OPF ISA WG External RFC LS001 08Sep2022

- RFC Author: Luke Kenneth Casson Leighton.
- RFC Contributors/Ideas: Brad Frey, Paul Mackerras, Konstantinos Magritis, Cesar Strauss, Jacob Lifshay, Toshaan Bharvani, Dimitry Selyutin, Andrey Miroshnikov
- https://git.openpower.foundation/isa/PowerISA/issues/64 [[ls001/discussion]]

This proposal is to extend the Power ISA with an Abstract RISC-Paradigm Vectorisation Concept that may be orthogonally applied to **all and any** suitable Scalar instructions, present and future, in the Scalar Power ISA. The Vectorisation System is called "Simple-V" and the Prefix Format is called "SVP64". **Simple-V is not a Traditional Vector ISA and therefore does not add Vector opcodes or regfiles**. An ISA Concept similar to Simple-V was originally invented in 1994 by Peter Hsu (Architect of the MIPS R8000) but was dropped as MIPS did not have an Out-of-Order Microarchitecture.

Simple-V is designed for Embedded Scenarios right the way through Audio/Visual DSPs to 3D GPUs and Supercomputing. As it does **not** add actual Vector Instructions, relying solely and exclusively on the **Scalar** ISA, it is **Scalar** instructions that need to be added to the **Scalar** Power ISA before Simple-V may orthogonally Vectorise them.

The goal of RED Semiconductor Ltd, an OpenPOWER Stakeholder, is to bring to market mass-volume general-purpose compute processors that are competitive in the 3D GPU Audio Visual DSP EDGE IoT desktop chromebook netbook smartphone laptop markets, performance-leveraged by Simple-V. Simple-V thus has to be accompanied by corresponding **Scalar** instructions that bring the **Scalar** Power ISA up-to-date. These include IEEE754 Transcendentals AV cryptographic Biginteger and bitmanipulation operations that ARM Intel AMD and many other ISAs have been adding over the past 12 years and Power ISA has not. Three additional FP-related sets are needed (missing from SFFS) - int_fp_mv fclass and fcvt and one set named crweird increase the capability of CR Fields.

Thus it becomes necesary to consider the Architectural Resource Allocation of not just Simple-V but the 80-100 Scalar instructions all at the same time.

It is also critical to note that Simple-V **does not modify the Scalar Power ISA**, that **only** Scalar words may be Vectorised, and that Vectorised instructions are **not** permitted to be different from their Scalar words. The sole exception to that is Vectorised Branch Conditional, in order to provide the usual Advanced Branching capability present in every Commercial 3D GPU ISA, but it is the *Vectorised* Branch-Conditional that is augmented, not Scalar Branch.

Extension Levels

Simple-V has been subdivided into levels akin to the Power ISA Compliancy Levels. For now let us call them "SV Extension Levels" to differentiate the two. The reason for the SV Extension Levels is the same as for the Power ISA Compliancy Levels (SFFS, SFS): to not overburden implementors with features that they do not need. *There is no dependence between the two types of Levels.* The resources below therefore are not all required for all SV Extension Levels but they are all required to be reserved.

Binary Interoperability

Power ISA is long-term stable. A catastrophic mistake has been made in ARM SVE/2 and RISC-V RVV: "Silicon-Partner" Scalability, marketed as a feature, allows the same instructions to mean different things on different implementations (a different Vector bitwidth). Binary interoperability is thus not only impossible to achieve but Illegal Instruction trap-and-emulate is also out of the question. Worse than that a **future** vendor may suddenly render **all existing** hardware non-interoperable, which is the worst possible thing for any specification to permit yet this is what SVE and RVV do *by design.*

Simple-V guarantees binary interoperability by defining fixed register file bitwidths and size for all instructions. This does mean that `RESERVED` space is crucial to have, in order to safely provide future expanded register file bitwidths and sizes.[1]

Hardware Implementations

The fundamental principle of Simple-V is that it sits between Issue and Decode, pausing the Program-Counter to service a "Sub-PC" hardware for-loop. This is very similar to "Zero-Overhead Loops" in High-end DSPs (TI MSP Series).

