RFC ls004 v2 Shift-And-Add and LD/ST-Shifted </>

- 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: 07 Feb 2024

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 LD/ST-Indexed-Shifted (Fixed and Floating)
```

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. Adding actual LD/ST Shifted saves even further.

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	lbzupsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhzupsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lhaupsx RT,RA,RB,SH

0-5	6-10	11-15	16-20	21-22	23-31	Instruction
PO	RT	RA	RB	SH	XO	lwzupsx RT,RA,RB,SH
PO	RT	RA	RB	SH	XO	lwaupsx RT,RA,RB,SH
PO	RS	RA	RB	SH	XO	stbupsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	sthupsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stwupsx RS,RA,RB,SH
PO	RS	RA	RB	SH	XO	stdupsx RS,RA,RB,SH
PO	RT	RA	RB	SH	XO	ldupsx 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
```

Load Byte and Zero Shifted Indexed </>

X-Form

- lbzsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- ([0] * (XLEN-8)) || MEM(EA, 1)
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The byte in storage addressed by EA is loaded into RT[56:63].
RT[0:55] are set to 0.
```

Special Registers Altered:

```
None
```

Load Byte and Zero Shifted with Update Indexed </>

X-Form

- lbzsux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- ([0] * (XLEN-8)) || MEM(EA, 1)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The byte in storage addressed by EA is loaded into RT[56:63].
RT[0:55] are set to 0.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Halfword and Zero Shifted Indexed </>

X-Form

- lhzsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- ([0] * (XLEN-16)) || MEM(EA, 2)
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are set to 0.
```

Special Registers Altered:

```
None
```

Load Halfword and Zero Shifted with Update Indexed </>

X-Form

- lhzsux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- ([0] * (XLEN-16)) || MEM(EA, 2)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are set to 0.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Halfword Algebraic Shifted Indexed </>

X-Form

- lhasx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- EXTS(MEM(EA, 2))
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are filled with a copy of bit 0 of the loaded halfword.
```

Special Registers Altered:

```
None
```

Load Halfword Algebraic Shifted with Update Indexed </>

X-Form

- lhasux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- EXTS(MEM(EA, 2))
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are filled with a copy of bit 0 of the loaded halfword.

EA is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Word and Zero Shifted Indexed </>

X-Form

- lwzsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- [0] * 32 || MEM(EA, 4)
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The word in storage addressed by EA is loaded into RT[32:63].
RT[0:31] are set to 0.
```

Special Registers Altered:

```
None
```

Load Word and Zero Shifted with Update Indexed </>

X-Form

- lwzsux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- [0] * 32 || MEM(EA, 4)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The word in storage addressed by EA is loaded into RT[32:63].
RT[0:31] are set to 0.

EA is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Word Algebraic Shifted Indexed </>

X-Form

- lwasx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- EXTS(MEM(EA, 4))
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The word in storage addressed by EA is loaded into RT[32:63].
RT[0:31] are filled with a copy of bit 0 of the loaded word.
```

Special Registers Altered:

```
None
```

Load Word Algebraic Shifted with Update Indexed </>

X-Form

- lwasux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- EXTS(MEM(EA, 4))
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The word in storage addressed by EA is loaded into RT[32:63].
RT[0:31] are filled with a copy of bit 0 of the loaded word.

EA is placed into register RA.
```

```
If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Doubleword Shifted Indexed </>

X-Form

- ldsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
RT <- MEM(EA, 8)
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

The doubleword in storage addressed by EA is loaded into RT.
```

Special Registers Altered:

```
None
```

Load Doubleword Shifted with Update Indexed </>

X-Form

- ldsux RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
RT <- MEM(EA, 8)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA).

The doubleword in storage addressed by EA is loaded into RT.

EA is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Halfword Byte-Reverse Shifted Indexed </>

X-Form

- lhbrsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
load_data <- MEM(EA, 2)
RT <- [0]*48 || load_data[8:15] || load_data[0:7]
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

Bits 0:7 of the halfword in storage addressed by EA are
loaded into RT[56:63]. Bits 8:15 of the halfword in storage
addressed by EA are loaded into RT[48:55].
RT[0:47] are set to 0.
```

Special Registers Altered:

```
None
```

Load Word Byte-Reverse Shifted Indexed </>

X-Form

- lwbrsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
load_data <- MEM(EA, 4)
RT <- ([0] * 32 || load_data[24:31] || load_data[16:23]
                || load_data[8:15]  || load_data[0:7])
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

Bits 0:7 of the word in storage addressed
by EA are loaded into RT[56:63]. Bits 8:15 of the word in
storage addressed by EA are loaded into RT[48:55]. Bits
16:23 of the word in storage addressed by EA are
loaded into RT[40:47]. Bits 24:31 of the word in storage
addressed by EA are loaded into RT 32:39.
RT[0:31] are set to 0.
```

