robfinch@<remove>Finitron.ca

## Document

This document outlines the proposed vector instruction set extension included with the Thor processing core.

## Some Considerations

### Interrupt latencies.

### Code density.

Code density impacts cache performance. When a single instruction can be used to represent a vector operation by specifying the vector length, code density is improved.

### Instruction Encoding

Thor uses variable length instruction encodings in order to achieve good code density. The instruction encoding allows for up to 64 vector registers.

# Vector Programming Model

There are eight, sixteen element vector registers.

|  |  |
| --- | --- |
| Reg no |  |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |

## Vector Length (VL register)

The vector length register controls how many elements of a vector are processed.

## Vector Predicates

There are eight sixteen element vector predicate registers. The predicate registers perform a function similar to the vector mask register in other architectures. All vector instructions are executed conditionally based on the value in a vector predicate register.

|  |  |
| --- | --- |
| Regno |  |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |

### Predicate Conditions

|  |  |  |  |
| --- | --- | --- | --- |
| Cond. |  | Test |  |
| 0 | PF | 0 | Always false – Instructions predicated with condition zero never execute regardless of the predicate register contents. This is used for extended immediate values as well. The false predicate byte for instructions is 90h. |
| 1 | PT | 1 | Always True – The instruction predicated with an always true condition always executes regardless of the predicate register contents. The always true predicate byte is 01h. Other true predicates are instruction short-forms. |
| 2 | PEQ | eq | Equal – instruction executes if the predicate register equal flag is set |
| 3 | PNE | !eq | Not Equal – instruction executes if the predicate register equal flag is clear |
| 4 | PLE | lt|eq | Less or Equal – predicate less or equal flag is set |
| 5 | PGT | !(lt|eq) | greater than |
| 6 | PGE | !lt | greater or equal |
| 7 | PLT | lt | less than |
| 8 | PLEU | ltu|eq | unsigned less or equal |
| 9 | PGTU | !(ltu|eq) | unsigned greater than |
| 10 | PGEU  POR | !ltu | unsigned greater or equal  Ordered for floating point |
| 11 | PLTU  PUN | ltu | unsigned less than  Unordered for floating point |
| 12 |  |  |  |
| 13 | PSIG | signal | execute if external signal is true |
| 14 |  |  |  |
| 15 |  |  |  |

## Detailed Vector Instruction Set

### VADD

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.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 03 | T3 | Vt6 | Vb6 | Va6 | 57h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL - 1

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

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | Float double |  |
| 4 | Float quad |  |

### VADDS

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.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 43 | T3 | Vt6 | Rb6 | Va6 | 57h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

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

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | Float double |  |
| 4 | Float quad |  |

### VAND

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.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 03 | T3 | Vt6 | Vb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

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

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VANDS

Synopsis

Vector register bitwise and. Vt = Va & Rb

***Description***

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 43 | T3 | Vt6 | Rb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

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

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VBITS2V

Synopsis

Convert bits to Boolean vector.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 66 | Vt6 | Ra6 | 568 | Pn4 | Pc4 |

**Description**

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

**Operation**

For x = 0 to [Ra]-1

Vt[x] = Ra[x]

**Exceptions:** none

### VCMP

Synopsis

Vector register compare. Vt = Va ? Vb

**Description**

Two vector registers (Va and Vb) are compared and the comparison result is placed in the target vector predicate register Vpt.

**Instruction Format**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3128 | 27 22 | 21 16 | 15 12 | 11 8 | 7 0 | |
| O4 | Vb6 | Va6 | 14 | Vpt4 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = Va[x] ? Vb[x]

**Operation:**

**For each vector element**

if signed Va < signed Vb

P.lt = true

else

P.lt = false

if unsigned Va < unsigned Vb

P.ltu = true

else

P.ltu = false

if Va = Vb

P.eq = true

else

P.eq = false

Operand Type

|  |  |  |
| --- | --- | --- |
| O4 |  |  |
| 8 | Integer |  |
| 9 | Float single |  |
| A | float double |  |
| C | Float Quad |  |

### VCMPS

Synopsis

Vector register compare. Vt = Va ? Rb

**Description**

A vector registers (Va) is compared to a scalar register (Rb) and the comparison result is placed in the target vector predicate register Vpt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 35 | 3432 | 31 28 | 27 22 | 21 16 | 15 8 | 7 0 | |
| ~5 | T3 | Vpt4 | Rb6 | Va6 | 5Ch8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = Va[x] ? Vb[x]

