External RFC ls012: Discuss priorities of Libre-SOC Scalar(Vector) ops

Date: 2023apr10. v1

- Funded by NLnet Grants under EU Horizon Grants 101069594 825310

- https://git.openpower.foundation/isa/PowerISA/issues/121
- https://bugs.libre-soc.org/show_bug.cgi?id=1051
- https://bugs.libre-soc.org/show_bug.cgi?id=1052

The purpose of this RFC is:

- to give a full list of upcoming Scalar opcodes developed by Libre-SOC (being cognisant that *all* of them are Vectoriseable)
- to give OPF Members and non-Members alike the opportunity to comment and get involved early in RFC submission
- formally agree a priority order on an iterative basis with new versions of this RFC,
- which ones should be EXT022 Sandbox, which in EXT0xx, which in EXT2xx, which not proposed at all,
- keep readers summarily informed of ongoing RFC submissions, with new versions of this RFC,
- for IBM (in their capacity as Allocator of Opcodes) to get a clear advance picture of Opcode Allocation *prior* to submission

As this is a Formal ISA RFC the evaluation shall ultimately define (in advance of the actual submission of the instructions themselves) which instructions will be submitted over the next 1-18 months.

It is expected that readers visit and interact with the Libre-SOC resources in order to do due-diligence on the prioritisation evaluation. Otherwise the ISA WG is overwhelmed by "drip-fed" RFCs that may turn out not to be useful, against a background of having no guiding overview or pre-filtering, and everybody's precious time is wasted. Also note that the Libre-SOC Team, being funded by NLnet under Privacy and Enhanced Trust Grants, are ***prohibited*** *from signing Commercial-Confidentiality NDAs, as doing so is a direct conflict of interest with their funding body's Charitable Foundation Status and remit, and therefore the* ***entire*** *set of almost 150 new SFFS instructions can only go via the External RFC Process. Also be advised and aware that "Libre-SOC" != "RED Semiconductor Ltd". The two are completely* ***separate*** *organisations.*

Worth bearing in mind during evaluation that every "Defined Word" may or may not be Vectoriseable, but that every "Defined Word" should have merits on its own, not just when Vectorised. An example of a borderline Vectoriseable Defined Word is `mv.swizzle` which only really becomes high-priority for Audio/Video, Vector GPU and HPC Workloads, but has less merit as a Scalar-only operation, yet when SVP64Single-Prefixed can be part of an atomic Compare-and-Swap sequence.

Although one of the top world-class ISAs, Power ISA Scalar (SFFS) has not been significantly advanced in 12 years: IBM's primary focus has understandably been on PackedSIMD VSX. Unfortunately, with VSX being 914 instructions and 128-bit it is far too much for any new team to consider (10+ years development effort) and far outside of Embedded or Tablet/Desktop/Laptop power budgets. Thus bringing Power Scalar up-to-date to modern standards *and on its own merits* is a reasonable goal, and the advantages of the reduced focus is that SFFS remains RISC-paradigm, with lessons being be learned from other ISAs from the intervening years. Good examples here include `bmask`.

SVP64 Prefixing - also known by the terms "Zero-Overhead-Loop-Prefixing" as well as "True-Scalable-Vector Prefixing" - also literally brings new dimensions to the Power ISA. Thus when adding new Scalar "Defined Words" it has to unavoidably and simultaneously be taken into consideration their value when Vector-Prefixed, *as well as* SVP64Single-Prefixed.

Target areas

Whilst entirely general-purpose there are some categories that these instructions are targetting: Bit-manipulation, Big-integer, cryptography, Audio/Visual, High-Performance Compute, GPU workloads and DSP.

Instruction count guide and approximate priority order

- 6 - SVP64 Management {RFC ls008} {RFC ls009} {RFC ls010}
- 5 - CR weirds {CR Weird ops}
- 4 - INT<->FP mv {RFC ls006}
- 19 - GPR LD/ST-PostIncrement-Update (saves hugely in hot-loops) {RFC ls011}
- ~12 - FPR LD/ST-PostIncrement-Update (ditto) {RFC ls011}
- 2 - Float-Load-Immediate (always saves one LD L1/2/3 D-Cache op) {RFC ls002}
- 5 - Big-Integer Chained 3-in 2-out (64-bit Carry) {Big Integer}
- 6 - Bitmanip LUT2/3 operations. high cost high reward {Bitmanip ops}
- 1 - fclass (Scalar variant of xvtstdcsp) {FP Class ops}
- 5 - Audio-Video {Audio and Video Opcodes}
- 2 - Shift-and-Add (mitigates LD-ST-Shift; Cryptography e.g. twofish) {RFC ls004}
- 2 - BMI group {SV Vector ops}
- 2 - GPU swizzle {Swizzle Move}
- 9 - FP DCT/FFT Butterfly (2/3-in 2-out)
- ~9 Integer DCT/FFT Butterfly https://bugs.libre-soc.org/show_bug.cgi?id=1028
- 18 - Trigonometric (1-arg) {Transcendentals}
- 15 - Transcendentals (1-arg) {Transcendentals}
- 25 - Transcendentals (2-arg) {Transcendentals}

Summary tables are created below by different sort categories. Additional columns (and tables) as necessary can be requested to be added as part of update revisions to this RFC.

