RFC ls014 Advanced Scalar Bitmanipulation </>

- Funded by NLnet under the Privacy and Enhanced Trust Programme, EU Horizon2020 Grant 825310, and NGI0 Entrust No 101069594

- https://libre-soc.org/openpower/sv/rfc/ls014/
- https://git.openpower.foundation/isa/PowerISA/issues/TODO
- https://bugs.libre-soc.org/show_bug.cgi?id=1065

Severity: Major

Status: New

Date: 14 Apr 2023

Target: v3.2B

Source: v3.1B

Books and Section affected:

```
Book I Fixed-Point Instructions
Appendix E Power ISA sorted by opcode
Appendix F Power ISA sorted by version
Appendix G Power ISA sorted by Compliancy Subset
Appendix H Power ISA sorted by mnemonic
```

Summary

```
Instructions added: bmask, grevlut, grevluti
```

Submitter: Luke Leighton (Libre-SOC)

Requester: Libre-SOC

Impact on processor:

```
Addition of new GPR-based instructions
```

Impact on software:

```
Requires support for new instructions in assembler, debuggers,
and related tools.
```

Keywords:

```
LUTs, Bitmanipulation, GPR
```

Motivation

Scalar Bitmanipulation in other high-end ISAs have had BMI subsets for over a decade. Their use and benefit is well-understood and compiler integration well-established. `bmask` brings *twenty four* BMI instructions to the Power ISA.

`grevlut` on the other hand is highly experimental and extremely powerful. Normally only `grev` (Generalised Reverse) and occasionally `gor` are added to a Bitmanip-strong ISA: grevlut utilises LUTs and inversion to add 512 Generalised Reverse instructions. Desirable savings in general binary size are achieved.

Notes and Observations:

1. bmask is a synthesis and generalisation of every "TBM" instruction with additional options not found in any other ISA BMI group.
2. grevluti as a 32-bit Defined Word-instruction is capable of generating over a thousand useful regular-patterned 64-bit "magic constants" that otherwise require either a Load or require several instructions to synthesise
3. word halfword byte nibble 2-bit 1-bit reversal at multiple levels are all achieved with grevlut. Some of these instructions were explicitly added in Power ISA v3.1 but grevlut is akin to xxeval.
4. grevlut can be expensive in hardware (estimated 20,000 gates) but like xxeval provides 512 equivalent instructions.

Changes

Add the following entries to:

- the Appendices of Book I
- Book I 3.3.13 Fixed-Point Logical Instructions
- Book I 1.6.1 and 1.6.2

Rationale </>

bmask </>

Based on RVV masked set-before-first, set-after-first etc. and Intel and AMD Bitmanip instructions made generalised then advanced further to include masks, this is a single instruction covering 24 individual instructions in other ISAs.

The patterns within the pseudocode for AMD TBM and x86 BMI1 are as follows:

- first pattern A: two options `x` or `~x`
- second pattern B: three options `|` `&` or `^`
- third pattern C: four options `x+1`, `x-1`, `~(x+1)` or `(~x)+1`

Thus it makes sense to create a single instruction that covers all of these. A crucial addition that is essential for Scalable Vector usage as Predicate Masks, is the second mask parameter (RB). The additional paramater, L, if set, will leave bits of RA masked by RB unaltered, otherwise those bits are set to zero. Note that when `RB=0` then instead of reading from the register file the mask is set to all ones.

Executable pseudocode demo:

```
def bmask(bm, RA, RB=None, zero=False, XLEN=64):
    mask = RB if RB is not None else ((1<<XLEN)-1)
    ra = RA & mask
    mode1 = bm&1
    a1 = ra if mode1 else ~ra
    mode2 = (bm >> 1) & 0b11
    if mode2 == 0: a2 = -ra
    if mode2 == 1: a2 = ra-1
    if mode2 == 2: a2 = ra+1
    if mode2 == 3: a2 = ~(ra+1)
    a1 = a1 & mask
    a2 = a2 & mask
    mode3 = (bm >> 3) & 0b11
    if mode3 == 0: RS = a1 | a2
    if mode3 == 1: RS = a1 & a2
    if mode3 == 2: RS = a1 ^ a2
    if mode3 == 3: RS = 0 # RESERVED
    RS &= mask
    if not zero:
        # put back masked-out bits of RA
        RS |= RA & ~mask
    return RS

SBF = 0b01010 # set before first
SOF = 0b01001 # set only first
SIF = 0b10000 # set including first 10011 also works no idea why yet
```

[[!tag standards]]

SV Vector-assist Operations. </>

Links:

- [[discussion]]
- https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc#vector-register-gather-instructions
- https://lists.libre-soc.org/pipermail/libre-soc-dev/2022-May/004884.html
- https://bugs.libre-soc.org/show_bug.cgi?id=865 implementation in simulator
- https://bugs.libre-soc.org/show_bug.cgi?id=213
- https://bugs.libre-soc.org/show_bug.cgi?id=142 specialist vector ops out of scope for this document [[openpower/sv/3d_vector_ops]]
- [[simple_v_extension/specification/bitmanip]] previous version, contains pseudocode for sof, sif, sbf
- https://en.m.wikipedia.org/wiki/X86_Bit_manipulation_instruction_set#TBM_(Trailing_Bit_Manipulation)

The core Power ISA was designed as scalar: SV provides a level of abstraction to add variable-length element-independent parallelism. Therefore there are not that many cases where *actual* Vector instructions are needed. If they are, they are more "assistance" functions. Two traditional Vector instructions were initially considered (conflictd and vmiota) however they may be synthesised from existing SVP64 instructions: vmiota may use [[svstep]]. Details in [[discussion]]

Notes:

- Instructions suited to 3D GPU workloads (dotproduct, crossproduct, normalise) are out of scope: this document is for more general-purpose instructions that underpin and are critical to general-purpose Vector workloads (including GPU and VPU)
- Instructions related to the adaptation of CRs for use as predicate masks are covered separately, by crweird operations. See {CR Weird ops}.

Mask-suited Bitmanipulation </>

BM2-Form

0..5	6..10	11..15	16..20	21-25	26	27..31	Form
PO	RS	RA	RB	bm	L	XO	BM2-Form

- bmask RS,RA,RB,bm,L

Pseudo-code:

```
if _RB = 0 then mask <- [1] * XLEN
else            mask <- (RB)
ra <- (RA) & mask
a1 <- ra
if bm[4] = 0 then a1 <- ¬ra
mode2 <- bm[2:3]
if mode2 = 0 then a2 <- (¬ra)+1
if mode2 = 1 then a2 <- ra-1
if mode2 = 2 then a2 <- ra+1
if mode2 = 3 then a2 <- ¬(ra+1)
a1 <- a1 & mask
a2 <- a2 & mask
# select operator
mode3 <- bm[0:1]
if mode3 = 0 then result <- a1 | a2
if mode3 = 1 then result <- a1 & a2
if mode3 = 2 then result <- a1 ^ a2
if mode3 = 3 then result <- undefined([0]*XLEN)
# mask output
result <- result & mask
# optionally restore masked-out bits
if L = 1 then
    result <- result | (RA & ¬mask)
RT <- result
```

- first pattern A: two options `x` or `~x`
- second pattern B: three options `|` `&` or `^`
- third pattern C: four options `x+1`, `x-1`, `~(x+1)` or `(~x)+1`

The lower two bits of `bm` set to 0b11 are `RESERVED`. An illegal instruction trap must be raised.

Special Registers Altered:

```
None
```

Carry-lookahead </>

As a single scalar 32-bit instruction, up to 64 carry-propagation bits may be computed. When the output is then used as a Predicate mask it can be used to selectively perform the "add carry" of biginteger math, with `sv.addi/sm=rN RT.v, RA.v, 1`.

- cprop RT,RA,RB (Rc=0)
- cprop. RT,RA,RB (Rc=1)

pseudocode:

```
P = (RA)
G = (RB)
RT = ((P|G)+G)^P
```

X-Form

0:5	6:10	11:15	16:20	21:30	31	name	Form
PO	RT	RA	RB	XO	Rc	cprop	X-Form

used not just for carry lookahead, also a special type of predication mask operation.

Instruction Formats </>

Add the following entries to Book I 1.6.1 Word Instruction Formats:

MM-FORM </>

```
    |0    |6    |11   |16   |21   |24 |25   |31  |
    | PO  | FRT | FRA | FRB | FMM     | XO  | Rc |
    | PO  | RT  | RA  | RB  | MMM | / | XO  | Rc |
```

Add the following new fields to Book I 1.6.2 Word Instruction Fields:

```
    FMM (21:24)
        Field used to specify minimum/maximum mode for fminmax[s].

        Formats: MM

    MMM (21:23)
        Field used to specify minimum/maximum mode for integer minmax.

        Formats: MM
```

Add `MM` to the `Formats:` list for all of `FRT`, `FRA`, `FRB`, `XO (25:30)`, `Rc`, `RT`, `RA` and `RB`.

Appendices </>

```
Appendix E Power ISA sorted by opcode
Appendix F Power ISA sorted by version
Appendix G Power ISA sorted by Compliancy Subset
Appendix H Power ISA sorted by mnemonic
```

Form	Book	Page	Version	Mnemonic	Description
MM	I	#	3.2B	fminmax	Floating Minimum/Maximum
MM	I	#	3.2B	fminmaxs	Floating Minimum/Maximum Single
MM	I	#	3.2B	minmax	Minimum/Maximum

[[!tag opf_rfc]]