ANY-1 Instruction Set

© 2021 Robert Finch

# Scalar Instructions

Immediate Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 47 20 | 19 14 | 13 8 | 7 | 6 0 |
| Constant28 | Ra6 | Rt6 | V | Opcode7 |

LUI / AUIPC

|  |  |  |  |
| --- | --- | --- | --- |
| 47 12 | 11 8 | 7 | 6 0 |
| Constant36 | Rt4 | V | Opcode7 |

Register Format:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 47 42 | 4140 | 39 36 | 35 33 | 32 27 | 26 | 25 20 | 19 14 | 13 8 | 7 | 6 0 |
| Func6 | U2 | ~4 | Pr3 | Rc6 | B | Rb6 | Ra6 | Rt6 | V | Opcode7 |

V: 1 = vector instruction, 0 = scalar

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

|  |  |
| --- | --- |
| U2 | Execution Unit |
| 0 | Integer |
| 1 | Floating-point |
| 2 | Decimal floating-point |
| 3 | Posit |

## 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

## LUI – Load Upper Immediate

**Description**:

This instruction loads an immediate value shifted left 28 times into a target register bits 29 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

## 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

## 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.

## SUB - Subtract

**Description:**

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

## 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

# Vector Instructions

### 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]

### 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]

### VSUBRS – 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]