Considerable effort has been expended to ensure that Simple-V is practical to implement on an extremely wide range of Industry-wide common **Scalar** micro-architectures. Finite State Machine (for ultra-low-resource and Mission-Critical), In-order single-issue, all the way through to Great-Big Out-of-Order Superscalar Multi-Issue. The SV Extension Levels specifically recognise these differing scenarios.

SIMD back-end ALUs particularly those with element-level predicate masks may be exploited to good effect with very little additional complexity to achieve high throughput, even on a single-issue in-order microarchitecture. As usually becomes quickly apparent with in-order, its limitations extend also to when Simple-V is deployed, which is why Multi-Issue Out-of-Order is the recommended (but not mandatory) Scalar Micro-architecture.

The only major concern is in the upper SV Extension Levels: the Hazard Management for increased number of Scalar Registers to 128 (in current versions) but given that IBM POWER9/10 has VSX register numbering 64, and modern GPUs have 128, 256 amd even 512 registers this was deemed acceptable. Strategies do exist in hardware for Hazard Management of such large numbers of registers, even for Multi-Issue microarchitectures.

[1] an MSR bit or bits, conceptually equivalent to `MSR.SF` and added for the same reasons, would suffice perfectly.

Simple-V Architectural Resources

- No new Interrupt types are required. (**No modifications to existing Power ISA opcodes are required either**).
- GPR FPR and CR Field Register extend to 128. A future version may extend to 256 or beyond[2] or also extend VSX[3]
- 24-bits are needed within the main SVP64 Prefix (equivalent to a 2-bit XO)
- Another 24-bit (a second 2-bit XO) is needed for a planned future encoding, currently named "SVP64-Single"[4]
- A third 24-bits (third 2-bit XO) is strongly recommended to be `RESERVED` such that future unforeseen capability is needed (although this may be alternatively achieved with a mandatory PCR or MSR bit)
- To hold all Vector Context, five SPRs are needed for userspace. If Supervisor and Hypervisor mode are to also support Simple-V they will correspondingly need five SPRs each. (Some 32/32-to-64 aliases are advantageous but not critical).
- Five 6-bit XO (A-Form) "Management" instructions are needed. These are Scalar 32-bit instructions and *may* be 64-bit-extended in future (safely within the SVP64 space: no need for an EXT001 encoding).

Summary of Simple-V Opcode space

- 75% of one Major Opcode (equivalent to the rest of EXT017)
- Five 6-bit XO 32-bit operations.

No further opcode space *for Simple-V* is envisaged to be required for at least the next decade (including if added on VSX)

Simple-V SPRs

- **SVSTATE** - Vectorisation State sufficient for Precise-Interrupt Context-switching and no adverse latency.
- **SVSRR0** - identical in purpose to SRR0/1: storing SVSTATE on context-switch along-side MSR and PC.
- **SVSHAPE0-3** - these are 32-bit and may be grouped in pairs, they REMAP (shape) the Vectors
- **SVLR** - again similar to LR for exactly the same purpose, SVSTATE is swapped with SVLR by SV-Branch-Conditional for exactly the same reason that NIA is swapped with LR

Vector Management Instructions

These fit into QTY 5of 6-bit XO 32-bit encoding (svshape and svshape2 share the same space):

- **setvl** - Cray-style Scalar Vector Length instruction
- **svstep** - used for Vertical-First Mode and for enquiring about internal state
- **svremap** - "tags" registers for activating REMAP
- **svshape** - convenience instruction for quickly setting up Matrix, DCT, FFT and Parallel Reduction REMAP
- **svshape2** - additional convenience instruction to set up "Offset" REMAP (fits within svshape's XO encoding)
- **svindex** - convenience instruction for setting up "Indexed" REMAP.

SVP64 and SVP64-Single 24-bit Prefixes

The SVP64 24-bit Prefix provides several options, too numerous to describe in this document but all fitting within the 24-bit space (and no other). The primary options are:

- element-width overrides, which dynamically redefine each SFFS or SFS Scalar prefixed instruction to be 8-bit, 16-bit, 32-bit or 64-bit operands **without requiring new 8/16/32 instructions**[5]
- predication. this is an absolutely essential feature for a 3D GPU VPU ISA. CR Fields are available as Predicate Masks hence the reason for their extension to 128.
- Saturation. **all** LD/ST and Arithmetic and Logical operations may be saturated (without adding explicit scalar saturated opcodes)
- Reduction and Prefix-Sum (Fibonnacci Series) Modes as well as vec2/3/4 "Packing" and "Unpacking".

