RFC ls004 Shift-And-Add </>

- 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/ls004/
- https://git.openpower.foundation/isa/PowerISA/issues/125
- feedback: https://bugs.libre-soc.org/show_bug.cgi?id=1091

Changes:

- initial shift-and-add https://bugs.libre-soc.org/show_bug.cgi?id=968
- add saddw: https://bugs.libre-soc.org/show_bug.cgi?id=996
- consider LD/ST-Shifted https://bugs.libre-soc.org/show_bug.cgi?id=1055

Severity: Major

Status: New

Date: 31 Oct 2022

Target: v3.2B

Source: v3.0B

Books and Section affected:

```
Book I Fixed-Point Shift Instructions 3.3.14.2
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
sadd - Shift and Add
saddw - Shift and Add Signed Word
sadduw - Shift and Add Unsigned Word
Also under consideration LD/ST-Indexed-Shifted
```

Submitter: Luke Leighton (Libre-SOC)

Requester: Libre-SOC

Impact on processor:

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

Impact on software:

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

Keywords:

```
GPR, Bit-manipulation, Shift, Arithmetic, Array Indexing
```

Motivation

Power ISA is missing LD/ST Indexed with shift, which is present in both ARM and x86. Adding more LD/ST is thirty eight instructions, a compromise is to add shift-and-add. Replaces a pair of explicit instructions in hot-loops.

Notes and Observations:

1. `sadd` and `sadduw` operate on unsigned integers.
2. `sadduw` is intended for performing address offsets, as the second operand is constrained to lower 32-bits and zero-extended.
3. All three are 2-in 1-out instructions.
4. shift-add operations are present in both x86 and aarch64, since they are useful for both general arithmetic and for computing addresses even when not immediately followed with a load/store.
5. `saddw` is often more useful than `sadduw` because C/C++ programmers like to use `int` for array indexing. for additional details see https://bugs.libre-soc.org/show_bug.cgi?id=996.
6. Even Motorola 68000 has LD/ST-Indexed-Shifted https://tack.sourceforge.net/olddocs/m68020.html#2.2.2.%20Extra%20MC68020%20addressing%20modes
7. should average-shift-add also be included? what about CA-in / CA-out?

Changes

Add the following entries to:

- the Appendices of Book I
- Instructions of Book I added to Section 3.3.14.2

Table of LD/ST-Indexed-Shift </>

The following demonstrates the alternative instructions that could be considered to be added. They are all 9-bit XO:

- 12 Load Indexed Shifted (with Update)
- 3 Load Indexed Shifted Byte-reverse
- 8 Store Indexed Shifted (with Update)
- 3 Store Indexed Shifted Byte-reverse
- 6 Floating-Point Load Indexed Shifted (with Update)
- 6 Floating-Point Store Indexed Shifted (with Update)
- 6 Load Indexed Shifted Update Post-Increment
- 4 Store Indexed Shifted Update Post-Increment
- 2 Floating-Point Load Indexed Shifted Update Post-Increment
- 2 Floating-Point Store Indexed Shifted Update Post-Increment

Total count: 51 new 9-bit XO instructions, for an approximate total XO cost of 3 bits within a single Primary Opcode. With the savings that these instructions represent in hot-loops, as evidenced by their inclusion in top-end ISAs such as x86 and ARM, the cost may be considered justifiable. However there is no point in placing the 38 Shifted-only group in EXT2xx, they need to be in EXT0xx, because if added as 64-bit Encoding the benefit reduction in binary size is not achieved. Post-Increment-Shifted on the other hand could reasonably be proposed in EXT2xx.

LD/ST-Shifted

0-5	6-10	11-15	16-20	21-22	23-31	Instruction
PO	RT	RA	RB	SH	XO	lbzsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhzsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhasx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwzsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwasx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	ldsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhbrsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwbrsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	ldbrsx RT,RA,RB,SH
PO	RS	RA	RB	SH	XO	stbsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	sthsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stwsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stdsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	sthbrsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stwbrsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stdbrsx RS,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfsxs FRT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfdxs FRT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfiwaxs FRT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfiwzxs FRT,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfsxs FRS,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfdxs FRS,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfiwxs FRS,RA,RB,SH

LD/ST-Shifted-Update

0-5	6-10	11-15	16-20	21-22	23-31	Instruction
PO	RT	RA	RB	SH	XO	lbzusx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhzusx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhausx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwzusx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwausx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	ldusx RT,RA,RB,SH
PO	RS	RA	RB	SH	XO	stbusx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	sthusx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stwusx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stdusx RS,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfsuxs FRT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfduxs FRT,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfsuxs FRS,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfduxs FRS,RA,RB,SH

