RFC ls013 Min/Max GPR/FPR </>

- 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/ls013/
- https://git.openpower.foundation/isa/PowerISA/issues/TODO
- https://bugs.libre-soc.org/show_bug.cgi?id=1057

Severity: Major

Status: New

Date: 14 Apr 2023

Target: v3.2B

Source: v3.1B

Books and Section affected:

```
Book I Fixed-Point and Floating-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
```

Submitter: Luke Leighton (Libre-SOC)

Requester: Libre-SOC

Impact on processor:

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

Impact on software:

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

Keywords:

```
GPR, FPR, min, max, fmin, fmax
```

Motivation

Minimum/Maximum are common operations that can take an astounding number of operations to implement in software. Additionally, Vector Reduce-Min/Max are common vector operations, and SVP64 Parallel Reduction needs a single Scalar instruction in order to effectively implement Reduce-Min/Max.

Notes and Observations:

1. SVP64 REMAP Parallel Reduction needs a single Scalar instruction to work with, for best effectiveness. With no SFFS minimum/maximum instructions Simple-V min/max Parallel Reduction is severely compromised.
2. Once one FP min/max mode is implemented the rest are not much more hardware.
3. There exists similar instructions in VSX (not IEEE754-2019 though). This is frequently used to justify not adding them. However SVP64/VSX may have different meaning from SVP64/SFFS, so it is *really* crucial to have SFFS ops even if "equivalent" to VSX in order for SVP64 to not be compromised (non-orthogonal).
4. FP min/max are rather complex to implement in software, the most commonly used FP max function `fmax` from glibc compiled for SFFS is an astounding 32 instructions.

Changes

Add the following entries to:

- the Appendices of Book I
- Book I 3.3.9 Fixed-Point Arithmetic Instructions
- Book I 4.6.6.1 Floating-Point Elementary Arithmetic Instructions
- Book I 1.6.1 and 1.6.2

Floating-Point Instructions </>

This group is to provide Floating-Point min/max, however with the 2019 version of IEEE 754 there are now subtle differences. These are selectable with a Mode Field, `FMM`.

`FMM` – Floating Min/Max Mode </>

`FMM`	Extended Mnemonic	Origin	Semantics
0000	fminnum08[s] FRT,FRA,FRB	IEEE 754-2008	minNum(FRA,FRB) (1)
0001	fmin19[s] FRT,FRA,FRB	IEEE 754-2019	minimum(FRA,FRB)
0010	fminnum19[s] FRT,FRA,FRB	IEEE 754-2019	minimumNumber(FRA,FRB)
0011	fminc[s] FRT,FRA,FRB	x86 minss (4)	FRA<FRB ? FRA:FRB
0100	fminmagnum08[s] FRT,FRA,FRB	IEEE 754-2008 (3)	mmmag(FRA,FRB,False,fminnum08) (2)
0101	fminmag19[s] FRT,FRA,FRB	IEEE 754-2019	mmmag(FRA,FRB,False,fmin19) (2)
0110	fminmagnum19[s] FRT,FRA,FRB	IEEE 754-2019	mmmag(FRA,FRB,False,fminnum19) (2)
0111	fminmagc[s] FRT,FRA,FRB	-	mmmag(FRA,FRB,False,fminc) (2)
1000	fmaxnum08[s] FRT,FRA,FRB	IEEE 754-2008	maxNum(FRA,FRB) (1)
1001	fmax19[s] FRT,FRA,FRB	IEEE 754-2019	maximum(FRA,FRB)
1010	fmaxnum19[s] FRT,FRA,FRB	IEEE 754-2019	maximumNumber(FRA,FRB)
1011	fmaxc[s] FRT,FRA,FRB	x86 maxss (4)	FRA>FRB ? FRA:FRB
1100	fmaxmagnum08[s] FRT,FRA,FRB	IEEE 754-2008 (3)	mmmag(FRA,FRB,True,fmaxnum08) (2)
1101	fmaxmag19[s] FRT,FRA,FRB	IEEE 754-2019	mmmag(FRA,FRB,True,fmax19) (2)
1110	fmaxmagnum19[s] FRT,FRA,FRB	IEEE 754-2019	mmmag(FRA,FRB,True,fmaxnum19) (2)
1111	fmaxmagc[s] FRT,FRA,FRB	-	mmmag(FRA,FRB,True,fmaxc) (2)

Note (1): for the purposes of minNum/maxNum, -0.0 is defined to be less than +0.0. This is left unspecified in IEEE 754-2008.

