Draft RFC (1 of 2) Vector Looping Engine*

BASED ON THE LIBRESOC DEVELOPED SIMPLE-V SCALABLE VECTORS

* WORKING TITLE

Draft RFC (1 of 2)
Vector Looping Engine

The Basics

- As engineers we all understand the concept and use of compression/decompression to enhance data transfer.
- Simple-V employs compressed instructions that are optimally transferred into the decoder where they are decompressed maximising instruction issue (throughput).
- Compression reduces the number and volume of instructions crossing the chip physical boundary which is a major use of power and normally takes many magnitudes of additional clock cycles.

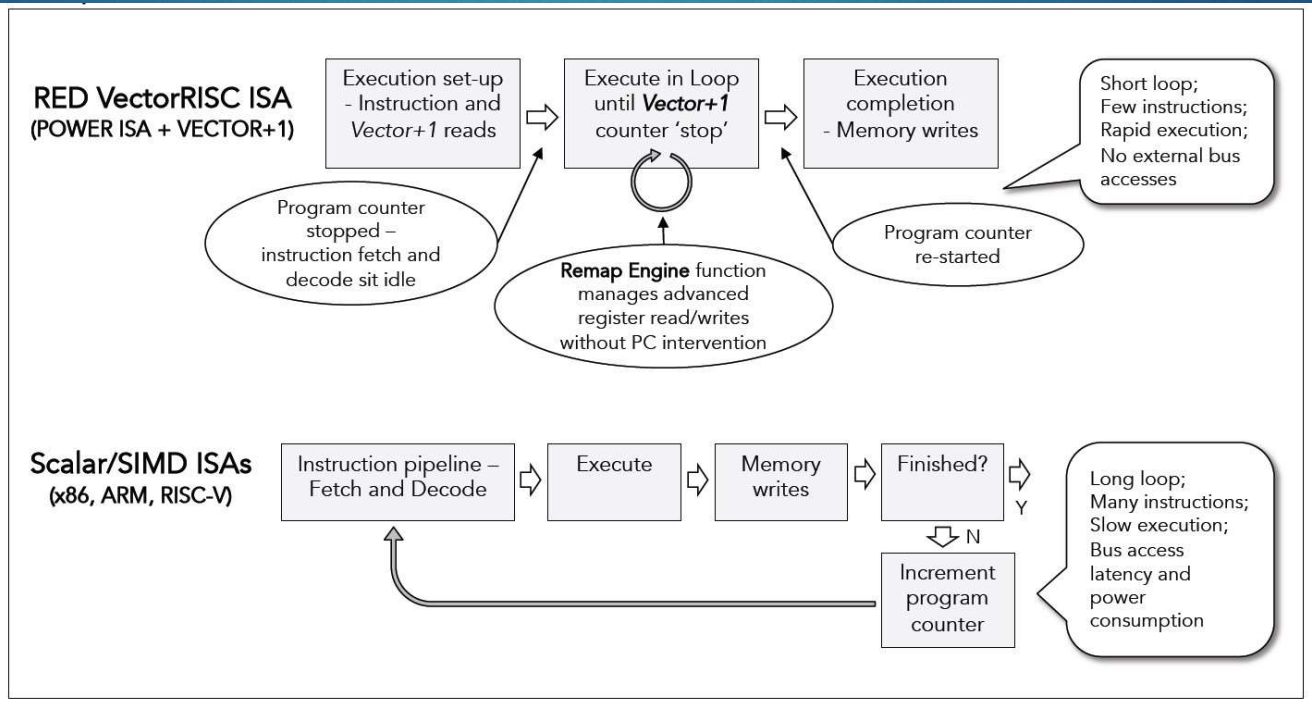
Draft RFC (1 of 2)
Vector Looping Engine

Simple-V is a looping mechanism similar in concept to Zilog Z80 LDIR and CPIR instructions, and 8086 REP instruction. In its most basic form it enumerates through sequential registers and therefore gives the effect of a Vector ISA, but instead, like LDIR CPIR and REP, relies exclusively on Scalar operations and Scalar register files.