**Operation:**

**For each vector element**

if signed Va < signed Rb

P.lt = true

else

P.lt = false

if unsigned Va < unsigned Rb

P.ltu = true

else

P.ltu = false

if Va = Rb

P.eq = true

else

P.eq = false

### VEINS / VMOVSV

Synopsis

Vector element insert.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 16 | Vt6 | Rb6 | Ra6 | 568 | Pn4 | Pc4 |

***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***

Rt = Va[Ra]

***Exceptions:*** none

### VEOR

Synopsis

Vector register bitwise exclusive or. Vt = Va ^ Vb

***Description***

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 23 | T3 | Vt6 | Vb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = Va[x] & Vb[x]

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VEORS

Synopsis

Vector register bitwise exclusive or. Vt = Va ^ Rb

**Description**

A vector registers (Va) is exclusively or’ed with a scalar register (Rb) and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 63 | T3 | Vt6 | Rb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

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

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VDIV

Synopsis

Vector register divide. Vt = Va / Vb

**Description**

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 13 | T3 | Vt6 | Vb6 | Va6 | 5Eh8 | Vpn4 | Pc4 |

**Operation**

Vt[Ra] =Va[Ra] + Vb[Ra]

or

for x = 1 to [Ra]

Vt[x] = Va[x] + Vb[x]

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VDIVS

Synopsis

Vector register divide. Vt = Va / Rb

**Description**

A vector register (Va) is divided by a scalar and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 53 | T3 | Vt6 | Vb6 | Va6 | 5Eh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = Va[x] / Rb

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VEX / VMOVS

Synopsis

Vector element extract.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 06 | Rt6 | Va6 | Ra6 | 568 | Pn4 | Pc4 |

***Description***

One element of a vector register Va is transferred to a general purpose register Rt. The element to extract is identified by Ra.

***Operation***

Rt = Va[Ra]

***Exceptions:*** none

### VFLT2INT

Synopsis

Vector float to integer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 46 | Vt6 | Va6 | 568 | Vpn4 | Pc4 |

***Description***

Elements of the vector are converted from floating point to integer.

***Operation***

For x = 0 to [Ra]-1

Vt[x] = (int)Va[x]

***Exceptions:*** none

### VINT2FLT

Synopsis

Vector float to integer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 56 | Vt6 | Va6 | 568 | Vpn4 | Pc4 |

***Description***

Elements of the vector are converted from integer to floating point.

***Operation***

For x = 0 to VL-1

Vt[x] =(float) Va[x]

***Exceptions:*** none

### LV

Synopsis

Load vector

**Description:**

Load a vector register from memory using register indirect.

**Instruction Format:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~1 | Vt6 | Ra6 | BDh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

Vt[x] = memory64[Ra + 8 \* x]

**Exceptions:** DBE, DBG, LMT, TLB

### LVWS

Synopsis

Load vector using stride

**Description:**

Load a vector register from memory using register indirect with stride addressing.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~3 | Vt6 | Rb6 | Ra6 | BEh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = memory64[Ra+Rb \* x]

**Exceptions:** DBE, DBG, LMT, TLB

### LVX

Synopsis

Load vector

**Description:**

Load a vector register from memory using vector indexed addressing.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~3 | Vt6 | Vb6 | Ra6 | BFh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

Vt[x] = memory64[Ra+Vb[x]]

**Exceptions:** DBE, DBG, LMT, TLB

### VMAC

Synopsis

Vector register multiply accumulate. Vt = +-( +- Va \*+- Vb +- Vc)

**Description**

Vector registers Va and Vb are multiplied together then accumulated with vector register Vc. The sign of Va, Vb, and Vc may be inverted. The sign of the entire result may be inverted.

**Instruction Format**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 47 43 | 42 40 | 39 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| Sn5 | T3 | Vt6 | Vc6 | Vb6 | Va6 | 5Ah8 | Vpn4 | Pc4 |

**Operation**

Vt[Ra] =Va[Ra] \* Vb[Ra] + Vc

**Operand Type**

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VMUL

**Synopsis**

Vector register add. Vt = Va \* Vb

**Description**

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 03 | T3 | Vt6 | Vb6 | Va6 | 5Eh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

if (VM[x]) Vt[x] = Va[x] \* Vb[x]

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VMULS

**Synopsis**

Vector register add. Vt = Va \* Rb

**Description**

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 43 | T3 | Vt6 | Rb6 | Va6 | 5Eh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

if (VM[x]) Vt[x] = Va[x] \* Rb

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VOR

Synopsis

Vector register bitwise or. Vt = Va | Vb

***Description***

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 13 | T3 | Vt6 | Vb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 01 to VL-1

Vt[x] = Va[x] | Vb[x]