Target Area summaries

Please note that there are some instructions developed thanks to NLnet funding that have not been included here for assessment. Examples include `pcdec` and the Galois Field arithmetic operations. From a purely practical perspective due to the quantity the lower-priority instructions were simply left out. However they remain in the Libre-SOC resources.

Some of these SFFS instructions appear to be duplicates of VSX. A frequent argument comes up that if instructions are in VSX already they should not be added to SFFS, especially if they are nominally the same. The logic that this effectively damages performance of an SFFS-only implementation was raised earlier, however there is a more subtle reason why the instructions are needed.

Future versions of SVP64 and SVP64Single are expected to be developed by future Power ISA Stakeholders on top of VSX. The decisions made there about the meaning of Prefixed Vectorised VSX may be **completely** different from those made for Prefixed SFFS instructions. At which point the lack of SFFS equivalents would penalise SFFS implementors in a much more severe way, effectively expecting them and SFFS programmers to work with a non-orthogonal paradigm, to their detriment. The solution is to give the SFFS Subset the space and respect that it deserves and allow it to be stand-alone on its own merits.

SVP64 Management instructions

These without question have to go in EXT0xx. Future extended variants, bringing even more powerful capabilities, can be followed up later with EXT1xx prefixed variants, which is not possible if placed in EXT2xx. *Only* ***`svstep`*** *is actually Vectoriseable*, all other Management instructions are UnVectoriseable. PO1-Prefixed examples include adding psvshape in order to support both Inner and Outer Product Matrix Schedules, by providing the option to directly reverse the order of the triple loops. Outer is used for standard Matrix Multiply (on top of a standard MAC or FMAC instruction), but Inner is required for Warshall Transitive Closure (on top of a cumulatively-applied max instruction).

The Management Instructions themselves are all Scalar Operations, so PO1-Prefixing is perfectly reasonable. SVP64 Management instructions of which there are only 6 are all 5 or 6 bit XO, meaning that the opcode space they take up in EXT0xx is not alarmingly high for their intrinsic strategic value.

Transcendentals

Found at {Transcendentals} these subdivide into high priority for accelerating general-purpose and High-Performance Compute, specialist 3D GPU operations suited to 3D visualisation, and low-priority less common instructions where IEEE754 full bit-accuracy is paramount. In 3D GPU scenarios for example even 12-bit accuracy can be overkill, but for HPC Scientific scenarios 12-bit would be disastrous.

There are a **lot** of operations here, and they also bring Power ISA up-to-date to IEEE754-2019. Fortunately the number of critical instructions is quite low, but the caveat is that if those operations are utilised to synthesise other IEEE754 operations (divide by `pi` for example) full bit-level accuracy (a hard requirement for IEEE754) is lost.

Also worth noting that the Khronos Group defines minimum acceptable bit-accuracy levels for 3D Graphics: these are **nowhere near** the full accuracy demanded by IEEE754, the reason for the Khronos definitions is a massive reduction often four-fold in power consumption and gate count when 3D Graphics simply has no need for full accuracy.

For 3D GPU markets this definitely needs addressing

Audio/Video

Found at {Audio and Video Opcodes} these do not require Saturated variants because Saturation is added via {SVP64 Chapter} (Vector Prefixing) and via [[sv/svp64_single]] Scalar Prefixing. This is important to note for Opcode Allocation because placing these operations in the UnVectoriseable areas would irredeemably damage their value. Unlike PackedSIMD ISAs the actual number of AV Opcodes is remarkably small once the usual cascading-option-multipliers (SIMD width, bitwidth, saturation, HI/LO) are abstracted out to RISC-paradigm Prefixing, leaving just absolute-diff-accumulate, min-max, average-add etc. as "basic primitives".

Twin-Butterfly FFT/DCT/DFT for DSP/HPC/AI/AV

The number of uses in Computer Science for DCT, NTT, FFT and DFT, is astonishing. The wikipedia page lists over a hundred separate and distinct areas: Audio, Video, Radar, Baseband processing, AI, Solomon-Reed Error Correction, the list goes on and on. ARM has special dedicated Integer Twin-butterfly instructions. TI's MSP Series DSPs have had FFT Inner loop support for over 30 years. Qualcomm's Hexagon VLIW Baseband DSP can do full FFT triple loops in one VLIW group.

It should be pretty clear this is high priority.

With SVP64 {REMAP subsystem} providing the Loop Schedules it falls to the Scalar side of the ISA to add the prerequisite "Twin Butterfly" operations, typically performing for example one multiply but in-place subtracting that product from one operand and adding it to the other. The *in-place* aspect is strategically extremely important for significant reductions in Vectorised register usage, particularly for DCT.