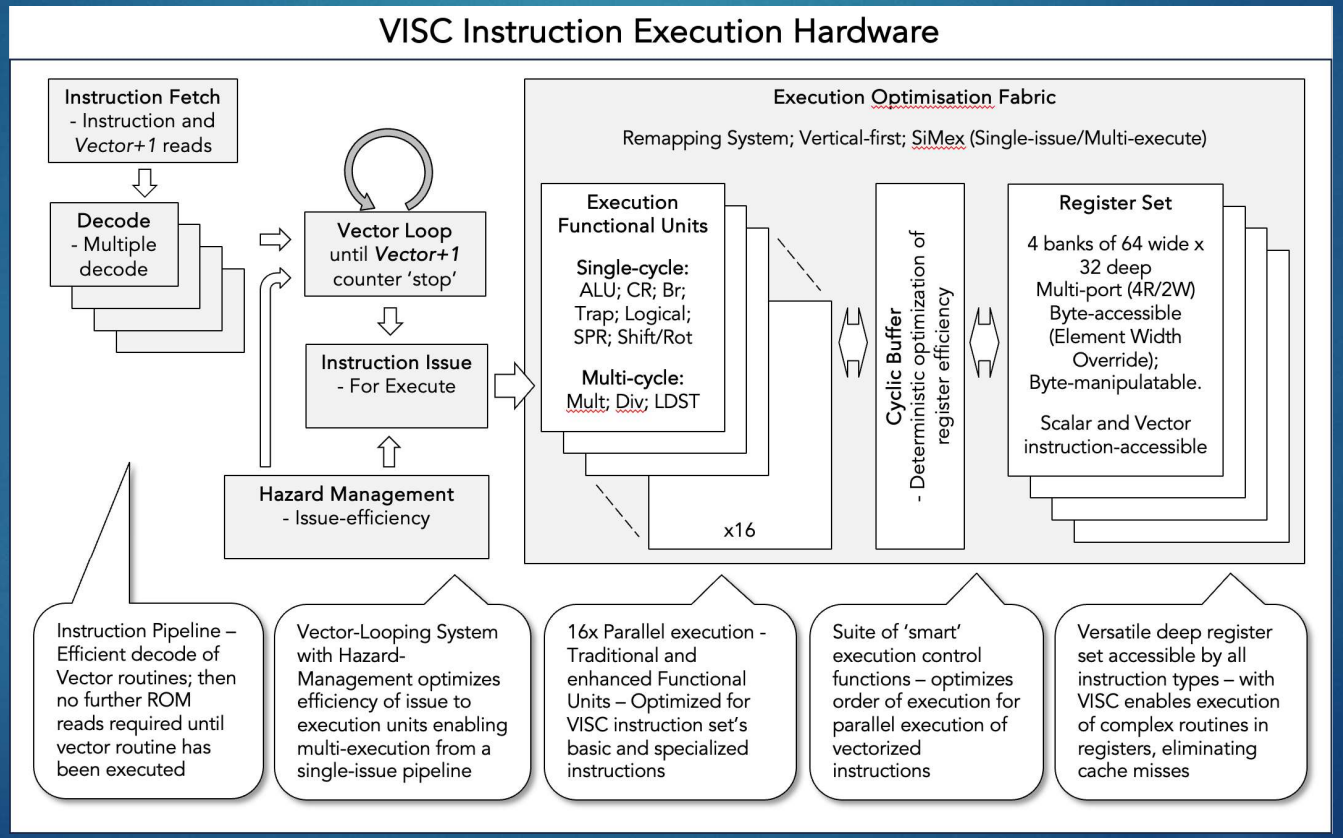
Simple-V provides a number of key compounding performance optimisations:

- It reduces code and ROM size saving cost
- It reduces memory access and power consumption
- The code for complex functions like cryptography can be entirely held in the L1 I-Cache and register files, greatly enhancing security
- It simplifies mathematical algorithm design, simultaneously reducing coding costs and improving security auditability.
- It does this at the same time as fulfilling its primary purpose, which is to maximise the saturation of the processor back end, whilst the decode engine can be idle, saving power.
- Finally it is readily compliable with simple patches to existing compilers. Simple-V is extensive, being originally based on Cray Vectors.

Draft RFC (1 of 2)
Vector Looping Engine



Draft RFC (1 of 2)
Vector Looping Engine



Draft RFC (1 of 2)
Vector Looping Engine

- **Predication**. Every looped instruction (termed an "element" in standard Vector ISAs) is individually predictable. However unlike standard Vector ISAs, Simple-V has **twin** predication: both source and destination may have separate predicate masks.
- **Element-width Overrides**. All standard Vector ISAs have operations at different bit-widths (8-bit, 16-bit...). Simple-V achieves this by *overriding* the usual bit-width of the prefixed instruction, on both source and destination registers.
- **Saturation**. Rather than add explicit "saturation" opcodes, Simple-V simply in a RISC-uniform fashion allows **all** Arithmetic and Logical Operations to be signed or unsigned saturated.
- **Data-Dependent Fail-First**. As the Looping is progressing, the result of the element is tested, and if the test fails the looping is terminated at that point. This is very useful for string copy (copying up to the NUL), linked-list walking (stopping at the end of the linked list) and other standard computer science algorithms including sorting algorithms.