Special Registers Altered:

```
None
```

Load Doubleword Byte-Reverse Shifted Indexed </>

X-Form

- ldbrsx RT,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
load_data <- MEM(EA, 8)
RT <- (load_data[56:63] || load_data[48:55]
    || load_data[40:47] || load_data[32:39]
    || load_data[24:31] || load_data[16:23]
    || load_data[8:15]  || load_data[0:7])
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

Bits 0:7 of the doubleword in storage addressed by EA
are loaded into RT[56:63]. Bits 8:15 of the doubleword in
storage addressed by EA are loaded into RT[48:55]. Bits
16:23 of the doubleword in storage addressed by EA
are loaded into RT[40:47]. Bits 24:31 of the doubleword in
storage addressed by EA are loaded into RT 32:39. Bits
32:39 of the doubleword in storage addressed by EA
are loaded into RT[24:31]. Bits 40:47 of the doubleword in
storage addressed by EA are loaded into RT[16:23]. Bits
48:55 of the doubleword in storage addressed by EA
are loaded into RT[8:15]. Bits 56:63 of the doubleword in
storage addressed by EA are loaded into RT[0:7].
```

Special Registers Altered:

```
None
```

Store Byte Shifted Indexed </>

X-Form

- stbsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 1) <- (RS)[XLEN-8:XLEN-1]
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

RS [56:63] are stored into the byte in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Byte Shifted with Update Indexed </>

X-Form

- stbsux RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
MEM(EA, 1) <- (RS)[XLEN-8:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA).

RS[56:63] are stored into the byte in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Halfword Shifted Indexed </>

X-Form

- sthsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 2) <- (RS)[XLEN-16:XLEN-1]
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

RS[48:63] are stored into the halfword in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Halfword Shifted with Update Indexed </>

X-Form

- sthsux RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
MEM(EA, 2) <- (RS)[XLEN-16:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA).

RS[48:63] are stored into the halfword in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Word Shifted Indexed </>

X-Form

- stwsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 4) <- (RS)[XLEN-32:XLEN-1]
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

RS[32:63] are stored into the word in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Word Shifted with Update Indexed </>

X-Form

- stwsux RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
MEM(EA, 4) <- (RS)[XLEN-32:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA).

RS[32:63] are stored into the word in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Doubleword Shifted Indexed </>

X-Form

- stdsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 8) <- (RS)
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

(RS) is stored into the doubleword in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Doubleword Shifted with Update Indexed </>

X-Form

- stdsux RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB) << (SH+1)
MEM(EA, 8) <- (RS)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register (RB) shifted by (SH+1), and (RA).

(RS) is stored into the doubleword in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Halfword Byte-Reverse Shifted Indexed </>

X-Form

- sthbrsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 2) <- (RS) [56:63] || (RS)[48:55]
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

(RS)56:63 are stored into bits 0:7 of the halfword in storage
addressed by EA.  (RS)[48:55] are stored into bits 8:15 of
the halfword in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Word Byte-Reverse Shifted Indexed </>

X-Form

- stwbrsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 4) <- ((RS)[56:63] || (RS)[48:55] || (RS)[40:47]
               ||(RS)[32:39])
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

(RS)[56:63] are stored into bits 0:7 of the
word in storage addressed by EA. (RS) [48:55] are stored
into bits 8:15 of the word in storage addressed by EA.
(RS)[40:47] are stored into bits 16:23 of the word in stor-
age addressed by EA. (RS) [32:39] are stored into bits
24:31 of the word in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Doubleword Byte-Reverse Shifted Indexed </>

X-Form

- stdbrsx RS,RA,RB,SH

Pseudo-code:

```
b <- (RA|0)
EA <- b + (RB) << (SH+1)
MEM(EA, 8) <- ((RS) [56:63] || (RS)[48:55]
                || (RS)[40:47] || (RS)[32:39]
                || (RS)[24:31] || (RS)[16:23]
                || (RS)[8:15]  || (RS)[0:7])
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and (RA|0).