CR Weird group

Outlined in {CR Weird ops} these instructions massively save on CR-Field instruction count. Multi-bit to single-bit and vice-versa normally requiring several CR-ops (crand, crxor) are done in one single instruction. The reason for their addition is down to SVP64 overloading CR Fields as Vector Predicate Masks. Reducing instruction count in hot-loops is considered high priority.

An additional need is to do popcount on CR Field bit vectors but adding such instructions to the *Condition Register* side was deemed to be far too much. Therefore, priority was given instead to transferring several CR Field bits into GPRs, whereupon the full set of Standard Scalar GPR Logical Operations may be used. This strategy has the side-effect of keeping the CRweird group down to only five instructions.

Big-integer Math

{Big Integer} has always been a high priority area for commercial applications, privacy, Banking, as well as HPC Numerical Accuracy: libgmp as well as cryptographic uses in Asymmetric Ciphers. poly1305 and ec25519 are finding their way into everyday use via OpenSSL.

A very early variant of the Power ISA had a 32-bit Carry-in Carry-out SPR. Its removal from subsequent revisions is regrettable. An alternative concept is to add six explicit 3-in 2-out operations that, on close inspection, always turn out to be supersets of *existing Scalar operations* that discard upper or lower DWords, or parts thereof.

Thus it is critical to note that not one single one of these operations expands the bitwidth of any existing Scalar pipelines.

The `dsld` instruction for example merely places additional LSBs into the 64-bit shift (64-bit carry-in), and then places the (normally discarded) MSBs into the second output register (64-bit carry-out). It does **not** require a 128-bit shifter to replace the existing Scalar Power ISA 64-bit shifters.

The reduction in instruction count these operations bring, in critical hot loops, is remarkably high, to the extent where a Scalar-to-Vector operation of *arbitrary length* becomes just the one Vector-Prefixed instruction.

Whilst these are 5-6 bit XO their utility is considered high strategic value and as such are strongly advocated to be in EXT04. The alternative is to bring back a 64-bit Carry SPR but how it is retrospectively applicable to pre-existing Scalar Power ISA multiply, divide, and shift operations at this late stage of maturity of the Power ISA is an entire area of research on its own deemed unlikely to be achievable.

fclass and GPR-FPR moves

{FP Class ops} - just one instruction. With SFFS being locked down to exclude VSX, and there being no desire within the nascent OpenPOWER ecosystem outside of IBM to implement the VSX PackedSIMD paradigm, it becomes necessary to upgrade SFFS such that it is stand-alone capable. One omission based on the assumption that VSX would always be present is an equivalent to `xvtstdcsp`.

Similar arguments apply to the GPR-INT move operations, proposed in {RFC ls006}, with the opportunity taken to add rounding modes present in other ISAs that Power ISA VSX PackedSIMD does not have. Javascript rounding, one of the worst offenders of Computer Science, requires a phenomenal 35 instructions with *six branches* to emulate in Power ISA! For desktop as well as Server HTML/JS back-end execution of javascript this becomes an obvious priority, recognised already by ARM as just one example.

Bitmanip LUT2/3

These LUT2/3 operations are high cost high reward. Outlined in {Bitmanip ops}, the simplest ones already exist in PackedSIMD VSX: `xxeval`. The same reasoning applies as to fclass: SFFS needs to be stand-alone on its own merits and should an implementor choose not to implement any aspect of PackedSIMD VSX the performance of their product should not be penalised for making that decision.

With Predication being such a high priority in GPUs and HPC, CR Field variants of Ternary and Binary LUT instructions were considered high priority, and again just like in the CRweird group the opportunity was taken to work on *all* bits of a CR Field rather than just one bit as is done with the existing CR operations crand, cror etc.

The other high strategic value instruction is `grevlut` (and `grevluti` which can generate a remarkably large number of regular-patterned magic constants). The grevlut set require of the order of 20,000 gates but provide an astonishing plethora of innovative bit-permuting instructions never seen in any other ISA.

The downside of all of these instructions is the extremely low XO bit requirements: 2-3 bit XO due to the large immediates *and* the number of operands required. The LUT3 instructions are already compacted down to "Overwrite" variants. (By contrast the Float-Load-Immediate instructions are a much larger XO because despite having 16-bit immediate only one Register Operand is needed).

Realistically these high-value instructions should be proposed in EXT2xx where their XO cost does not overwhelm EXT0xx.

(f)mv.swizzle

{Swizzle Move} is dicey. It is a 2-in 2-out operation whose value as a Scalar instruction is limited *except* if combined with `cmpi` and SVP64Single Predication, whereupon the end result is the RISC-synthesis of Compare-and-Swap, in two instructions.

Where this instruction comes into its full value is when Vectorised. 3D GPU and HPC numerical workloads astonishingly contain between 10 to 15% swizzle operations: access to YYZ, XY, of an XYZW Quaternion, performing balancing of ARGB pixel data. The usage is so high that 3D GPU ISAs make Swizzle a first-class priority in their VLIW words. Even 64-bit Embedded GPU ISAs have a staggering 24-bits dedicated to 2-operand Swizzle.