The `SVP64-Single` 24-bit encoding focusses primarily on ensuring that all 128 Scalar registers are fully accessible, provides element-width overrides, one-bit predication and brings Saturation to all existing Scalar operations. BF16 and FP16 are thus provided in the Scalar Power ISA without one single explicit FP16 or BF16 32-bit opcode being added. The downside: such Scalar operations are all 64-bit encodings.

[2] Prefix opcode space (or MSR bits) **must** be reserved in advance to do so, in order to avoid the catastrophic binary-incompatibility mistake made by RISC-V RVV and ARM SVE/2

[3] A future version or other Stakeholder *may* wish to drop Simple-V onto VSX: this would be a separate RFC

[4] SVP64-Single is remarkably similar to the "bit 1" of EXT001 being set to indicate that the 64-bits is to be allocated in full to a new encoding, but in fact SVP64-single still embeds v3.0 Scalar operations.

[5] elwidth overrides does however mean that all SFS / SFFS pseudocode will need rewriting to be in terms of XLEN. This has the indirect side-effect of automatically making a 32-bit Scalar Power ISA Specification possible, as well as a future 128-bit one (Cross-reference: RISC-V RV32 and RV128

Simple-V REMAP subsystem

REMAP is extremely advanced but brings features already present in other DSPs and Supercomputing ISAs.

- **DCT/FFT** REMAP brings more capability than TI's MSP-Series DSPs and Qualcom Hexagon DSPs, and is not restricted to Integer or FP. (Galois Field is possible, implementing NTT). Operates *in-place* significantly reducing register usage.
- **Matrix** REMAP brings more capability than any other Matrix Extension (AMD GPUs, Intel, ARM), not being restricted to Power-2 sizes. Also not limited to the type of operation, it may perform Warshall Transitive Closure, Integer Matrix, Bitmanipulation Matrix, Galois Field (carryless mul) Matrix, and with care potentially Graph Maximum Flow as well. Also suited to Convolutions, Matrix Transpose and rotate, *all* of which is in-place.
- **General-purpose Indexed** REMAP, this option is provided to implement an equivalent of VSX `vperm`
- **Parallel Reduction** REMAP, performs an automatic map-reduce using *any suitable scalar operation.*

Scalar Operations

The primary reason for mentioning the additional Scalar operations is because they are so numerous, with Power ISA not having advanced in the *general purpose* compute area in the past 12 years, that some considerable care is needed.

Summary: **Including Simple-V, to fit everything at least 75% of 3 separate Major Opcodes would be required**

Candidates (for all but the X-Form instructions) include:

- EXT006 (80% free)
- EXT017 (75% free but not recommended)
- EXT001 (50% free)
- EXT009 (100% free)
- EXT005 (100% free)
- brownfield space in EXT019 (25% but NOT recommended)

SVP64, SVP64-Single and SVP64-Reserved would require on their own each 25% of one Major Opcode for a total of 75% of one Major Opcode. The remaining **Scalar** opcodes, due to there being two separate sets of operations with 16-bit immediates, will require the other space totalling two 75% Majors.

Note critically that:

- Unlike EXT001, SVP64's 24-bits may **not** hold also any Scalar operations. There is no free available space: a 25th bit would be required. The entire 24-bits is **required** for the abstracted Hardware-Looping Concept **even when these 24-bits are zero**
- Any Scalar 64-bit instruction (regardless of how it is encoded) is unsafe to then Vectorise because this creates the situation of Prefixed-Prefixed, resulting in deep complexity in Hardware Decode at a critical juncture, as well as introducing 96-bit instructions.
- **All** of these Scalar instructions are candidates for Vectorisation. Thus none of them may be 64-bit-Scalar-only.

Minor Opcodes to fit candidates above

In order of size, for bitmanip and A/V DSP purposes:

- QTY 3of 2-bit XO: ternlogi, crternlogi, grevlogi
- QTY 7of 3-bit XO: xpermi, binlut, grevlog, swizzle-mv/fmv, bitmask, bmrevi
- QTY 8of 5/6-bit (A-Form): xpermi, bincrflut, bmask, fmvis, fishmv, bmrev, Galois Field
- QTY 30of 10-bit (X-Form): cldiv/mul, av-min/max/diff, absdac, xperm etc. (easily fit EXT019, EXT031).

