements are loaded only up to the length specified in the vector length register.

Vm[x]	z	Result
0	0	Memory = Memory (unchanged)
0	1	Memory = 0 (set to zero)

1	0	memory = Vt[x]
1	1	memory = Vt[x]

U_2	Unit
0	integer
1	floating-point
2	decimal-float
3	posit

Sz_4	Operation Size
0	byte
1	wyde
2	tetra
3	octa
4	hexi

Operation:

```
for x = 0 to vector length
    if (Vm[x])
        Memory[d+Ra + Rb * x] = Vt[x]
    else
        Memory[d+Ra + Rb * x]  = z ? 0 : Memory[d+Ra + Rb * x]
```

Indexed Form

Data is stored to memory addresses beginning with the sum of Ra and a vector element from Vb.

63 48	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
$Const_{23..8}$	Sz_4	m_3	z	$Const_{7..0}$	Vb_8	Ra_8	Rt_8	$62h_8$

Operation:

```
n = 0
for x = 0 to vector length
    if (Vm[x])
        Memory[d + Ra + Vb[x]] = Vt[x]
    else
        Memory = z ? 0 : Memory
```

Exceptions: none

STB – Store Byte (8 bits)

Description:

This instruction stores a byte (8 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled.

Instruction Format: ST

Operation:

$Memory_8$[d + Ra + Rb*Sc] = Rs

STBZ – Store Byte and Zero (8 bits)

Description:

This instruction stores a byte (8 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled. After the byte is stored to memory the register is zeroed out.

Instruction Format: ST

Operation:

$Memory_8$[d + Ra + Rb*Sc] = Rs
Rs = 0

STT – Store Tetra (32 bits)

Description:

This instruction stores a tetra-byte (32 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled.

Instruction Format: ST

Operation:

$\text{Memory}_{32}[d + Ra + Rb*Sc] = Rs$

STTZ – Store Tetra and Zero (32 bits)

Description:

This instruction stores a tetra-byte (32 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled. After the tetra is stored to memory the register is zeroed out.

Instruction Format: ST

Operation:

$\text{Memory}_{32}[d + Ra + Rb*Sc] = Rs$

$Rs = 0$

STW – Store Wyde (16 bits)

Description:

This instruction stores a byte (16 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled.

Instruction Format: ST

Operation:

$\text{Memory}_{16}[d + Ra + Rb*Sc] = Rs$

STWZ – Store Wyde and Zero (16 bits)

Description:

This instruction stores a byte (16 bit) value to memory. The memory address is calculated as the sum of an immediate constant and the sum of Ra and Rb scaled. After the wyde is stored to memory the register is zeroed out.

Instruction Format: ST

Operation:

$\text{Memory}_{16}[d + Ra + Rb*Sc] = Rs$

$Rs = 0$

Flow Control (Branch Unit) Operations

BEQ – Branch if Equal

Description:

This instruction branches to the target address if the contents of Ra and Rb are equal, otherwise program execution continues with the next instruction. The target address is formed as the sum of Rc and a displacement. If Rc is r63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\text{Displacement}_{16..3}$	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	0_8	$4Eh_8$

Operation:

```
If (Ra = Rb)
    PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

Notes:

For a floating-point comparison positive and negative zero are considered equal.

BGE – Branch if Greater Than or Equal

Description:

This instruction branches to the target address if the contents of Ra is greater than or equal to Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as signed values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	0$_8$	49h$_8$

Operation:

```
If (Ra >= Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BGEU – Branch if Greater Than or Equal Unsigned

Description:

This instruction branches to the target address if the contents of Ra is greater than or equal to Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as unsigned values. The target address is formed as the sum of Rc and a displacement. If Rc is r63 then the program counter value is used.

Formats Supported: BR

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	0$_8$	4Bh$_8$

Operation:

```
If (Ra >= Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BGT – Branch if Greater Than

Description:

This instruction is an alternate mnemonic for the BLT instruction where the register operands have been swapped.

This instruction branches to the target address if the contents of Ra is less than Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as signed values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	0$_8$	48h$_8$

Operation:

```
If (Ra < Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BGTU – Branch if Greater Than Unsigned

Description:

This instruction is an alternate mnemonic for the BLTU instruction where the register operands have been swapped.