So as not to radicalise the Power ISA the Libre-SOC team decided to introduce mv Swizzle operations, which can always be Macro-op fused in exactly the same way that ARM SVE predicated-move extends 3-operand "overwrite" opcodes to full independent 3-in 1-out.

BMI (bit-manipulation) group.

Whilst the {SV Vector ops} instructions are only two in number, in reality the `bmask` instruction has a Mode field allowing it to cover **24** instructions, more than have been added to any other CPUs by ARM, Intel or AMD. Analysis of the BMI sets of these CPUs shows simple patterns that can greatly simplify both Decode and implementation. These are sufficiently commonly used, saving instruction count regularly, that they justify going into EXT0xx.

The other instruction is `cprop` - Carry-Propagation - which takes the P and Q from carry-propagation algorithms and generates carry look-ahead. Greatly increases the efficiency of arbitrary-precision integer arithmetic by combining what would otherwise be half a dozen instructions into one. However it is still not a huge priority unlike `bmask` so is probably best placed in EXT2xx.

Float-Load-Immediate

Very easily justified. As explained in {RFC ls002} these always saves one LD L1/2/3 D-Cache memory-lookup operation, by virtue of the Immediate FP value being in the I-Cache side. It is such a high priority that these instructions are easily justifiable adding into EXT0xx, despite requiring a 16-bit immediate. By designing the second-half instruction as a Read-Modify-Write it saves on XO bit-length (only 5 bits), and can be macro-op fused with its first-half to store a full IEEE754 FP32 immediate into a register.

There is little point in putting these instructions into EXT2xx. Their very benefit and inherent value *is* as 32-bit instructions, not 64-bit ones. Likewise there is less value in taking up EXT1xx Encoding space because EXT1xx only brings an additional 16 bits (approx) to the table, and that is provided already by the second-half instruction.

Thus they qualify as both high priority and also EXT0xx candidates.

FPR/GPR LD/ST-PostIncrement-Update

These instruction, outlined in {RFC ls011}, save hugely in hot-loops. Early ISAs such as PDP-8, PDP-11, which inspired the iconic Motorola 68000, 88100, Mitch Alsup's MyISA 66000, and can even be traced back to the iconic ultra-RISC CDC 6600, all had both pre- and post- increment Addressing Modes.

The reason is very simple: it is a direct recognition of the practice in c to frequently utilise both `*p++` and `*++p` which itself stems from common need in Computer Science algorithms.

The problem for the Power ISA is - was - that the opcode space needed to support both was far too great, and the decision was made to go with pre-increment, on the basis that outside the loop a "pre-subtraction" may be performed.

Whilst this is a "solution" it is less than ideal, and the opportunity exists now with the EXT2xx Primary Opcodes to correct this and bring Power ISA up a level.

Shift-and-add

Shift-and-Add are proposed in {RFC ls004}. They mitigate the need to add LD-ST-Shift instructions which are a high-priority aspect of both x86 and ARM. LD-ST-Shift is normally just the one instruction: Shift-and-add brings that down to two, where Power ISA presently requires three. Cryptography e.g. twofish also makes use of Integer double-and-add, so the value of these instructions is not limited to Effective Address computation. They will also have value in Audio DSP.

Being a 10-bit XO it would be somewhat punitive to place these in EXT2xx when their whole purpose and value is to reduce binary size in Address offset computation, thus they are best placed in EXT0xx.

Vectorisation: SVP64 and SVP64Single

To be submitted as part of {RFC ls001}, {RFC ls008}, {RFC ls009} and {RFC ls010}, with SVP64Single to follow in a subsequent RFC, SVP64 is conceptually identical to the 50+ year old 8080 `REP` instruction and the Zilog Z80 `CPIR` and `LDIR` instructions. Parallelism is best achieved by exploiting a Multi-Issue Out-of-Order Micro-architecture. It is extremely important to bear in mind that at no time does SVP64 add even one single actual Vector instruction. It is a *pure* RISC-paradigm Prefixing concept only.

This has some implications which need unpacking. Firstly: in the future, the Prefixing may be applied to VSX. The only reason it was not included in the initial proposal of SVP64 is because due to the number of VSX instructions the Due Diligence required is obviously five times higher than the 3+ years work done so far on the SFFS Subset.

Secondly: **any** Scalar instruction involving registers **automatically** becomes a candidate for Vector-Prefixing. This in turn means that when a new instruction is proposed, it becomes a hard requirement to consider not only the implications of its inclusion as a Scalar-only instruction, but how it will best be utilised as a Vectorised instruction **as well**. Extreme examples of this are the Big-Integer 3-in 2-out instructions that use one 64-bit register effectively as a Carry-in and Carry-out. The instructions were designed in a *Scalar* context to be inline-efficient in hardware (use of Operand-Forwarding to reduce the chain down to 2-in 1-out), but in a *Vector* context it is extremely straightforward to Micro-code an entire batch onto 128-bit SIMD pipelines, 256-bit SIMD pipelines, and to perform a large internal Forward-Carry-Propagation on for example the Vectorised-Multiply instruction.