Note: Some of the Galois Field operations will require QTY 1of Polynomial SPR (per userspace supervisor hypervisor).

EXT004

For biginteger math, two instructions in the same space as "madd" are to be proposed. They are both 3-in 2-out operations taking or producing a 64-bit "pair" (like RTp), and perform 128/64 mul and div/mod operations respectively. These are **not** the same as VSX operations which are 128/128, and they are **not** the same as existing Scalar mul/div/mod, all of which are 64/64 (or 64/32).

EXT059 and EXT063

Additionally for High-Performance Compute and Competitive 3D GPU, IEEE754 FP Transcendentals are required, as are some DCT/FFT "Twin-Butterfly" operations. For each of EXT059 and EXT063:

- QTY 33of X-Form "1-argument" (fsin, fsins, fcos, fcoss)
- QTY 15of X-Form "2-argument" (pow, atan2, fhypot)
- QTY 5of A-Form "3-in 2-out" FP Butterfly operations for DCT/FFT
- QTY 8of X-Form "2-in 2-out" FP Butterfly operations (again for DCT/FFT)
- An additional 16 instructions for IEEE754-2019 (fminss/fmaxss, fminmag/fmaxmag) under evaluation as of 08Sep2022

Potential Opcode allocation solution

There are unfortunately some inviolate requirements that directly place pressure on the EXT000-EXT063 (32-bit) opcode space to such a degree that it risks jeapordising the Power ISA. These requirements are:

- all of the scalar operations must be Vectoriseable
- all of the scalar operations intended for Vectorisation must be in a 32-bit encoding (not prefixed-prefixed to 96-bit)
- bringing Scalar Power ISA up-to-date from the past 12 years needs 75% of two Major opcodes all on its own

There exists a potential scheme which meets (exceeds) the above criteria, providing plenty of room for both Scalar (and Vectorised) operations, *and* provides SVP64-Single with room to grow. It is based loosely around Public v3.1 EXT001 Encoding.[6]

0-5	6	7	8-31	Description
PO	0	0	0000	new-suffix `RESERVED1`
PO	0	0	!zero	new-suffix, scalar (SVP64Single)
PO	1	0	0000	new scalar-only word, or `RESERVED2`
PO	1	0	!zero	old-suffix, scalar (SVP64Single)
PO	0	1	nnnn	new-suffix, vector (SVP64)
PO	1	1	nnnn	old-suffix, vector (SVP64)

- **PO** - Primary Opcode. Likely candidates: EXT005, EXT009
- **bit 6** - specifies whether the suffix is old (EXT000-EXT063) or new (EXTn00-EXTn63, n greater than 1)
- **bit 7** - defines whether the Suffix is Scalar-Prefixed or Vector-Prefixed (caveat: see bits 8-31)
- **old-suffix** - the EXT000 to EXT063 32-bit Major opcodes of Power ISA 3.0
- **new scalar-only** - a **new** Major Opcode area **exclusively** for Scalar-only instructions that shall **never** be Prefixed by SVP64 (RESERVED2 EXT300-EXT363)
- **new-suffix** - a **new** Major Opcode area (RESERVED1 EXT200-EXT263) that **may** be Prefixed by SVP64 and SVP64Single
- **0000** - all 24 bits bits 8-31 are zero (0x000000)
- **!zero** - bits 8-31 may be any value *other* than zero (0x000001-0xffffff)
- **nnnn** - bits 8-31 may be any value in the range 0x000000 to 0xffffff
- **SVP64Single** - a (TBD) *Scalar* Encoding that is near-identical to SVP64 except that it is equivalent to hard-coded VL=1 at all times. Predication is permitted, Element-width-overrides is permitted, Saturation is permitted.
- **SVP64** - a (well-defined, 2 years) DRAFT Proposal for a Vectorisation Augmentation of suffixes.