(RS)[56:63] are stored into bits 0:7 of the
doubleword in storage addressed by EA. (RS)[48:55] are
stored into bits 8:15 of the doubleword in storage
addressed by EA. (RS) [40:47] are stored into bits 16:23 of
the doubleword in storage addressed by EA. (RS)[32:39]
are stored into bits 23:31 of the doubleword in storage
addressed by EA. (RS) [24:31] are stored into bits 32:39 of
the doubleword in storage addressed by EA. (RS)[16:23]
are stored into bits 40:47 of the doubleword in storage
addressed by EA. (RS)[8:15] are stored into bits 48:55 of
the doubleword in storage addressed by EA. (RS)[0:7]
are stored into bits 56:63 of the doubleword in storage
addressed by EA.
```

Special Registers Altered:

```
None
```

Load Floating-Point Single Indexed Shifted </>

X-Form

- lfssx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)<<(SH+1)
FRT <- DOUBLE(MEM(EA, 4))
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The word in storage addressed by EA is interpreted as
a floating-point single-precision operand. This word is
converted to floating-point double format (see
page 138) and placed into register FRT.
```

Special Registers Altered:

```
None
```

Load Floating-Point Single with Update Indexed Shifted </>

X-Form

- lfsusx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
FRT <- DOUBLE(MEM(EA, 4))
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The word in storage addressed by EA is interpreted as
a floating-point single-precision operand. This word is
converted to floating-point double format (see
page 138) and placed into register FRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Floating-Point Double Indexed Shifted </>

X-Form

- lfdsx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)<<(SH+1)
FRT <- MEM(EA, 8)
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The doubleword in storage addressed by EA is loaded
into register FRT.
```

Special Registers Altered:

```
None
```

Load Floating-Point Double with Update Indexed </>

X-Form

- lfdusx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
FRT <- MEM(EA, 8)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The doubleword in storage addressed by EA is loaded
into register FRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Floating-Point as Integer Word Algebraic Indexed Shifted </>

X-Form

- lfiwasx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)<<(SH+1)
FRT <- EXTS(MEM(EA, 4))
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The word in storage addressed by EA is loaded into
FRT [32:63]. FRT [0:31] are filled with a copy of bit 0 of the
loaded word.
```

Special Registers Altered:

```
None
```

Load Floating-Point as Integer Word Zero Indexed Shifted </>

X-Form

- lfiwzsx FRT,RA,RB

Pseudo-code:

```
EA <- (RA|0) + (RB)<<(SH+1)
FRT <- [0]*32 || MEM(EA, 4)
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The word in storage addressed by EA is loaded into
FRT [32:63]. FRT [0:31] are set to 0.
```

Special Registers Altered:

```
None
```

Store Floating-Point Single Indexed Shifted </>

X-Form

- stfssx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)>>(SH+1)
MEM(EA, 4)<- SINGLE( (FRS) )
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The contents of register FRS are converted to single
format (see page 142) and stored into the word in stor-
age addressed by EA.
```

Special Registers Altered:

```
None
```

Store Floating-Point Single with Update Indexed Shifted </>

X-Form

- stfsusx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)>>(SH+1)
MEM(EA, 4)<- SINGLE( (FRS) )
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The contents of register FRS are converted to single
format (see page 142) and stored into the word in stor-
age addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Floating-Point Double Indexed Shifted </>

X-Form

- stfdsx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)>>(SH+1)
MEM(EA, 8)<- (FRS)
```

Description:

```
Let the effective address (EA) be the sum (RA|0)+(RB).
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

The contents of register FRS are stored into the dou-
bleword in storage addressed by EA.
```

Special Registers Altered:

```
None
```

Store Floating-Point Double with Update Indexed Shifted </>

X-Form

- stfdusx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)>>(SH+1)
MEM(EA, 8)<- (FRS)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The contents of register FRS are stored into the dou-
bleword in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Floating-Point as Integer Word Indexed Shifted </>

X-Form

- stfiwsx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA|0) + (RB)>>(SH+1)
MEM(EA, 8)<- (FRS)[32:63]
```

Description:

```
Let the effective address (EA) be the sum of (RA|0) with
the contents of register RB shifted by (SH+1).

(FRS)[32:63] are stored, without conversion, into the word
in storage addressed by EA.