Note (2): mmmag(x, y, cmp, fallback) is defined as:

```python
def mmmag(x, y, is_max, fallback):
    a = abs(x) < abs(y)
    b = abs(x) > abs(y)
    if is_max:
        a, b = b, a  # swap
    if a:
        return x
    if b:
        return y
    # equal magnitudes, or NaN input(s)
    return fallback(x, y)
```

Note (3): TODO: icr if IEEE 754-2008 has min/maxMagNum like IEEE 754-2019's minimum/maximumMagnitudeNumber

Note (4) or Win32's min macro

Floating Minimum/Maximum MM-form </>

- fminmax FRT, FRA, FRB, FMM
- fminmax. FRT, FRA, FRB, FMM

```
|0     |6     |11    |16    |21    |25   |31   |
| PO   | FRT  | FRA  | FRB  | FMM  | XO  | Rc  |
```

```
result <- [0] * 64
a <- (FRA)
b <- (FRB)
abs_a <- 0b0 || a[1:63]
abs_b <- 0b0 || b[1:63]
a_is_nan <- abs_a >u 0x7FF0_0000_0000_0000
a_is_snan <- a_is_nan & (a[12] = 0)
b_is_nan <- abs_b >u 0x7FF0_0000_0000_0000
b_is_snan <- b_is_nan & (b[12] = 0)
any_snan <- a_is_snan | b_is_snan
a_quieted <- a
a_quieted[12] <- 1
b_quieted <- b
b_quieted[12] <- 1
if a_is_nan | b_is_nan then
    if FMM[2:3] = 0b00 then  # min/maxnum08
        if a_is_snan then result <- a_quieted
        else if b_is_snan then result <- b_quieted
        else if a_is_nan & b_is_nan then result <- a_quieted
        else if a_is_nan then result <- b
        else result <- a
    if FMM[2:3] = 0b01 then  # min/max19
        if a_is_nan then result <- a_quieted
        else result <- b_quieted
    if FMM[2:3] = 0b10 then  # min/maxnum19
        if a_is_nan & b_is_nan then result <- a_quieted
        else if a_is_nan then result <- b
        else result <- a
    if FMM[2:3] = 0b11 then  # min/maxc
        result <- b
else
    cmp_l <- a
    cmp_r <- b
    if FMM[1] then  # min/maxmag
        if abs_a != abs_b then
            cmp_l <- abs_a
            cmp_r <- abs_b
    if FMM[2:3] = 0b11 then  # min/maxc
        if abs_a = 0 then cmp_l[0:63] <- 0
        if abs_b = 0 then cmp_r[0:63] <- 0
    if FMM[0] then  # max
        # swap cmp_* so comparison goes the other way
        cmp_l, cmp_r <- cmp_r, cmp_l
    if cmp_l[0] = 1 then
        if cmp_r[0] = 0 then result <- a
        else if cmp_l >u cmp_r then
            # IEEE 754 is sign-magnitude,
            # so bigger magnitude negative is smaller
            result <- a
        else result <- b
    else if cmp_r[0] = 1 then result <- b
    else if cmp_l <u cmp_r then result <- a
    else result <- b
if any_snan then SetFX(FPSCR.VXSNAN)
if (FPSCR.VE = 0) | ¬any_snan then (FRT) <- result
```

Compute the minimum/maximum of FRA and FRB, according to FMM, and store the result in FRT.

Special Registers altered:

```
FX VXSNAN
CR1     (if Rc=1)
```

Extended Mnemonics:

see `FMM` – Floating Min/Max Mode

Fixed-Point Instructions </>

These are signed and unsigned, min or max. SVP64 Prefixing defines Saturation semantics therefore Saturated variants of these instructions need not be proposed.