For the needs identified by Libre-SOC (75% of 2 POs), `RESERVED1` space *needs* allocation to new POs, `RESERVED2` does not.[7]

	Scalar (bit7=0,8-31=0000)	Scalar (bit7=0,8-31=!zero)	Vector (bit7=1)
new bit6=0	`RESERVED1`:{EXT200-263}	SVP64-Single:{EXT200-263}	SVP64:{EXT200-263}
old bit6=1	`RESERVED2`:{EXT300-363}	SVP64-Single:{EXT000-063}	SVP64:{EXT000-063}

- **`RESERVED2:{EXT300-363}`** (not strictly necessary to be added) is not and **cannot** ever be Vectorised or Augmented by Simple-V or any future Simple-V Scheme. it is a pure **Scalar-only** word-length PO Group. It may remain `RESERVED`.
- **`RESERVED1:{EXT200-263}`** is also a new set of 64 word-length Major Opcodes. These opcodes do not *need* to be Simple-V-Augmented *but the option to do so exists* should an Implementor choose to do so. This is unlike `EXT300-363` which may **never** be Simple-V-Augmented under any circumstances.
- **`SVP64-Single:{EXT200-263}`** - Major opcodes 200-263 with Single-Augmentation, providing a one-bit predicate mask, element-width overrides on source and destination, and the option to extend the Scalar Register numbering (r0-32 extends to r0-127). **Placing of alternative instruction encodings other than those exactly defined in EXT200-263 is prohibited**.
- **`SVP64-Single:{EXT000-063}`** - Major opcodes 000-063 with Single-Augmentation, just like SVP64-Single on EXT200-263, these are in effect Single-Augmented-Prefixed variants of the v3.0 32-bit Power ISA. Alternative instruction encodings other than the exact same 32-bit word from EXT000-EXT063 are likewise prohibited.
- **`SVP64:{EXT000-063}`** and **`SVP64:{EXT200-263}`** - Full Vectorisation of EXT000-063 and EXT200-263 respectively, these Prefixed instructions are likewise prohibited from being a different encoding from their 32-bit scalar versions.

Limitations of this scheme is that new 32-bit Scalar operations have to have a 32-bit "prefix pattern" in front of them. If commonly-used this could increase binary size. Thus the Encodings EXT300-363 and EXT200-263 should only be allocated for less-popular operations. However the scheme does have the strong advantage of *tripling* the available number of Major Opcodes in the Power ISA, caveat being that care on allocation is needed because EXT200-EXT263 may be SVP64-Augmented whilst EXT300-EXT363 may **not**. The issues of allocation for bitmanip etc. from Libre-SOC is therefore overwhelmingly made moot. The only downside is that there is no `SVP64-Reserved` which will have to be achieved with SPRs (PCR or MSR).

[6] Recall that EXT100 to EXT163 is for Public v3.1 64-bit-augmented Operations prefixed by EXT001, for which, from Section 1.6.3, bit 6 is set to 1. This concept is where the above scheme originated. Section 1.6.3 uses the term "defined word" to refer to pre-existing EXT000-EXT063 32-bit instructions so prefixed to create the new numbering EXT100-EXT163, respectively

[7] reminder that this proposal only needs 75% of two POs for Scalar instructions. The rest of EXT200-263 is for general use.

EXT000-EXT063

These are Scalar word-encodings. Often termed "v3.0 Scalar" in this document Power ISA v3.1 Section 1.6.3 Book I calls it a "defined word".

0-5	6-31
PO	EXT000-063 Scalar (v3.0 or v3.1) operation

RESERVED2 / EXT300-363 bit6=old bit7=scalar

This is entirely at the discretion of the ISA WG. Libre-SOC is *not* proposing the addition of EXT300-363: it is merely a possibility for future. The reason the space is not needed is because this is within the realm of Scalar-extended (SVP64Single), and with the 24-bit prefix area being all-zero (bits 8-31) this is defined as "having no augmentation" (in the Simple-V Specification it is termed `Scalar Identity Behaviour`). This in turn makes this prefix a *degenerate duplicate* so may be allocated for other purposes.

0-5	6	7	8-31	32-63
PO (9)?	1	0	0000	EXT300-363 or `RESERVED1`

{EXT200-263} bit6=new bit7=scalar

This encoding represents the opportunity to introduce EXT200-263. It is a Scalar-word encoding, and does not require implementing SVP64 or SVP64-Single. PO2 is in the range 0b00000 to 0b11111 to represent EXT200-263 respectively.

