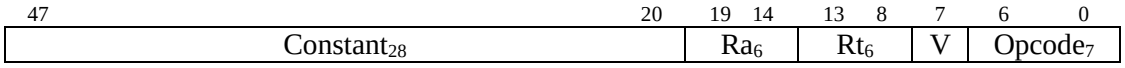
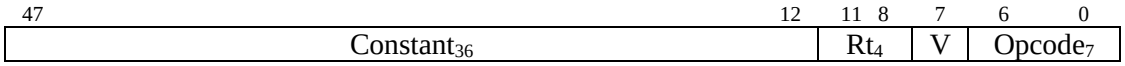
ANY-1 Instruction Set

Scalar Instructions

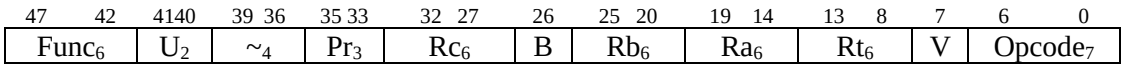
Immediate Format:



LUI / AUIPC



Register Format:



V: 1 = vector instruction, 0 = scalar
B: 1 = Rb is vector register, 0 = Rb is scalar

U_2	Execution Unit
0	Integer
1	Floating-point
2	Decimal floating-point
3	Posit

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:

Operation:

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

Exceptions: none

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

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.

Operation:

Rt = Ra & Imm

or

Rt = Ra & Rb & Rc

Exceptions: none

AUIPC – Add Upper Immediate to PC

Description:

This instruction forms the sum of the program counter and an immediate value shifted left 28 times. The result is then placed in the target register. The low order 28 bits of the target register are zeroed out.

The target register for this instruction must be one of x0 to x15.

Exceptions: none

CNTPOP – Count Population

Description:

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

Execution Units: ALU

Exceptions: none

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. There are two forms of this instruction, one uses registers to specify the offset and width, the other uses immediate constants supplied in the instruction to specify the offset and width. The width specified should be one less than the desired width.

Instruction Format: BFR

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

Instruction Format: BFI

A bitfield in the source specified by Ra is extracted, the result is copied to the target register. Bo specifies the bit offset. Bw specifies the bit width. Bo and Bw are constants supplied in the instruction.

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. There are two forms of this instruction, one uses registers to specify the offset and width, the other uses immediate constants supplied in the instruction to specify the offset and width. The width specified should be one less than the desired width.

Instruction Format: BFR

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

Instruction Format: BFI

A bitfield in the source specified by Ra is extracted, the result is copied to the target register. Bo specifies the bit offset. Bw specifies the bit width. Bo and Bw are constants supplied in the instruction.

Execution Units: Integer ALU

Exceptions: none

Notes:

LUI – Load Upper Immediate

Description:

This instruction loads an immediate value shifted left 28 times into a target register bits 28 to 63. The low order 28 bits of the target register are zeroed out.

The target register for this instruction must be one of x0 to x15.

Exceptions: none

MAX – Maximum Value

Description:

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

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.

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.

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.

Exceptions: none

NEG - Negate

Description:

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

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.

Operation:

Rt = !Ra

Exceptions: none

OR – Bitwise Or

Description:

Perform a bitwise or operation between operands.

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.

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.

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.

SUB - Subtract

Description:

Subtract two values. Both operands must be in a register.

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.

Operation:

Rt = Imm - 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: R2

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.

Exceptions: none

Memory Operations

LDB – Load Byte (8 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 times a scale. The value loaded is sign extended from bit 7 to the machine width.

Formats Supported: RR,RI

Operation:

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

Exceptions: none

LDBZ – Load Byte, Zero Extend (8 bits)

Description:

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

Formats Supported: RR,RI

Operation:

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

Exceptions: none

LDO – Load Octa (64 bits)

Description:

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

Formats Supported: RR,RI

Operation:

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

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]
or
Rt = $Memory_{32}$[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]
or
Rt = $Memory_{32}$[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: RR,RI

Operation:

Rt = $Memory_{16}$[d+Ra]
or
Rt = $Memory_{16}$[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: RR,RI

Operation:

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

Execution Units: Mem

Exceptions: none

SB – Store Byte (8 bits)

Description:

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

Instruction Format:

Operation:

$Memory_8[Ra + immediate] = Rs$

OR

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

SBZ – 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 Ra and an immediate constant OR the sum of Ra and Rb scaled. After the byte is stored to memory the register is zeroed out.

Instruction Format:

Operation:

$Memory_8[Ra + immediate] = Rs$
$Rs = 0$
OR

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

SW – Store Wyde (16 bits)

Description:

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

Instruction Format:

Operation:

$Memory_{16}[Ra + immediate] = Rs$

OR

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

SWZ – 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 Ra and an immediate constant OR the sum of Ra and Rb scaled. After the wyde is stored to memory the register is zeroed out.