Thirdly: as far as Opcode Allocation is concerned, SVP64 needs to be considered as an independent stand-alone instruction (just like `REP`). In other words, the Suffix **never** gets decoded as a completely different instruction just because of the Prefix. The cost of doing so is simply too high in hardware.

Guidance for evaluation

Deciding which instructions go into an ISA is extremely complex, costly, and a huge responsibility. In public standards mistakes are irrevocable, and in the case of an ISA the Opcode Allocation is a finite resource, meaning that mistakes punish future instructions as well. This section therefore provides some Evaluation Guidance on the decision process, particularly for people new to ISA development, given that this RFC is circulated widely and publicly. Constructive feedback from experienced ISA Architects welcomed to improve this section.

Does anyone want it?

Sounds like an obvious question but if there is no driving need (no "Stakeholder") then why is the instruction being proposed? If it is purely out of curiosity or part of a Research effort not intended for production then it's probably best left in the EXT022 Sandbox.

How many registers does it need?

The basic RISC Paradigm is not only to make instruction encoding simple (often "wasting" encoding space compared to highly-compacted ISAs such as x86), but also to keep the number of registers used down to a minimum.

Counter-examples are FMAC which had to be added to IEEE754 because the *internal* product requires more accuracy than can fit into a register (it is well-known that FMUL followed by FADD performs an additional rounding on the intermediate register which loses accuracy compared to FMAC). Another would be a dot-product instruction, which again requires an accumulator of at least double the width of the two vector inputs. And in the AMDGPU ISA, there are Texture-mapping instructions taking up to an astounding *twelve* input operands!

The downside of going too far however has to be a trade-off with the next question. Both MIPS and RISC-V lack Condition Codes, which means that emulating x86 Branch-Conditional requires *ten* MIPS instructions.

The downside of creating too complex instructions is that the Dependency Hazard Management in high-performance multi-issue out-of-order microarchitectures becomes infeasibly large, and even simple in-order systems may have performance severely compromised by an overabundance of stalls. Also worth remembering is that register file ports are insanely costly, not just to design but also use considerable power.

That said there do exist genuine reasons why more registers is better than less: Compare-and-Swap has huge benefits but is costly to implement, and DCT/FFT Twin-Butterfly instructions allow creation of in-place in-register algorithms reducing the number of registers needed and thus saving power due to making the *overall* algorithm more efficient, as opposed to micro-focussing on a localised power increase.

How many register files does it use?

Complex instructions pulling in data from multiple register files can create unnecessary issues surrounding Dependency Hazard Management in Out-of-Order systems. As a general rule it is better to keep complex instructions reading and writing to the same register file, relying on much simpler (1-in 1-out) instructions to transfer data between register files.

Can other existing instructions (plural) do the same job

The general rule being: if two or more instructions can do the same job, leave it out... *unless* the number of occurrences of that instruction being missing is causing huge increases in binary size. RISC-V has gone too far in this regard, as explained here: https://news.ycombinator.com/item?id=24459314

Good examples are LD-ST-Indexed-shifted (multiply RB by 2, 4 8 or 16) which are high-priority instructions in x86 and ARM, but lacking in Power ISA, MIPS, and RISC-V. With many critical hot-loops in Computer Science having to perform shift and add as explicit instructions, adding LD/ST-shifted should be considered high priority, except that the sheer *number* of such instructions needing to be added takes us into the next question

How costly is the encoding?

This can either be a single instruction that is costly (several operands or a few long ones) or it could be a group of simpler ones that purely due to their number increases overall encoding cost. An example of an extreme costly instruction would be those with their own Primary Opcode: addi is a good candidate. However the sheer overwhelming number of times that instruction is used easily makes a case for its inclusion.

Mentioned above was Load-Store-Indexed-Shifted, which only needs 2 bits to specify how much to shift: x2 x4 x8 or x16. And they are all a 10-bit XO Field, so not that costly for any one given instruction. Unfortunately there are *around 30* Load-Store-Indexed Instructions in the Power ISA, which means an extra *five* bits taken up of precious XO space. Then let us not forget the two needed for the Shift amount. Now we are up to *three* bit XO for the group.

Is this a worthwhile tradeoff? Honestly it could well be. And that's the decision process that the OpenPOWER ISA Working Group could use some assistance on, to make the evaluation easier.

How many gates does it need?

`grevlut` comes in at an astonishing 20,000 gates, where for comparison an FP64 Multiply typically takes between 12 to 15,000. Not counting the cost in hardware terms is just asking for trouble.

How long will it take to complete?

In the case of divide or Transcendentals the algorithms needed are so complex that simple implementations can often take an astounding 128 clock cycles to complete. Other instructions waiting for the results will back up and eventually stall, where in-order systems pretty much just stall straight away.

Less extreme examples include instructions that take only a few cycles to complete, but if used in tight loops with Conditional Branches, an Out-of-Order system with Speculative capability may need significantly more Reservation Stations to hold in-flight data for instructions which take longer than those which do not.

Can one instruction do the job of many?