This instruction branches to the target address if the contents of Ra is less than Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as unsigned values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\text{Displacement}_{16..3}$	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	0_8	$4Ah_8$

Operation:

```
If (Ra < Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BNE – Branch if Not Equal

Description:

This instruction branches to the target address if the contents of Ra and Rb are not equal, otherwise program execution continues with the next instruction. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\text{Displacement}_{16..3}$	U_2	Sz_4	m_3	z	Rc_8	Rb_8	Ra_8	0_8	$4Fh_8$

Operation:

```
If (Ra <> Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BLE – Branch if Less Than or Equal

Description:

This is an alternate mnemonic for the BGE instruction, where the register operands have been swapped.

This instruction branches to the target address if the contents of Ra is greater than or equal to Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as signed values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

Operation:

```
If (Ra >= Rb)
    PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BLEU – Branch if Less Than or Equal Unsigned

Description:

This is an alternate mnemonic for the BGEU instruction, where the register operands have been swapped.

This instruction branches to the target address if the contents of Ra is greater than or equal to Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as unsigned values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

Operation:

```
If (Ra >= Rb)
    PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BLT – Branch if Less Than

Description:

This instruction branches to the target address if the contents of Ra is less than Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as signed values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	0$_8$	48h$_8$

Operation:

```
If (Ra < Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BLTU – Branch if Less Than Unsigned

Description:

This instruction branches to the target address if the contents of Ra is less than Rb, otherwise program execution continues with the next instruction. The values in Ra and Rb are treated as unsigned values. The target address is formed as the sum of Rc and a displacement. If Rc is x63 then the program counter value is used.

Formats Supported: BR

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	0$_8$	4Ah$_8$

Operation:

```
If (Ra < Rb)
        PC = Rc + Displacement
```

Execution Units: Branch

Exceptions: none

BRA – Unconditional Branch

Description:

This instruction is an alternate mnemonic for the BEQ instruction.

Formats Supported: BR

63 50	4948	47 44	4341	40	39 32	31 24	23 16	15 8	7 0
Displacement$_{16..3}$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	0$_8$	0$_8$	0$_8$	4Eh$_8$

Flags Affected: none

Operation:

PC = PC + Displacement

Execution Units: Branch

Exceptions: none

Notes:

CHK – Check Register Against Bounds

Description:

A register is compared to two values. If the register is outside of the bounds then an exception will occur.

Immediate Instruction Format

63 32	31 30	29 28	27 24	23 16	15 8	7 0
Constant$_{32}$	Cn$_2$	U$_2$	Sz$_4$	Ra$_8$	Rs$_8$	22h$_8$

Cn$_2$	Interpretation
0	Rs <= Ra <= Constant
1	Rs < Ra <= Constant
2	Rs <= Ra < Constant
3	Rs < Ra < Constant

Instruction Format: S3

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Cn$_2$	Rm$_3$	Func$_8$	U$_2$	Sz$_4$	m$_3$	z	Rc$_8$	Rb$_8$	Ra$_8$	~$_8$	03h$_8$

Supported Formats: .b .w, .t, .o

Clock Cycles: 1

Execution Units: Integer ALU, Float, Decimal Float, Posit

Exceptions: bounds check

Notes:

The system exception handler will typically transfer processing back to a local exception handler.

JAL – Jump and Link

Description:

This instruction may be used to both call a subroutine and return from it. The address of the instruction after the JAL is stored in the specified return address register (Rt) then a jump to the address specified in the instruction plus an index register value is made. The address range is 44 bits or 16TB. The resulting calculated address is always hexi-byte (16 byte) aligned.

The return address register is assumed to be x1 if not otherwise specified. The JAL instruction does not require space in branch predictor tables.

If x63 is specified for Ra then the current program counter value is used.

Note the branch instructions may also be used to return from a subroutine.

Formats Supported: JAL

63 … 24	23 … 16	15 … 8	7 … 0
$\text{Constant}_{43..4}$	Ra_8	Rt_8	40h_8

Flags Affected: none

Operation:

```
Rt = PC + 8
If Ra=63
        PC = PC + displacement
Else
        PC = Ra + Displacement
```

Execution Units: Branch

Exceptions: none

Notes:

JMP – Jump

Description:

This instruction is an alternate mnemonic for the JAL instruction. It may be used to jump directly to a specific address. The address range is 44 bits or 16TB. The resulting calculated address is always hexi-byte (16 byte) aligned.

The return address register is assumed to be x0 (discarding the return address). The JMP instruction does not require space in branch predictor tables.

If r63 is specified for Ra then the current program counter value is used.

Formats Supported: JAL

63 24	23 16	15 8	7 0
Constant$_{43..4}$	Ra$_8$	00$_8$	40h$_8$

Flags Affected: none

Operation:

```
If Ra=63
        PC = PC + displacement
Else
        PC = Ra + Displacement
```

Execution Units: Branch

Exceptions: none

Notes:

RET – Return from Subroutine

Description:

This instruction is an alternate mnemonic for the JAL instruction. Register Ra is assumed to be x1 and register Rt is assumed to be x0. The constant is assumed to be zero.

Formats Supported: JAL

63 24	23 16	15 8	7 0
Constant$_{43..4}$	01_8	00_8	$40h_8$

Flags Affected: none

Operation:

Execution Units: Branch

Exceptions: an unimplemented instruction exception may occur if a vector register is specified.

Notes:

Return address prediction hardware may make use of the RET instruction.

SYNC -Synchronize

Description:

All instructions for a particular unit before the SYNC are completed and committed to the architectural state before instructions of the unit type after the SYNC are issued. This instruction is used to ensure that the machine state is valid before subsequent instructions are executed.

Floating Point Instructions

Vector Specific Instructions

Arithmetic / Logical

V2BITS

Synopsis

Convert Boolean vector to bits.

Description

The least significant bit of each vector element is copied to the corresponding bit in the target register. The target register is a scalar register.

Instruction Format

63 61	60 58	57 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$\sim_3$	Rm$_3$	0Ch$_8$	U$_2$	$\sim_4$	m$_3$	z	$\sim_8$	21h$_8$	Ra$_8$	Rt$_8$	03h$_8$

Operation

```
For x = 0 to VL-1
    if (Vm[x])
        Rt[x] = Va[x].LSB
    else if (z)
        Rt[x] = 0
```

Exceptions: none

VACC - Accumulate

Synopsis

Register accumulation. Rt = Va + Rb

Description

A vector register (Va) and scalar register (Rb) are added together and placed in the target scalar register Rt. Rb and Rt may be the same register which results in an accumulation of the values in the register.

Instruction Format: V2

Operation

```
for x = 0 to VL - 1
    if (Vm[x]) Rt = Va[x] + Rb
```

Example

```
ldi         x1,#0             ; clear results
vfmul.s     v1,v2,v3          ; multiply inputs (v2) times weights (v3)
vfacc.s     x1,v1,x1                   ; accumulate results
fadd.s      x1,x1,x2                   ; add bias (r2 = bias amount)
fsigmoid.s          x1,x1              ; compute sigmoid
```

VBITS2V

Synopsis

Convert bits to Boolean vector.

Description

Bits from a general register are copied to the corresponding vector target register.

Operation

```
For x = 0 to VL-1
    if (Vm[x]) Vt[x] = Ra[x]
```

Exceptions: none

VCIDX – Compress Index

Synopsis

Vector compression.

Description

A value in a register Ra is multiplied by the element number and copied to elements of vector register Vt guided by a vector mask register.

Operation

```
y = 0
for x = 0 to VL - 1
    if (Vm[x])
        Vt[y] = Ra * x
        y = y + 1
```

VCMPRSS – Compress Vector

Synopsis

Vector compression.

Description

Selected elements from vector register Va are copied to elements of vector register Vt guided by a vector mask register.

Operation

```
y = 0
for x = 0 to VL - 1
    if (Vm[x])
        Vt[y] = Va[x]
        y = y + 1
```

VEINS / VMOVSV – Vector Element Insert

Synopsis

Vector element insert.

Description

A general-purpose register Rb is transferred into one element of a vector register Vt. The element to insert is identified by Ra.

Operation

Vt[Ra] = Rb

Exceptions: none

VEX / VMOVS – Vector Element Extract

Synopsis

Vector element extract.

Description

A vector register element from Vb is transferred into a general-purpose register Rt. The element to extract is identified by Ra.

Operation

Rt = Vb[Ra]

Exceptions: none

VSCAN

Synopsis

.

Description

Elements of Vt are set to the cumulative sum of a value in register Ra. The summation is guided by a vector mask register.

Operation