0-5	6	7	8-31	32-37	38-63
PO (9)?	0	0	0000	PO2	{EXT200-263}

SVP64Single:{EXT200-263} bit6=new bit7=scalar

This encoding, which is effectively "implicit VL=1" and comprising (from bits 8-31) *at least some* form of Augmentation, it represents the opportunity to Augment EXT200-263 with the SVP64Single capabilities. Instructions may not be placed in this category without also being implemented as pure Scalar.

0-5	6	7	8-31	32-37	38-63
PO (9)?	0	0	!zero	PO2	SVP64Single:{EXT200-263}

SVP64Single:{EXT000-063} bit6=old bit7=scalar

This encoding, identical to SVP64Single:{EXT200-263}, introduces SVP64Single Augmentation of v3.0 Scalar word instructions. All meanings must be identical to EXT000 to EXT063, and is is likewise prohibited to add an instruction in this area without also adding the exact same (non-Augmented) instruction in EXT000-063 with the exact same Scalar word. PO2 is in the range 0b00000 to 0b11111 to represent EXT000-063 respectively. Augmenting EXT001 is prohibited.

0-5	6	7	8-31	32-37	38-63
PO (9)?	1	0	!zero	PO2	SVP64Single:{EXT000-063}

SVP64:{EXT200-263} bit6=new bit7=vector

This encoding, which permits VL to be dynamic (settable from GPR or CTR) is the Vectorisation of EXT200-263. Instructions may not be placed in this category without also being implemented as pure Scalar *and* SVP64Single. Unlike SVP64Single however, there is **no reserved encoding** (bits 8-24 zero). VL=1 may occur dynamically at runtime, even when bits 8-31 are zero.

0-5	6	7	8-31	32-37	38-63
PO (9)?	0	1	nnnn	PO2	SVP64:{EXT200-263}

SVP64:{EXT000-063} bit6=old bit7=vector

This encoding is identical to **SVP64:{EXT200-263}** except it is the Vectorisation of existing v3.0/3.1 Scalar-words, EXT000-063. All the same rules apply with the addition that Vectorisation of EXT001 is prohibited.

0-5	6	7	8-31	32-37	38-63
PO (9)?	1	1	nnnn	PO2	SVP64:{EXT000-063}

Use cases

In the following examples the programs are fully executable under the Libre-SOC Simple-V-augmented Power ISA Simulator. Reproducible (scripted) Installation instructions: https://libre-soc.org/HDL_workflow/devscripts/

LD/ST-Multi

Context-switching saving and restoring of registers on the stack often requires explicit loop-unrolling to achieve effectively. In SVP64 it is possible to use a Predicate Mask to "compact" or "expand" a swathe of desired registers, dynamically. Known as "VCOMPRESS" and "VEXPAND", runtime-configurable LD/ST-Multi is achievable with 2 instructions.

```
# load 64 registers off the stack, in-order, skipping unneeded ones
# by using CR0-CR63's "EQ" bits to select only those needed.
setvli 64
sv.ld/sm=EQ *rt,0(ra)
```

Twin-Predication, re-entrant

This example demonstrates two key concepts: firstly Twin-Predication (separate source predicate mask from destination predicate mask) and that sufficient state is stored within the Vector Context SPR, SVSTATE, for full re-entrancy on a Context Switch or function call *even if in the middle of executing a loop*. Also demonstrates that it is permissible for a programmer to write **directly** to the SVSTATE SPR, and still expect Deterministic Behaviour. It's not exactly recommended (performance may be impacted by direct SVSTATE access), but it is not prohibited either.

```
292     # checks that we are able to resume in the middle of a VL loop,
293     # after an interrupt, or after the user has updated src/dst step
294     # let's assume the user has prepared src/dst step before running this
295     # vector instruction
296     # test_intpred_reentrant
297         #   reg num        0 1 2 3 4 5 6 7 8 9 10 11 12
298         #   srcstep=1                         v
299         #   src r3=0b0101                   Y  N  Y  N
300         #                                   :     |
301         #                               + - - +     |
302         #                               :   +-------+
303         #                               :   |
304         #   dest ~r3=0b1010          N Y N Y
305         #   dststep=2                   ^
306
307         sv.extsb/sm=r3/dm=~r3 *5, *9
```

https://git.libre-soc.org/?p=openpower-isa.git;a=blob;f=src/openpower/decoder/isa/test_caller_svp64_predication.py;hb=HEAD