Draft RFC (1 of 2)
Vector Looping Engine

- **Horizontal-First and Vertical-First**. Horizontal Mode is the normal expected mode of operation of any standard Vector ISA: the elements are all processed (together) before the Program Counter moves to the next instruction. **In Vertical-First Mode this is reversed**. Instructions execute **one** element, and it is required to execute an explicit "stepping" instruction to move on to the next element. Explained in https://www.youtube.com/watch?v=fn2KJvWyBKg
- **Extensive Branch-Conditional capabilities**. 3D GPUs and CUDA require predication testing to help skip over parallel operations that could be conditionally 100% masked out (parallel if/else). Simple-V combines Data-Dependent Fail-First and Branch-Conditional CTR Mode into SV Branch-Conditional, to replace what would otherwise require 5 to 10 instructions to achieve.
- **Sub-Vectors**. For 3D GPU, Audio/Video and HPC work it is very common to work with 2-vector (XY, LR Channel), 3-vector (XYZ, RGB), and 4-Vector (XYZW, ARGB). Simple-V therefore builds support for 2/3/4 sub-vectors into the Prefix as it is extremely commonly needed.

Draft RFC (1 of 2)
Vector Looping Engine

- **Pack and Unpack: Data Transpose**. For Matrix and other 2/3/4 Vector operations it is extremely useful to be able to transpose the data in-place. Pack and Unpack allows for transposing of both the source operands *and destination* operands.
- **REMAP Subsystem**. Reordering of the order of the elements is possible under hardware control. This allows for example Matrix Multiply to be executed in three instructions with arbitrary matrix dimensions, *without* needing to perform additional transpose operations as the transpose is taken care of by REMAP. Hardware modes include Matrix, DCT, FFT, Parallel-Reduction and Big-Integer. https://youtube.com/watch?v=NpmbUfgiuFE

Draft RFC (1 of 2)
Vector Looping Engine

- Simple-V recognises that not everyone will want to implement the full capability. Therefore Compliancy Levels have been introduced, including Embedded Levels through to HPC / 3D. https://libre-soc.org/openpower/sv/compliancy_levels/
- SV's upper Compliancy Levels extend the three register files (CR GPR and FPR) with up to QTY 128x64-bit GPR and FPR registers and QTY 128x4-bit CR Fields (available for use as Predicate Masks).
- SV's Compliancy Levels are unrelated to Power ISA Compliancy Subsets
- A critically-important aspect of Simple-V is that the Scalar instruction is NOT modified in any way shape or form. sv.addi is simply a sequence of addi instructions. However there are "qualifiers" such as Element-width overrides that reduce the bit-width of the operation down to 8/16/32 but this is the only seeming-modification of the Power ISA. RFC ls005.xlen covers the uniform changes required to the Scalar Power ISA.

Draft RFC (1 of 2)
Vector Looping Engine

- With modification of Scalar instructions being prohibited just by being Prefixed, the uniform RISC-like nature of Simple-V Prefixing extends into the compiler toolchain, drastically simplifying the addition of compiler support.
- The full specification is at https://libre-soc.org/openpower/sv
- Three ISA modifications are required
  - Modification to the Compliance levels SFFS and SFS to move PREFIX instructions to their own compliance group
  - Additional eight SPRs: SVSTATE, SVLR, SVSHAPE0, SVSHAPE1, SVSHAPE2, SVSHAPE3, 2 reserved
  - Adoption of seven 32Bit management instructions
    - setvl — Sets the vector length
    - svstep — Stepping instruction used in vertical first mode
    - svremap — Associates "schedules" with registers (GPR, FR, CR)
    - svindex — A general-purpose "remap" instruction (generalised xxperm)
    - svshape (includes svshape2) — Sets up commonly-used "schedules" such as DCT, FFT, Matrix, Parallel Reduction
    - svshape3 — Sets up commonly-used "schedules" such as DCT, FFT, Matrix, Parallel Reduction
    - svshape4 — Sets up commonly-used "schedules" such as DCT, FFT, Matrix, Parallel Reduction