If the contents of register FRS were produced, either
directly or indirectly, by a Load Floating-Point Single
instruction, a single-precision Arithmetic instruction, or
frsp, then the value stored is undefined. (The contents
of register FRS are produced directly by such an
instruction if FRS is the target register for the instruc-
tion. The contents of register FRS are produced indi-
rectly by such an instruction if FRS is the final target
register of a sequence of one or more Floating-Point
Move instructions, with the input to the sequence hav-
ing been produced directly by such an instruction.)
```

Special Registers Altered:

```
None
```

Load Byte and Zero with Post-Update Indexed Shifted </>

X-Form

- lbzupsx RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- ([0] * (XLEN-8)) || MEM(EA, 1)
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The byte in storage addressed by EA is loaded into RT[56:63].
RT[0:55] are set to 0.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Halfword and Zero with Post-Update Indexed Shifted </>

X-Form

- lhzupsx RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- ([0] * (XLEN-16)) || MEM(EA, 2)
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are set to 0.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Halfword Algebraic with Post-Update Indexed Shifted </>

X-Form

- lhaupsx RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- EXTS(MEM(EA, 2))
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are filled with a copy of bit 0 of the loaded halfword.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Word and Zero with Post-Update Indexed Shifted </>

X-Form

* lwzupsx RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- [0] * 32 || MEM(EA, 4)
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The halfword in storage addressed by EA is loaded into RT[48:63].
RT[0:47] are filled with a copy of bit 0 of the loaded halfword.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Word Algebraic with Post-Update Indexed Shifted </>

X-Form

* lwaupsx RT,RA,RB.SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- EXTS(MEM(EA, 4))
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The word in storage addressed by EA is loaded into RT[32:63].
RT[0:31] are filled with a copy of bit 0 of the loaded word.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Doubleword with Post-Update Indexed Shifted </>

X-Form

* ldupsx RT,RA,RB,SH

Pseudo-code:

```
EA <- (RA)<<(SH+1)
RT <- MEM(EA, 8)
RA <- (RA) + (RB)
```

Description:

```
Let the effective address (EA) be the contents of
register RA shifted by (SH+1).

The doubleword in storage addressed by EA is loaded into RT.

The sum (RA) + (RB) is placed into register RA.

If RA=0 or RA=RT, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Byte with Post-Update Indexed Shifted </>

Z23-Form

- stbupsx RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
ea <- (RA)
MEM(ea, 1) <- (RS)[XLEN-8:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

(RS)[56:63] are stored into the byte in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Halfword with Post-Update Indexed Shifted </>

Z23-Form

- sthupsx RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
ea <- (RA)
MEM(ea, 2) <- (RS)[XLEN-16:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

(RS)[56:63] are stored into the byte in storage addressed by EA.

EA is placed into register RA.

If RA=0, the instruction form is invalid
```

Special Registers Altered:

```
None
```

Store Word with Post-Update Indexed Shifted </>

Z23-Form

- stwupsx RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
ea <- (RA)
MEM(ea, 4) <- (RS)[XLEN-32:XLEN-1]
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

(RS)[32:63] are stored into the word in storage addressed by RA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Doubleword with Post-Update Indexed Shifted </>

Z23-Form

- stdupsx RS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
ea <- (RA)
MEM(ea, 8) <- (RS)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

(RS) is stored into the doubleword in storage addressed by RA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Floating-Point Single with Post-Update Shifted Indexed </>

Z23-Form

- lfsupsx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
FRT <- DOUBLE(MEM(RA, 4))
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The word in storage addressed by EA is interpreted as
a floating-point single-precision operand. This word is
converted to floating-point double format (see
page 138) and placed into register FRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Load Floating-Point Double with Post-Update Shifted Indexed </>

Z23-Form

- lfdupsx FRT,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
FRT <- MEM(RA, 8)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The doubleword in storage addressed by EA is loaded
into register FRT.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Floating-Point Single with Update Shifted Indexed </>

X-Form

- stfsupsx FRS,RA,RB,SH

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
MEM(RA, 4)<- SINGLE( (FRS) )
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The contents of register FRS are converted to single
format (see page 142) and stored into the word in stor-
age addressed by RA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

Store Floating-Point Double with Update Shifted Indexed </>

X-Form

- stfdupsx FRS,RA,RB

Pseudo-code:

```
EA <- (RA) + (RB)<<(SH+1)
MEM(RA, 8)<- (FRS)
RA <- EA
```

Description:

```
Let the effective address (EA) be the sum of the contents of
register RB shifted by (SH+1), and the contents of register RA.

The contents of register FRS are stored into the dou-
bleword in storage addressed by RA.

EA is placed into register RA.

If RA=0, the instruction form is invalid.
```

Special Registers Altered:

```
None
```

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 |
|  PO    | RS   |  RA   |  RB   |  SH   |  XO   | Rc | Z23-Form |
|  PO    | FRT  |  RA   |  RB   |  SH   |  XO   | Rc | Z23-Form |
|  PO    | FRS  |  RA   |  RB   |  SH   |  XO   | Rc | Z23-Form |
```

Instruction Fields </>

Add Z23 to the following Formats in Book I 1.6.2: `FRS FRT 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]]