Large numbers of disparate instructions adversely affects resource utilisation in In-Order systems. However it is not always that simple: every one of the Power ISA "add" and "subtract" instructions, as shown by the Microwatt source code, may be micro-coded as one single instruction where RA may optionally be inverted, output likewise, and Carry-In set to 1, 0 or XER.CA. From these options the *entire* suite of add/subtract may be synthesised (subtract by inverting RA and adding an extra 1 it produces a 2s-complement of RA).

`bmask` for example is to be proposed as a single instruction with a 5-bit "Mode" operand, greatly simplifying some micro-architectural implementations. Likewise the FP-INT conversion instructions are grouped as a set of four, instead of over 30 separate instructions. Aside from anything this strategy makes the ISA Working Group's evaluation task easier, as well as reducing the work of writing a Compliance Test Suite.

Summary

There are many tradeoffs here, it is a huge list of considerations: any others known about please do submit feedback so they may be included, here. Then the evaluation process may take place: again, constructive feedback on that as to which instructions are a priority also appreciated. The above helps explain the columns in the tables that follow.

Tables

The original tables are available publicly as as CSV file at https://git.libre-soc.org/?p=libreriscv.git;a=blob;f=openpower/sv/rfc/ls012/optable.csv;hb=HEAD. A python program auto-generates the tables in the following sections by sorting into different useful priorities.

The key to headings and sections are as follows:

- **Area** - Target Area as described in above sections
- **XO Cost** - the number of bits required in the XO Field. whilst not the full picture it is a good indicator as to how costly in terms of Opcode Allocation a given instruction will be. Lower number is a higher cost for the Power ISA's precious remaining Opcode space. "PO" indicates that an entire Primary Opcode is required.
- **rfc** the Libre-SOC External RFC resource, https://libre-soc.org/openpower/sv/rfc/ where advance notice of upcoming RFCs in development may be found. *Reading advance Draft RFCs and providing feedback strongly advised*, it saves time and effort for the OPF ISA Workgroup.
- **SVP64** - Vectoriseable (SVP64-Prefixable) - also implies that SVP64Single is also permitted (required).
- **page** - Libre-SOC wiki page at which further information can be found. Again: **advance reading strongly advised due to the sheer volume of information**.
- **PO1** - the instruction is capable of being PO1-Prefixed (given an EXT1xx Opcode Allocation). Bear in mind that this option is **mutually exclusively incompatible** with Vectorisation.
- **group** - the Primary Opcode Group recommended for this instruction. Options are EXT0xx (EXT000-EXT063), EXT1xx and EXT2xx. A third area (UnVectoriseable), EXT3xx, was available in an early Draft RFC but has been made "RESERVED" instead. see [[sv/po9_encoding]].
- **regs** - a guide to register usage, to how costly Hazard Management will be, in hardware:

  - `1R: reads one GPR/FPR/SPR/CR.`
  - `1W: writes one GPR/FPR/SPR/CR.`
  - `1r: reads one CR *Field* (not necessarily the entire CR)`
  - `1w: writes one CR *Field* (not necessarily the entire CR)`

Areas

LD/ST-Postincrement

op	rfc	priority	cost	SVP64	group	PO1	page	regs
lbzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lbzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lhzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lhzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lhaup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lhaupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lwzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lwzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lwaupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
ldup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
ldupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
stbup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stbupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
sthup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
sthupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stwup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stwupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stdup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stdupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W

FP LD/ST-Postincrement

op	rfc	priority	cost	SVP64	group	PO1	page	regs
lfdu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lfsu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lfdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lsdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
stfdu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stfsu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stfdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stfsux	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W

Bitmanip LUT2/3 operations. high cost high reward

op	rfc	priority	cost	SVP64	group	PO1	page	regs
grevlut	TBD	high	3	yes	TBD	no	sv/bitmanip	2R1W
grevluti	TBD	high	3	yes	TBD	yes	sv/bitmanip	1R1W
ternlogi	ls007	high	2	yes	TBD	yes	sv/bitmanip	3R1W1w
crternlogi	ls007	high	5	yes	TBD	yes	sv/bitmanip	3r1w
binlut	ls007	high	6	yes	TBD	no	sv/bitmanip	3R1W
crbinlut	ls007	high	5	yes	TBD	no	sv/bitmanip	3r1w

Float-Load-Immediate (always saves one LD L1/2/3 D-Cache op)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
fmvis	ls002	high	5	yes	TBD	no	sv/bitmanip	1W
fishmv	ls002	high	5	yes	TBD	no	sv/bitmanip	1R1W

Shift-and-Add (mitigates LD-ST-Shift; Cryptography e.g. twofish)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
shadd	ls003	med	7	yes	TBD	no	sv/bitmanip	2R1W1w
shadduw	ls003	med	7	yes	TBD	no	sv/bitmanip	2R1W1w

Audio-Video

op	rfc	priority	cost	SVP64	group	PO1	page	regs
absdu	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	2R1W1w
avgadd	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	2R1W1w
minmax	TBD	high	10	yes	TBD	no	sv/av_opcodes	2R1W1w
absaccs	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	3R1W1w
absaccu	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	3R1W1w