Instruction Format:

Operation:

$Memory_{16}[Ra + immediate] = Rs$

$Rs = 0$

OR

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

$Rs = 0$

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 x63 then the program counter value is used.

Formats Supported: BR

Operation:

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

Execution Units: Branch

Exceptions: none

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

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 x63 then the program counter value is used.

Formats Supported: BR

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

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

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

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

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

Operation:

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

Execution Units: Branch

Exceptions: none

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 32 bits or 4GB. 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

Flags Affected: none

Operation:

Rt = PC + 8
PC = Ra + Displacement

Execution Units: Branch

Exceptions: none

Notes:

Floating Point Instructions

Vector 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.

Operation

For x = 0 to VL-1

Rt[x] = Va[x].LSB

Exceptions: none

VABS – Absolute value

Synopsis

Vector register absolute value. Vt = Va < 0 ? –Va : Va

Description

The absolute value of a vector register is placed in the target vector register Vt.

Operation

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

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
```

VADD - Add

Synopsis

Vector register add. Vt = Va + Vb

Description

Two vector registers (Va and Vb) are added together and placed in the target vector register Vt.

Operation

for x = 0 to VL - 1

if (Vm[x]) Vt[x] = Va[x] + Vb[x]

VADDS – Add Scalar

Synopsis

Vector register add. Vt = Va + Rb

Description

A vector and a scalar (Va and Rb) are added together and placed in the target vector register Vt.

Operation

for x = 0 to VL-1

if (Vm[x]) Vt[x] = Vb[x] + Rb

VAND – Bitwise And

Synopsis

Vector register bitwise and. Vt = Va & Vb

Description

Two vector registers (Va and Vb) are bitwise and'ed together and placed in the target vector register Vt.

Operation

for x = 0 to VL-1

if (Vm[x]) Vt[x] = Va[x] & Vb[x]

VANDS – Bitwise And with Scalar

Synopsis

Vector register bitwise and. Vt = Va & Rb

Description

A vector register (Va) is bitwise and'ed with a scalar register and placed in the target vector register Vt.

Operation

for x = 0 to VL-1

if (Vm[x]) Vt[x] = Va[x] & Rb

VASR – Arithmetic Shift Right

Synopsis

Vector signed shift right.

Description

Elements of the vector are shifted right. The most significant bits are loaded with the sign bit.

Operation

For x = 0 to VL-1

if (Vm[x]) Vt[x] = Va[x] >> amt

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
```

VCNTPOP – Population Count

Synopsis

Vector register population count. Vt = popcnt(Va)

Description

The number of bits set in a vector register is placed in the target vector register Vt.

Operation

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

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

VMUL - Multiply

Synopsis

Vector register multiply. Vt = Va * Vb

Description

Two vector registers (Va and Vb) are multiplied together and placed in the target vector register Vt.

Operation

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

VMULS – Multiply by Scalar

Synopsis

Vector register multiply by scalar. Vt = Va * Rb

Description

A vector register (Va) and a scalar register (Rb) are multiplied together and placed in the target vector register Vt.

Operation

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

VNEG – Negate

Synopsis

Vector register subtract. Vt = R0 - Va

Description

A vector is made negative by subtracting it from zero and placing it in the target vector register Vt. This instruction is an alternate mnemonic for the VSUBRS instruction.

Operation

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

VOR – Bitwise Or

Synopsis

Vector register bitwise or. Vt = Va | Vb

Description

Two vector registers (Va and Vb) are bitwise or'ed together and placed in the target vector register Vt.

Operation

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

VORS – Bitwise Or with Scalar

Synopsis

Vector register bitwise and. Vt = Va | Rb

Description

A vector register (Va) is bitwise ord'ed with a scalar register and placed in the target vector register Vt.

Operation

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

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
```

VSEQ – Set if Equal

Synopsis

Vector register set. Vm = Va == Vb

Description

Two vector registers (Va and Vb) are compared for equality and the comparison result is placed in the target vector mask register Vmt.

Operation

```
for x = 0 to VL-1
        Vm[x] = Va[x] == Vb[x]
```

Operation:

For each vector element

```
if signed Va equals signed Vb
        Vm = true
else
        Vm = false
```

VSEQS – Set if Equal Scalar

Synopsis

Vector register set. Vm = Va == Rb

Description

All elements of a vector are compared for equality to a scalar value. If equal a one is written to the output vector mask register, otherwise a zero is written to the output mask register.

Operation

```
for x = 0 to VL-1
        Vm[x] = Va[x] == Rb
```

Operation:

For each vector element

```
if signed Va equals signed Rb
        Vm = true
else
        Vm = false
```

VSGE – Set if Greater or Equal

Synopsis

Vector register set. Vm = Va >= Vb

Description

Two vector registers (Va and Vb) are compared for greater or equal and the comparison result is placed in the target vector mask register Vmt.

Operation

for x = 0 to VL-1

Vm[x] = Va[x] >= Vb[x]

Operation:

For each vector element

