RFC ls011 LD/ST-Update-PostIncrement </>

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

- https://bugs.libre-soc.org/show_bug.cgi?id=1048
- https://libre-soc.org/openpower/sv/rfc/ls011/
- https://bugs.libre-soc.org/show_bug.cgi?id=1045
- https://git.openpower.foundation/isa/PowerISA/issues/TODO

Severity: Major

Status: New

Date: 21 Apr 2023.

Target: v3.2B

Source: v3.0B

Books and Section affected:

```
Chapter 2 Book I, new Fixed-Point Load / Store Sections 3.3.2 3.3.3
Chapter 4 Book I, new Floating-Point Load / Store Sections 4.6.2 4.6.3
```

Summary

```
TODO
```

Submitter: Luke Leighton (Libre-SOC)

Requester: Libre-SOC

Impact on processor:

```
Addition of new Load/Store Fixed and Floating Point instructions
```

Impact on software:

```
Requires support for new instructions in assembler, debuggers, and related tools.
Reduces instructions in hot-loops
```

Keywords:

Motivation

Moving the update of RA to *after* the Memory operation saves on instruction count both outside and inside hot-loops. strncpy may be reduced to 11 Vector instructions, 3 of which are the zeroing loop, 5 of which are the copy. Percentage-wise LD/ST Update Post-Increment represents a massive 20% reduction.

Notes and Observations:

These types of instructions are already present in x86 (sort-of).

- x86 chose that store should be pre-indexed and load should be post-indexed
- Power ISA chose everything to be pre-indexed
- Motorola 68000 (decades old) has pre- and post- indexed

https://tack.sourceforge.net/olddocs/m68020.html#2.2.2.%20Extra%20MC68020%20addressing%20modes

https://azeria-labs.com/memory-instructions-load-and-store-part-4/

Changes

Add the following entries to:

- New Load/Store Sections
- Appendices

[[!tag opf_rfc]]

TODO (key stub notes below)

The LD/ST-Immediate-Post-Increment instructions are all Primary Opcode: there are 13 of these. LD/ST-Indexed-Post-Increment are all effectively 9-bit XO and consequently may easily fit into one single Primary Opcode. EXT2xx is recommended.

One alternative idea is that bit 31 could be allocated (retrospectively) to Post-Increment. Although it may be too late for Scalar Power ISA it **may** be possible to consider for SVP64Single and/or SVP64-Vector, but this risks creating a non-Orthogonal ISA.

```
# LD/ST-Postincrement <a name="ls011.mdwn_ldstpostincrement"> </>
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 <a name="ls011.mdwn_fp_ldstpostincrement"> </>
lfdup,    ls011, high, PO, yes, EXT2xx, no, isa/pifixedload, 1R2W
lfsup,    ls011, high, PO, yes, EXT2xx, no, isa/pifixedload, 1R2W
lfdupx,   ls011, high, 10, yes, EXT2xx, no, isa/pifixedload, 2R2W
lsdupx,   ls011, high, 10, yes, EXT2xx, no, isa/pifixedload, 2R2W
stfdup,    ls011, high, PO, yes, EXT2xx, no, isa/pifixedstore, 2R1W
stfsup,    ls011, high, PO, yes, EXT2xx, no, isa/pifixedstore, 2R1W
stfdupx,   ls011, high, 10, yes, EXT2xx, no, isa/pifixedstore, 3R1W
stfsupx,   ls011, high, 10, yes, EXT2xx, no, isa/pifixedstore, 3R1W

# LD/ST-Shifted-Postincrement <a name="ls011.mdwn_ldstshiftedpostincrement"> </>
lbzuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lhzuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lhauspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lwzuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lwauspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lduspx,    ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
stbuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W
sthuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W
stwuspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W
stduspx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W

# FP LD/ST-Shifted-Postincrement <a name="ls011.mdwn_fp_ldstshiftedpostincrement"> </>
lfdupsx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
lfsupsx,   ls011, med, 10, yes, EXT2xx, no, ls011, 2R2W
stfdupsx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W
stfsupsx,   ls011, med, 10, yes, EXT2xx, no, ls011, 3R1W
```

Example </>

Here is an annotated example where the pseudo-code changes to just use `RA` as the address, otherwise remaining the same. No actual change to the Effective Address computation itself occurs, in any of the Post-Update instructions.