Post-Increment-Update LD/ST-Shifted

0-5	6-10	11-15	16-20	21-22	23-31	Instruction
PO	RT	RA	RB	SH	XO	lbzuspx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhzuspx RT,RA,RB,SH

0-5	6-10	11-15	16-20	21-22	23-31	Instruction
PO	RT	RA	RB	SH	XO	lhauspx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwzuspx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwauspx RT,RA,RB,SH
PO	RS	RA	RB	SH	XO	stbuspx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	sthuspx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stwuspx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stduspx RS,RA,RB,SH
PO	RT	RA	RB	SH	XO	lduspx RT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfdupxs FRT,RA,RB,SH
PO	FRT	RA	RB	SH	XO	lfsupxs FRT,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfdupxs FRS,RA,RB,SH
PO	FRS	RA	RB	SH	XO	stfsupxs FRS,RA,RB,SH

Shift-and-Add </>

`sadd RT, RA, RB, SH`

0-5	6-10	11-15	16-20	21-22	23-30	31	Form
PO	RT	RA	RB	SH	XO	Rc	Z23-Form

Pseudocode:

```
shift <- SH + 1                     # Shift is between 1-4
sum[0:63] <- ((RB) << shift) + (RA) # Shift RB, add RA
RT <- sum                           # Result stored in RT
```

When `SH` is zero, the contents of register RB are multiplied by 2, added to the contents of register RA, and the result stored in RT.

`SH` is a 2-bit bit-field, and allows multiplication of RB by 2, 4, 8, 16.

Operands RA and RB, and the result RT are all 64-bit, unsigned integers.

NEED EXAMPLES (not sure how to embed SH)!!! Examples:

```
# adds r1 to (r2*8)
sadd r4, r1, r2, 3
```

Shift-and-Add Signed Word </>

`saddw RT, RA, RB, SH`

0-5	6-10	11-15	16-20	21-22	23-30	31	Form
PO	RT	RA	RB	SH	XO	Rc	Z23-Form

Pseudocode:

```
shift <- SH + 1                  # Shift is between 1-4
n <- EXTS64((RB)[32:63])         # Only use lower 32-bits of RB
sum[0:63] <- (n << shift) + (RA) # Shift n, add RA
RT <- sum                        # Result stored in RT
```

When `SH` is zero, the lower word contents of register RB are multiplied by 2, added to the contents of register RA, and the result stored in RT.

`SH` is a 2-bit bit-field, and allows multiplication of RB by 2, 4, 8, 16.

Operands RA and RB, and the result RT are all 64-bit, signed integers.

Programmer's Note: The advantage of this instruction is doing address offsets. RA is the base 64-bit address. RB is the offset into data structure limited to 32-bit.

Examples:

```
# r4 = r1 + (r2*16) <a name="ls004.mdwn_r4"> </>
saddw r4, r1, r2, 3
```

Shift-and-Add Unsigned Word </>

```
sadduw RT, RA, RB, SH
```

0-5	6-10	11-15	16-20	21-22	23-30	31	Form
PO	RT	RA	RB	SH	XO	Rc	Z23-Form

Pseudocode:

```
    shift <- SH + 1                 # Shift is between 1-4
    n <- (RB)[32:63]                # Only use lower 32-bits of RB
    sum[0:63] <- (n << shift) + (RA) # Shift n, add RA
    RT <- sum                       # Result stored in RT
```

When `SH` is zero, the lower word contents of register RB are multiplied by 2, added to the contents of register RA, and the result stored in RT.

`SH` is a 2-bit bit-field, and allows multiplication of RB by 2, 4, 8, 16.

Operands RA and RB, and the result RT are all 64-bit, unsigned integers.

Programmer's Note: The advantage of this instruction is doing address offsets. RA is the base 64-bit address. RB is the offset into data structure limited to 32-bit.

Examples:

```
# <a name="ls004.mdwn"> </>
sadduw r4, r1, r2, 2
```

Instruction Formats </>

Add the following to Book I 1.6.1

Z23-Form:

```
|  0-5  | 6-10 | 11-15 | 16-20 | 21-22 | 23-30 | 31 | Form     |
|-------|------|-------|-------|-------|-------|----|----------|
|  PO   | RT   |  RA   |  RB   |  SH   |  XO   | Rc | Z23-Form |
```

Instruction Fields </>

Add Z23 to the following Formats in Book I 1.6.2: `RT RA RB XO Rc`

Add the following new fields:

```
    SH (21:22)
        Field used to specify a shift amount.
        Formats: Z23
```

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
Z23	I	#	3.0B	sadd	Shift-and-Add
Z23	I	#	3.0B	saddw	Shift-and-Add Signed Word
Z23	I	#	3.0B	sadduw	Shift-and-Add Unsigned Word

[[!tag opf_rfc]]