BMI group

op	rfc	priority	cost	SVP64	group	PO1	page	regs
bmask	TBD	high	5	yes	TBD	yes	sv/vector_ops	2R1W1w
cprop	TBD	high	5	yes	TBD	yes	sv/vector_ops	2R1W1w

SVP64 Management.

op	rfc	priority	cost	SVP64	group	PO1	page	regs
setvl	ls008	high	5	no	EXT0xx	yes	sv/setvl	3R2W
svstep	ls008	high	5	no	EXT0xx	yes	sv/svstep	1R2W1w
svremap	ls009	high	5	no	EXT0xx	yes	sv/remap	1R1W
svshape	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W
svshape2	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W
svindex	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W

GPU swizzle

op	rfc	priority	cost	SVP64	group	PO1	page	regs
mv.swizzle	TBD	TBD	4	yes	TBD	yes	sv/mv.swizzle	2R2W
fmv.swizzle	TBD	TBD	4	yes	TBD	yes	sv/mv.swizzle	2R2W

CR weirds

op	rfc	priority	cost	SVP64	group	PO1	page	regs
crrweird	TBD	high	8	yes	TBD	no	sv/cr_int_predication	1r1W1w
mfcrweird	TBD	high	8	yes	TBD	no	sv/cr_int_predication	1r1W1w
mtcrrweird	TBD	high	9	yes	TBD	no	sv/cr_int_predication	1R1r1w
mtcrweird	TBD	high	9	yes	TBD	no	sv/cr_int_predication	1R1r1w
crweirder	TBD	high	9	yes	TBD	no	sv/cr_int_predication	2r1w
mcrfm	TBD	high	9	yes	EXT0xx	no	sv/cr_int_predication	2r1w

fclass (Scalar variant of xvtstdcsp)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
fptstp(s)	TBD	high	10	yes	EXT0xx	no	sv/fclass	1R1w

INT<->FP mv

op	rfc	priority	cost	SVP64	group	PO1	page	regs
fmvfg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvtfg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvttg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvtstg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w

Big-Integer Chained 3-in 2-out (64-bit Carry)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
dsld	ls003	high	5	yes	EXT0xx	no	sv/biginteger	3R2W1w
dsrd	ls003	high	5	yes	EXT0xx	no	sv/biginteger	3R2W1w
maddedu	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W
maddedus	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W
divmod2du	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W1w

FP DCT/FFT Butterfly (2/3-in 2-out)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
ffadd(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffsub(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffmul(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffdiv(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
fdmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffmsub(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffnmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffnmsub(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w

Trigonometric (1-arg)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
fsin(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fcos(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
ftan(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fasin(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facos(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatan(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fsinpi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
fcospi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
ftanpi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
fasinpi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facospi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatanpi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fsinh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fcosh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
ftanh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fasinh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facosh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatanh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w

Transcendentals (1-arg)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
frsqrt(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fcbrt(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
frecip(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fexp2m1(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
flog2p1(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fexp2(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
flog2(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fexpm1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flogp1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp10m1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog10p1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp10(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog10(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w

Transcendentals (2-arg)

op	rfc	priority	cost	SVP64	group	PO1	page	regs
fatan2(s)	TBD	med	10	yes	EXT2xx	no	transcendentals	2R1W1w
fatan2pi(s)	TBD	med	10	yes	EXT2xx	no	transcendentals	2R1W1w
fpow(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
fpown(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
fpowr(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
frootn(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
fhypot(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w
fminnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmin19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmax19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmag19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmag19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmod(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w
fremainder(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w

XO cost

PO

op	rfc	priority	cost	SVP64	group	PO1	page	regs
lbzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lhzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lhaup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lwzup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
ldup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lfdu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
lfsu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedload	1R2W
stbup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
sthup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stwup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stdup	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stfdu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W
stfsu	ls011	high	PO	yes	EXT2xx	no	isa/pifixedstore	2R1W

2

op	rfc	priority	cost	SVP64	group	PO1	page	regs
ternlogi	ls007	high	2	yes	TBD	yes	sv/bitmanip	3R1W1w

3

op	rfc	priority	cost	SVP64	group	PO1	page	regs
grevlut	TBD	high	3	yes	TBD	no	sv/bitmanip	2R1W
grevluti	TBD	high	3	yes	TBD	yes	sv/bitmanip	1R1W

4

op	rfc	priority	cost	SVP64	group	PO1	page	regs
mv.swizzle	TBD	TBD	4	yes	TBD	yes	sv/mv.swizzle	2R2W
fmv.swizzle	TBD	TBD	4	yes	TBD	yes	sv/mv.swizzle	2R2W