Load Byte and Zero with Post-Update

D-Form

- lbzup RT,D(RA)

Pseudo-code:

```
    EA <- (RA)                          # EA just RA
    RT <- ([0] * (XLEN-8)) || MEM(EA, 1) # then load
    RA <- (RA) + EXTS(D)                # then update RA after
```

Special Registers Altered:

```
    None
```

where the same pseudocode for `lbzu` is:

```
EA <- (RA) + EXTS(D)                  # EA includes D
RT <- ([0] * (XLEN-8)) || MEM(EA, 1)  # load from RA+D
RA <- EA                              # and update RA
```

Fixed-point Load with Post-Update </>

Add the following additional Section to Fixed-Point Load: Book I 3.3.2.1

Load Byte and Zero with Post-Update </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- lbzup RT,D(RA)

Pseudo-code:

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

Let the effective address (EA) be (RA|0). The byte in storage addressed by EA is loaded into RT[56:63]. RT[0:55] are set to 0.

The sum (RA|0)+D is placed into register RA.

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

Special Registers Altered:

None

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

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- lbzupx RT,RA,RB

Pseudo-code:

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

Let the effective address (EA) be (RA). 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 </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- lhzup RT,D(RA)

Pseudo-code:

```
EA <- (RA)
RT <- ([0] * (XLEN-16)) || MEM(EA, 2)
RA <- (RA) + EXTS(D)
```

Let the effective address (EA) be (RA|0). The halfword in storage addressed by EA is loaded into RT[48:63]. RT[0:47] are set to 0.

The sum (RA|0)+D 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 </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- lhzupx RT,RA,RB

Pseudo-code:

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

Let the effective address (EA) be (RA). 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 </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- lhaup RT,D(RA)

Pseudo-code:

```
EA <- (RA)
RT <- EXTS(MEM(EA, 2))
RA <- (RA) + EXTS(D)
```

Special Registers Altered:

None

Load Halfword Algebraic with Post-Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- lhaupx RT,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Load Word and Zero with Post-Update </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- lwzup RT,D(RA)

Pseudo-code:

```
EA <- (RA)
RT <- [0]*32 || MEM(EA, 4)
RA <- (RA) + EXTS(D)
```

Let the effective address (EA) be (RA|0). The word in storage addressed by EA is loaded into RT[32:63]. RT[0:31] are set to 0.

The sum (RA|0)+D 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 </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- lwzupx RT,RA,RB

Pseudo-code:

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

Let the effective address (EA) be (RA). The word in storage addressed by EA is loaded into RT[32:63]. RT[0:31] 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 Word Algebraic with Post-Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- lwaupx RT,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Load Doubleword with Post-Update Indexed </>

DS-Form

- ldup RT,DS(RA)

Pseudo-code:

```
EA <- (RA)
RT <- MEM(EA, 8)
RA <- (RA) + EXTS(DS || 0b00)
```

Special Registers Altered:

None

Load Doubleword with Post-Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RT      |    RA       |    RB       |   XO |  / |
```

- ldupx RT,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Fixed-Point Store Post-Update </>

Add the following as a new section in Fixed-Point Store, Book I

Store Byte with Update </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- stbup RS,D(RA)

Pseudo-code:

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

Special Registers Altered:

None

Store Byte with Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RS      |    RA       |    RB       |   XO |  / |
```

- stbupx RS,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Store Halfword with Update </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- sthup RS,D(RA)

Pseudo-code:

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

Special Registers Altered:

None

Store Halfword with Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RS      |    RA       |    RB       |   XO |  / |
```

- sthupx RS,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Store Word with Update </>

D-Form

```
|0     |6   |9  |10 |11   |16      |31 |
| PO   |    RT      |   RA|   D        |
```

- stwup RS,D(RA)

Pseudo-code:

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

Special Registers Altered:

None

Store Word with Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RS      |    RA       |    RB       |   XO |  / |
```

- stwupx RS,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

Store Doubleword with Update </>

DS-Form

- stdup RS,DS(RA)

Pseudo-code:

```
EA <- (RA) + EXTS(DS || 0b00)
ea <- (RA)
MEM(ea, 8) <- (RS)
RA <- EA
```

Special Registers Altered:

None

Store Doubleword with Update Indexed </>

X-Form

```
|0     |6 |7|8|9  |10  |11|12|13  |15|16|17     |20|21    |31  |
| PO   |       RS      |    RA       |    RB       |   XO |  / |
```

- stdupx RS,RA,RB

Pseudo-code:

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

Special Registers Altered:

None

[[!tag opf_rfc]]