`MMM` – Integer Min/Max Mode </>

- bit 0: set if word variant else dword
- bit 1: set if signed else unsigned
- bit 2: set if max else min

`MMM`	Extended Mnemonic	Semantics
000	`minu RT,RA,RB`	`(uint64_t)RA < (uint64_t)RB ? RA : RB`
001	`maxu RT,RA,RB`	`(uint64_t)RA > (uint64_t)RB ? RA : RB`
010	`mins RT,RA,RB`	`(int64_t)RA < (int64_t)RB  ? RA : RB`
011	`maxs RT,RA,RB`	`(int64_t)RA > (int64_t)RB  ? RA : RB`
100	`minuw RT,RA,RB`	`(uint32_t)RA < (uint32_t)RB ? RA : RB`
101	`maxuw RT,RA,RB`	`(uint32_t)RA > (uint32_t)RB ? RA : RB`
110	`minsw RT,RA,RB`	`(int32_t)RA < (int32_t)RB  ? RA : RB`
111	`maxsw RT,RA,RB`	`(int32_t)RA > (int32_t)RB  ? RA : RB`

Minimum/Maximum MM-Form </>

- minmax RT, RA, RB, MMM
- minmax. RT, RA, RB, MMM

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

```
a <- (RA|0)
b <- (RB)
if MMM[0] then  # word mode
    # shift left by XLEN/2 to make the dword comparison
    # do word comparison of the original inputs
    a <- a[XLEN/2:XLEN-1] || [0] * XLEN/2
    b <- b[XLEN/2:XLEN-1] || [0] * XLEN/2
if MMM[1] then  # signed mode
    # invert sign bits to make the unsigned comparison
    # do signed comparison of the original inputs
    a[0] <- ¬a[0]
    b[0] <- ¬b[0]
# if Rc = 1 then store the result of comparing a and b to CR0
if Rc = 1 then
    if a <u b then
        CR0 <- 0b100 || XER.SO
    if a = b then
        CR0 <- 0b001 || XER.SO
    if a >u b then
        CR0 <- 0b010 || XER.SO
if MMM[2] then  # max mode
    # swap a and b to make the less than comparison do
    # greater than comparison of the original inputs
    t <- a
    a <- b
    b <- t
# store the entire selected source (even in word mode)
# if Rc = 1 then store the result of comparing a and b to CR0
if a <u b then RT <- (RA|0)
else RT <- (RB)
```

Compute the integer minimum/maximum according to `MMM` of `(RA|0)` and `(RB)` and store the result in `RT`.

Special Registers altered:

```
CR0     (if Rc=1)
```

Extended Mnemonics:

see `MMM` – Integer Min/Max Mode

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.

    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	minmax	Minimum/Maximum

fmax instruction count </>

32 instructions are required in SFFS to emulate fmax.

```
#include <stdint.h>
#include <string.h>

inline uint64_t asuint64(double f) {
    union {
        double f;
        uint64_t i;
    } u = {f};
    return u.i;
}

inline int issignaling(double v) {
    // copied from glibc:
    // https://github.com/bminor/glibc/blob/e2756903/sysdeps/ieee754/dbl-64/math_config.h#L101
    uint64_t ix = asuint64(v);
    return 2 * (ix ^ 0x0008000000000000) > 2 * 0x7ff8000000000000ULL;
}

double fmax(double x, double y) {
    // copied from glibc:
    // https://github.com/bminor/glibc/blob/e2756903/math/s_fmax_template.c
    if(__builtin_isgreaterequal(x, y))
        return x;
    else if(__builtin_isless(x, y))
        return y;
    else if(issignaling(x) || issignaling(y))
        return x + y;
    else
        return __builtin_isnan(y) ? x : y;
}
```

Assembly listing:

```
    fmax(double, double):
        fcmpu 0,1,2
        fmr 0,1
        cror 30,1,2
        beq 7,.L12
        blt 0,.L13
        stfd 1,-16(1)
        lis 9,0x8
        li 8,-1
        sldi 9,9,32
        rldicr 8,8,0,11
        ori 2,2,0
        ld 10,-16(1)
        xor 10,10,9
        sldi 10,10,1
        cmpld 0,10,8
        bgt 0,.L5
        stfd 2,-16(1)
        ori 2,2,0
        ld 10,-16(1)
        xor 9,10,9
        sldi 9,9,1
        cmpld 0,9,8
        ble 0,.L6
.L5:
        fadd 1,0,2
        blr
.L13:
        fmr 1,2
        blr
.L6:
        fcmpu 0,2,2
        fmr 1,2
        bnulr 0
.L12:
        fmr 1,0
        blr
        .long 0
        .byte 0,9,0,0,0,0,0,0
```

[[!tag opf_rfc]]