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VORS

Synopsis

Vector register bitwise or. Vt = Va | Rb

***Description***

A vector register (Va) is or’ed with a scalar register (Rb) and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 8 | 7 | 0 |
| 53 | T3 | Vt6 | Rb6 | Va6 | 52h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

Vt[x] = Va[x] | Rb

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
|  |  |  |
|  |  |  |
|  |  |  |

### VREC

**Synopsis**

Vector reciprocal. Vt = 1/Va

**Description**

The reciprocal of a vector (Va) is calculated placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 66 | Vt6 | Va6 | 568 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

if (VM[x]) Vt[x] = 1/ Va[x]

Operand Type

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VSHLV

Synopsis

Vector shift left.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 26 | Vt6 | Va6 | Ra6 | 568 | Pn4 | Pc4 |

***Description***

Elements of the vector are transferred upwards to the next element position. Element #0 is loaded with the value zero.

***Operation***

For x = 0 to [Ra]-1

Vt[x+1] = Va[x]

Vt[0] = 0

***Exceptions:*** none

### VSHRV

Synopsis

Vector shift right.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 36 | Vt6 | Va6 | Ra6 | 568 | Pn4 | Pc4 |

***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 [Ra]-1

Vt[x] = Va[x+1]

Vt[Ra-1] = 0

***Exceptions:*** none

### SV

Synopsis

Store vector

**Description:**

Store a vector register to memory using register indirect.

**Instruction Format:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 31 29 | 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~1 | Vs6 | Ra6 | CDh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to VL-1

memory64[Rb + x \* 8] = Vs[x]

**Exceptions:** DBE, DBG, LMT, TLB

### SVWS

Synopsis

Store vector using stride

**Description:**

Store a vector register to memory using register indirect with stride addressing.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~3 | Vt6 | Rb6 | Ra6 | CEh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

memory64[Ra+Rb \* x] = Vs[x]

**Exceptions:** DBE, DBG, LMT, TLB

### SVX

Synopsis

Store vector

**Description:**

Store a vector register to memory using vector indexed addressing.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| Seg3 | ~3 | Vs6 | Vb6 | Ra6 | CFh8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

memory64[Ra+Vb[x]] = Vs[x]

**Exceptions:** DBE, DBG, LMT, TLB

### VSUB

Synopsis

Vector register subtract. Vt = Va - Vb

**Description**

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

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 23 | T3 | Vt6 | Vb6 | Va6 | 57h8 | Vpn4 | Pc4 |

**Operation**

Vt[Ra] =Va[Ra] + Vb[Ra]

or

for x = 1 to [Ra]

Vt[x] = Va[x] + Vb[x]

**Operand Type**

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VSUBS

Synopsis

Vector register subtract. Vt = Va - Rb

**Description**

A scalar register is subtracted from a vector register and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 53 | T3 | Vt6 | Rb6 | Va6 | 57h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

Vt[x] = Va[x] + Rb

**Operand Type**

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### VSUBRS

Synopsis

Vector register subtract. Vt = Rb - Va

**Description**

A vector register is subtracted from a scalar register and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 37 | 36 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 63 | T3 | Vt6 | Rb6 | Va6 | 57h8 | Vpn4 | Pc4 |

**Operation**

for x = 0 to [Ra]-1

Vt[x] = Va[x] + Rb

**Operand Type**

|  |  |  |
| --- | --- | --- |
| T3 |  |  |
| 0 | Integer |  |
| 1 | Float single |  |
| 2 | float double |  |
| 4 | Float Quad |  |

### V2BITS

Synopsis

Convert Boolean vector to bits.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 28 | 27 22 | 21 16 | 15 0 | 7 | 0 |
| 66 | Rt6 | Va6 | Ra6 | 568 | Vpn4 | Pc4 |

***Description***

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

***Operation***

For x = 0 to VL-1

Rt[x] = Va[x].LSB

***Exceptions:*** none

Notes:

The register tag associated with a vector register contains both the element number and vector register number. The vector element being processed needs to be uniquely identified in the processor’s pipeline. With a large number of vector registers and a large number of elements the tag becomes quite large. The current core has 8 vector registers with sixteen elements each. This is to keep the tag within seven bits. The core ends up needing to process an eight bit register tag in order to handle all the registers.

Use of vector instructions serializes the core’s queuing of instructions. A separate instruction is queued for each element of the vector. This is done by stalling the instruction queued indicator until instructions for all the elements have been queued. In order to queue the vector instruction, queuing an element isn’t complete until the length is known. The length is in special register VL. If this register is valid the instruction will queue right away, otherwise it will be delayed until VL (argument a1) becomes valid. Once a1 becomes valid a new instruction should enqueue every clock cycle.