5

op	rfc	priority	cost	SVP64	group	PO1	page	regs
svremap	ls009	high	5	no	EXT0xx	yes	sv/remap	1R1W
svshape	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W
svshape2	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W
svindex	ls009	high	5	no	EXT0xx	yes	sv/remap	5R5W
setvl	ls008	high	5	no	EXT0xx	yes	sv/setvl	3R2W
svstep	ls008	high	5	no	EXT0xx	yes	sv/svstep	1R2W1w
dsld	ls003	high	5	yes	EXT0xx	no	sv/biginteger	3R2W1w
dsrd	ls003	high	5	yes	EXT0xx	no	sv/biginteger	3R2W1w
crternlogi	ls007	high	5	yes	TBD	yes	sv/bitmanip	3r1w
crbinlut	ls007	high	5	yes	TBD	no	sv/bitmanip	3r1w
fmvis	ls002	high	5	yes	TBD	no	sv/bitmanip	1W
fishmv	ls002	high	5	yes	TBD	no	sv/bitmanip	1R1W
bmask	TBD	high	5	yes	TBD	yes	sv/vector_ops	2R1W1w
cprop	TBD	high	5	yes	TBD	yes	sv/vector_ops	2R1W1w
fdmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffmsub(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffnmadd(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w
ffnmsub(s)	TBD	med	5	yes	EXT2xx	no	isa/svfparith	3R2W1w

6

op	rfc	priority	cost	SVP64	group	PO1	page	regs
maddedu	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W
maddedus	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W
divmod2du	ls003	high	6	yes	EXT0xx	no	sv/biginteger	3R2W1w
binlut	ls007	high	6	yes	TBD	no	sv/bitmanip	3R1W

7

op	rfc	priority	cost	SVP64	group	PO1	page	regs
shadd	ls003	med	7	yes	TBD	no	sv/bitmanip	2R1W1w
shadduw	ls003	med	7	yes	TBD	no	sv/bitmanip	2R1W1w

8

op	rfc	priority	cost	SVP64	group	PO1	page	regs
crrweird	TBD	high	8	yes	TBD	no	sv/cr_int_predication	1r1W1w
mfcrweird	TBD	high	8	yes	TBD	no	sv/cr_int_predication	1r1W1w

9

op	rfc	priority	cost	SVP64	group	PO1	page	regs
mtcrrweird	TBD	high	9	yes	TBD	no	sv/cr_int_predication	1R1r1w
mtcrweird	TBD	high	9	yes	TBD	no	sv/cr_int_predication	1R1r1w
crweirder	TBD	high	9	yes	TBD	no	sv/cr_int_predication	2r1w
mcrfm	TBD	high	9	yes	EXT0xx	no	sv/cr_int_predication	2r1w

10

op	rfc	priority	cost	SVP64	group	PO1	page	regs
lbzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lhzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lhaupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lwzupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lwaupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
ldupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lfdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
lsdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedload	2R2W
stbupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
sthupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stwupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stdupx	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stfdux	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
stfsux	ls011	high	10	yes	EXT2xx	no	isa/pifixedstore	3R1W
minmax	TBD	high	10	yes	TBD	no	sv/av_opcodes	2R1W1w
fptstp(s)	TBD	high	10	yes	EXT0xx	no	sv/fclass	1R1w
fmvfg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvtfg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvttg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fcvtstg(s)	ls006	high	10	yes	EXT0xx	no	sv/int_fp_mv	1R1W1w
fsin(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fcos(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
ftan(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fsinpi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
fcospi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
ftanpi(s)	TBD	high	10	yes	TBD	no	transcendentals	1R1W1w
frsqrt(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
frecip(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fexp2m1(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
flog2p1(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fexp2(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
flog2(s)	TBD	high	10	yes	EXT0xx	no	transcendentals	1R1W1w
fminnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmin19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmax19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagnum08(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmag19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmag19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagnum19(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fminmagc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
fmaxmagc(s)	TBD	high	10	yes	TBD	no	transcendentals	2R1W1w
ffadd(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffsub(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffmul(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
ffdiv(s)	TBD	med	10	yes	EXT2xx	no	isa/svfparith	2R1W1w
fexpm1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flogp1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp10m1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog10p1(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fexp10(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
flog10(s)	TBD	med	10	yes	TBD	no	transcendentals	1R1W1w
fatan2(s)	TBD	med	10	yes	EXT2xx	no	transcendentals	2R1W1w
fatan2pi(s)	TBD	med	10	yes	EXT2xx	no	transcendentals	2R1W1w
fasin(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facos(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatan(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fasinpi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facospi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatanpi(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fsinh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fcosh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w

op	rfc	priority	cost	SVP64	group	PO1	page	regs
ftanh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fasinh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
facosh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fatanh(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fcbrt(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	1R1W1w
fpow(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
fpown(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
fpowr(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
frootn(s)	TBD	low	10	yes	EXT2xx	no	transcendentals	2R1W1w
absdu	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	2R1W1w
avgadd	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	2R1W1w
absaccs	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	3R1W1w
absaccu	TBD	TBD	10	yes	TBD	no	sv/av_opcodes	3R1W1w
fhypot(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w
fmod(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w
fremainder(s)	TBD	TBD	10	yes	TBD	no	transcendentals	2R1W1w

[[!tag opf_rfc]]