3D GPU style "Branch Conditional"

(*Note: Specification is ready, Simulator still under development of full specification capabilities*) This example demonstrates a 2-long Vector Branch-Conditional only succeeding if *all* elements in the Vector are successful. This avoids the need for additional instructions that would need to perform a Parallel Reduction of a Vector of Condition Register tests down to a single value, on which a Scalar Branch-Conditional could then be performed. Full Rationale at https://libre-soc.org/openpower/sv/branches/

```
80   # test_sv_branch_cond_all
81       for i in [7, 8, 9]:
83               addi 1, 0, i+1        # set r1 to i
84               addi 2, 0, i          # set r2 to i
85               cmpi cr0, 1, 1, 8     # compare r1 with 10 and store to cr0
86               cmpi cr1, 1, 2, 8     # compare r2 with 10 and store to cr1
87               sv.bc/all 12, *1, 0xc # bgt 0xc - branch if BOTH
88                                     # r1 AND r2 greater 8 to the nop below
89               addi 3, 0, 0x1234,    # if tests fail this shouldn't execute
90               or 0, 0, 0            # branch target
```

https://git.libre-soc.org/?p=openpower-isa.git;a=blob;f=src/openpower/decoder/isa/test_caller_svp64_bc.py;hb=HEAD

DCT

DCT has dozens of uses in Audio-Visual processing and CODECs. A full 8-wide in-place triple-loop Inverse DCT may be achieved in 8 instructions. Expanding this to 16-wide is a matter of setting `svshape 16` **and the same instructions used**. Lee Composition may be deployed to construct non-power-two DCTs. The cosine table may be computed (once) with 18 Vector instructions (one of them `fcos`)

```
1014     # test_sv_remap_fpmadds_ldbrev_idct_8_mode_4
1015                         # LOAD bit-reversed with half-swap
1016                         svshape 8, 1, 1, 14, 0
1017                         svremap 1, 0, 0, 0, 0, 0, 0
1018                         sv.lfs/els *0, 4(1)
1019                         # Outer butterfly, iterative sum
1020                         svremap 31, 0, 1, 2, 1, 0, 1
1021                         svshape 8, 1, 1, 11, 0
1022                         sv.fadds *0, *0, *0
1023                         # Inner butterfly, twin +/- MUL-ADD-SUB
1024                         svshape 8, 1, 1, 10, 0
1025                         sv.ffmadds *0, *0, *0, *8
```

https://git.libre-soc.org/?p=openpower-isa.git;a=blob;f=src/openpower/decoder/isa/test_caller_svp64_dct.py;hb=HEAD

Matrix Multiply

Matrix Multiply of any size (non-power-2) up to a total of 127 operations is achievable with only three instructions. Normally in any other SIMD ISA at least one source requires Transposition and often massive rolling repetition of data is required. These 3 instructions may be used as the "inner triple-loop kernel" of the usual 6-loop Massive Matrix Multiply.

```
  28     # test_sv_remap1   5x4 by 4x3 matrix multiply
  29                         svshape 5, 4, 3, 0, 0
  30                         svremap 31, 1, 2, 3, 0, 0, 0
  31                         sv.fmadds *0, *8, *16, *0
```

https://git.libre-soc.org/?p=openpower-isa.git;a=blob;f=src/openpower/decoder/isa/test_caller_svp64_matrix.py;hb=HEAD

Parallel Reduction

Parallel (Horizontal) Reduction is often deeply problematic in SIMD and Vector ISAs. Parallel Reduction is Fully Deterministic in Simple-V and thus may even usefully be deployed on non-associative and non-commutative operations.

```
  75     # test_sv_remap2
  76                         svshape 7, 0, 0, 7, 0
  77                         svremap 31, 1, 0, 0, 0, 0, 0 # different order
  78                         sv.subf *0, *8, *16
  79
  80                 REMAP sv.subf RT,RA,RB - inverted application of RA/RB
  81                                         left/right due to subf
```

https://git.libre-soc.org/?p=openpower-isa.git;a=blob;f=src/openpower/decoder/isa/test_caller_svp64_parallel_reduce.py;hb=HEAD

[[!tag opf_rfc]]