```
if signed Va greater than or equal signed Vb
        Vm = true
else
        Vm = false
```

VSGES – Set if Greater or Equal Scalar

Synopsis

Vector register set. Vm = Va >= Rb

Description

All elements of a vector are compared for greater or equal to a scalar value. If the condition is true a one is written to the output vector mask register, otherwise a zero is written to the output mask register.

Operation

for x = 0 to VL-1

Vm[x] = Va[x] >= Rb

Operation:

For each vector element

```
if signed Va greater than or equal signed Rb
        Vm = true
else
        Vm = false
```

VSHL – Shift Left

Synopsis

Vector shift left.

Description

Elements of the vector are shifted left. The least significant bits are loaded with the value zero.

Operation

For x = 0 to VL-1

if (Vm[x]) Vt[x] = Va[x] << amt

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.

Operation

For x = VL-1 to Amt

Vt[x] = Va[x-amt]

For x = Amt-1 to 0

Vt[x] = 0

Exceptions: none

VSHR – Shift Right

Synopsis

Vector shift right.

Description

Elements of the vector are shifted right. The most significant bits are loaded with the value zero.

Operation

For x = 0 to VL-1

if (Vm[x]) Vt[x] = Va[x] >> amt

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.

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

VSIGN – Sign

Synopsis

Vector register sign value. Vt = Va < 0 ? –1 : Va = 0 ? 0 : 1

Description

The sign of a vector register is placed in the target vector register Vt.

Operation

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

VSLT – Set if Less Than

Synopsis

Vector register set. Vm = Va < Vb

Description

Two vector registers (Va and Vb) are compared for less than and the comparison result is placed in the target vector mask register Vmt.

Operation

```
for x = 0 to VL-1
        Vm[x] = Va[x] < Vb[x]
```

Operation:

For each vector element

```
if signed Va less than signed Vb
        Vm = true
else
        Vm = false
```

VSLTS – Set if Less Than Scalar

Synopsis

Vector register set. Vm = Va < Rb

Description

A vector register (Va) and a scalar register (Rb) are compared for less than and the comparison result is placed in the target vector mask register Vmt.

Operation

```
for x = 0 to VL-1
        Vmt[x] = Va[x] < Rb
```

Operation:

For each vector element

```
if signed Va less than signed Rb
        Vmt = true
else
        Vmt = false
```

VSLTU – Set if Less Than Unsigned

Synopsis

Vector register set. Vm = Va < Vb

Description

Two vector registers (Va and Vb) are compared for less than and the comparison result is placed in the target vector mask register Vmt. The vector registers are treated as unsigned values.

Operation

```
for x = 0 to VL-1
        Vm[x] = Va[x] < Vb[x]
```

Operation:

For each vector element

```
if unsigned Va less than unsigned Vb
        Vm = true
else
        Vm = false
```

VSUB - Subtract

Synopsis

Vector register add. Vt = Va - Vb

Description

Two vector registers (Va and Vb) are subtracted and placed in the target vector register Vt.

Operation

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

VSUBFS – Subtract from Scalar

Synopsis

Vector register subtract. Vt = Rb - Va

Description

A vector and a scalar (Va and Rb) are subtracted and placed in the target vector register Vt.

Operation

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

VSUBS – Subtract Scalar

Synopsis

Vector register subtract. Vt = Va - Rb

Description

A vector and a scalar (Va and Rb) are subtracted and placed in the target vector register Vt.

Operation

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

VSYNC -Synchronize

Description:

All vector instructions before the VSYNC are completed and committed to the architectural state before vector instructions after the VSYNC are issued. This instruction is used to ensure that the machine state is valid before subsequent instructions are executed.

VXOR – Bitwise Exclusive Or

Synopsis

Vector register bitwise or. Vt = Va ^ Vb

Description

Two vector registers (Va and Vb) are exclusive or'ed together and placed in the target vector register Vt.

Operation

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

VXORS – Bitwise Exclusive Or with Scalar

Synopsis

Vector register bitwise and. Vt = Va ^ Rb

Description

A vector register (Va) is bitwise exclusive ord'ed with a scalar register and placed in the target vector register Vt.

Operation

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

Memory Operations

VLD – Load

Description:

Formats Supported: RR,RI

Register Indirect with Displacement

Data is loaded from consecutive memory addresses 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

Operation:

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

Stridden Form

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

Operation:

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

```
            Vt[x] = z ? 0 : Vt[x]
        n = n + vector stride
```

Indexed Form

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

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

VCLD – Compressed Load

Description:

Formats Supported: RR,RI

Register Indirect with Displacement

Data is loaded from consecutive memory addresses 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

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]
```

Stridden Form

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

Operation:

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

```
                y = y + 1
for y = y to vector length

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

n = 0
```

Indexed Form

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

Operation:

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

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

Exceptions: none