```
sum = 0
for x = 0 to VL - 1
    Vt[x] = sum
    if (Vm[x])
        sum = sum + Ra
```

VSHLV – Shift Vector Left

Synopsis

Vector shift left.

Description

Elements of the vector are transferred upwards to the next element position. The first is loaded with the value zero. This is also called a slide operation.

Operation

```
For x = VL-1 to Amt
    Vt[x] = Va[x-amt]
For x = Amt-1 to 0
    Vt[x] = 0
```

Exceptions: none

VSHRV – Shift Vector Right

Synopsis

Vector shift right.

Description

Elements of the vector are transferred downwards to the next element position. The last is loaded with the value zero. This is also called a slide operation.

Operation

```
For x = 0 to VL-Amt
    Vt[x] = Va[x+amt]
For x = VL-Amt +1 to VL-1
    Vt[x] = 0
```

Exceptions: none

Memory Operations

CVLDx – Compressed Vector Load

Description:

Formats Supported:

Stridden Form

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
$Const_{21..8}$	U_2	Sz_4	m_3	z	$Const_{7..0}$	Rb_8	Ra_8	Rt_8	$65h_8$

Data is loaded from memory locations beginning at the sum of Ra and a constant and separated by the stride amount in the stride register Rb. Rb may also be a constant in the range -62 to 63. If Rb = -63 then the Sz_4 field is used to determine the stride.

Operation:

```
y = 0
for x = 0 to vector length
    if Rb is a constant
        if Rb = -63
            stride = Sz4
        else
            stride = Rb
    else
        stride = [Rb]
    n = stride * y
    if (Vm[x])
        Vt[y] = Memory[d+Ra + n]
        y = y + 1
for y = y to vector length

    Vt[y] = z ? 0 : Vt[y]

n = 0
```

If the vector mask bit is clear and the ‘z’ bit is set in the instruction then the corresponding element of the vector register is loaded with zero. If the vector mask bit is clear and the ‘z’ bit is clear in the instruction then the corresponding element of the vector register is left unchanged (no value is loaded from memory).

Elements are loaded only up to the length specified in the vector length register.

Vm[x]	z	Result

0	0	Vt[x] = Vt[x] (unchanged)
0	1	Vt[x] = 0 (set to zero)
1	0	Vt[x] = memory, sign extended
1	1	Vt[x] = memory, zero extended

Operation:

```
n = 0
y = 0
for x = 0 to vector length
    if (Vm[x])
        Vt[y] = Memory[d+Ra + n]
        n = n + sizeof precision
        y = y + 1
for y = y to vector length

    Vt[y] = z ? 0 : Vt[y]
```

Indexed Form

63 50	49 48	47 44	43 41	40	39 32	31 24	23 16	15 8	7 0
Const$_{21..8}$	U$_2$	Sz$_4$	m$_3$	z	Const$_{7..0}$	Rb$_8$	Ra$_8$	Rt$_8$	66h$_8$

Data is loaded from memory addresses beginning with the sum of Ra and a vector element from Vb.

Operation:

```
y = 0
for x = 0 to vector length
    if (Vm[x])
        Vt[y] = Memory[d+Ra + Vb[x]]
        y = y + 1
for y = y to vector length

    Vt[y] = z ? 0 : Vt[y]
```

Exceptions: none

CVSTx – Compressed Vector Store

Description:

Formats Supported:

Register Indirect with Displacement

Data is stored to consecutive memory addresses beginning with the sum of Ra and an immediate

Elements are stored only up to the length specified in the vector length register.

47 42	41 40	39 36	35 33	32	31 20	19 14	13 8	7	6 0
Const$_6$	U$_2$	Sz$_4$	m$_3$	z	Constant$_{12}$	Ra$_6$	Vs$_6$	1	74h$_7$

Vm[x]	z	Result
1	0	memory = Vs[x]
1	1	memory = Vs[x], Vs[x] = 0

Operation:

```
n = 0
for x = 0 to vector length
    if (Vm[x])
        Memory[d+Ra + n] = Vs[x]
        if (z) Vs[x] = 0
        n = n + sizeof precision
```

Stridden Form

The stridden form works much the same as the register indirect form except that data is stored to memory locations separated by the stride amount in the stride register.

47 42	41 40	39 36	35 33	32	31 26	25 20	19 14	13 8	7	6 0
Const$_6$	U$_2$	Sz$_4$	m$_3$	z	Const$_6$	Rb$_6$	Ra$_6$	Vs$_6$	1	75h$_7$

Operation:

```
y = 0
for x = 0 to vector length
    n = Rb * y
    if (Vm[x])
        Memory[d+Ra + n] = Vs[x]
        if (z) Vs[x] = 0
        y = y + 1
```

Indexed Form

Data is stored to memory addresses beginning with the sum of Ra and a vector element from Vb.

47 42	41 40	39 36	35 33	32	31 26	25 20	19 14	13 8	7	6 0
$Const_6$	U_2	Sz_4	m_3	z	$Const_6$	Vb_6	Ra_6	Vs_6	1	$76h_7$

Operation:

```
y = 0
for x = 0 to vector length
        if (Vm[x])
                Memory[d+Ra + Vb[y]] = Vs[x]
                if (z) Vs[x] = 0
                y = y + 1
```

Exceptions: none

Floating Point Instructions

Root Opcode Map

	000	001	010	011	100	101	110	111
ALU								
00000				{R3}	ADD	SUBF	MUL	
00001	AND	OR	EOR	SRAP	SRLP	{SET}	MULU	CSR
00010	DIV	DIVU	DIVSU		EXTU	MULF	MULSU	PERM
00011	REM	REMU	BYTNDX	WYDNDX	EXT	DEP	DEPI	FFO
00100	REMSU	DIVR	CHK	U21NDX	SAND	SOR	SEQ	SNE
00101	SLT	SGT	SLTU	SGTU				
00110	MADD	MSUB	NMADD	NMSUB				FDP
00111	ADDSI	ANDSI	ORSI	XORSI	APCSI			
Branch Unit								
01000	JAL							
01001	BLT	BGE	BLTU	BGEU	BEQI		BEQ	BNE
01010								
01011								
01100								
01101								
01110	PUSH	PUSHC	LINK	UNLINK				
01111	BRK	NOP	{OSR2}		DBG	ATNI	EXEC	
Memory Unit								
10000	LDB	LDBU	LDW	LDWU	LDT	LDTU	LDO	LDOR
10001	LDO	LEA	POP	PLDO			FLDO	
10010	LDO	LEA*	FLDO*	PLDO*	PLDT*	PLDW*		LDM
10011	LDB*	LDBU*	LDW*	LDWU*	LDT*	LDTU*	LDO*	LDOR*
10100	STB	STW	STT	STO	STOC	STPTR	STO lk	STOI
10101	STO lk			FSTO	PSTO			
10110	STO		FSTO*	PSTO*	PSTT*	PSTW*		STM
10111	STB*	STW*	STT*	STO*	STOC*	STPTR*	STO*lk	STOI*
11000								
11001								
11010								
11011								
Floating Point / Posit Arithmetic Unit								
11100								
11101								
11110								
11111								

{SR2} Dyadic Register Ops

	000	001	010	011	100	101	110	111
000	AND	OR	EOR	BMM	ADD	SUB	MUL	
001	NAND	NOR	ENOR	U21NDX	{R1}	MOV	MULU	MULH
010	DIV	DIVU	DIVSU	REM	REMU	REMSU	MULSU	PERM
011	PTRDIF		BYTNDX	WYDNDX	MULF	MULSUH	MULUH	RGF
100								
101								
110								
111								

{SR3} Triadic Register Ops

	000	001	010	011	100	101	110	111
000	AND	OR	EOR		ADD	SUB		
001	NAND	NOR	ENOR		{R2}		MULU	MULH
010							MULSU	PERM
011	PTRDIF				MULF	MULSUH	MULUH	
100								
101								
110								
111								