Report 30.09.20

30/09/2020

Matteo Perotti
Luca Bertaccini
Pasquale Davide Schiavone
Stefan Mach

Professor Luca Benini
Integrated Systems Laboratory
ETH Zürich

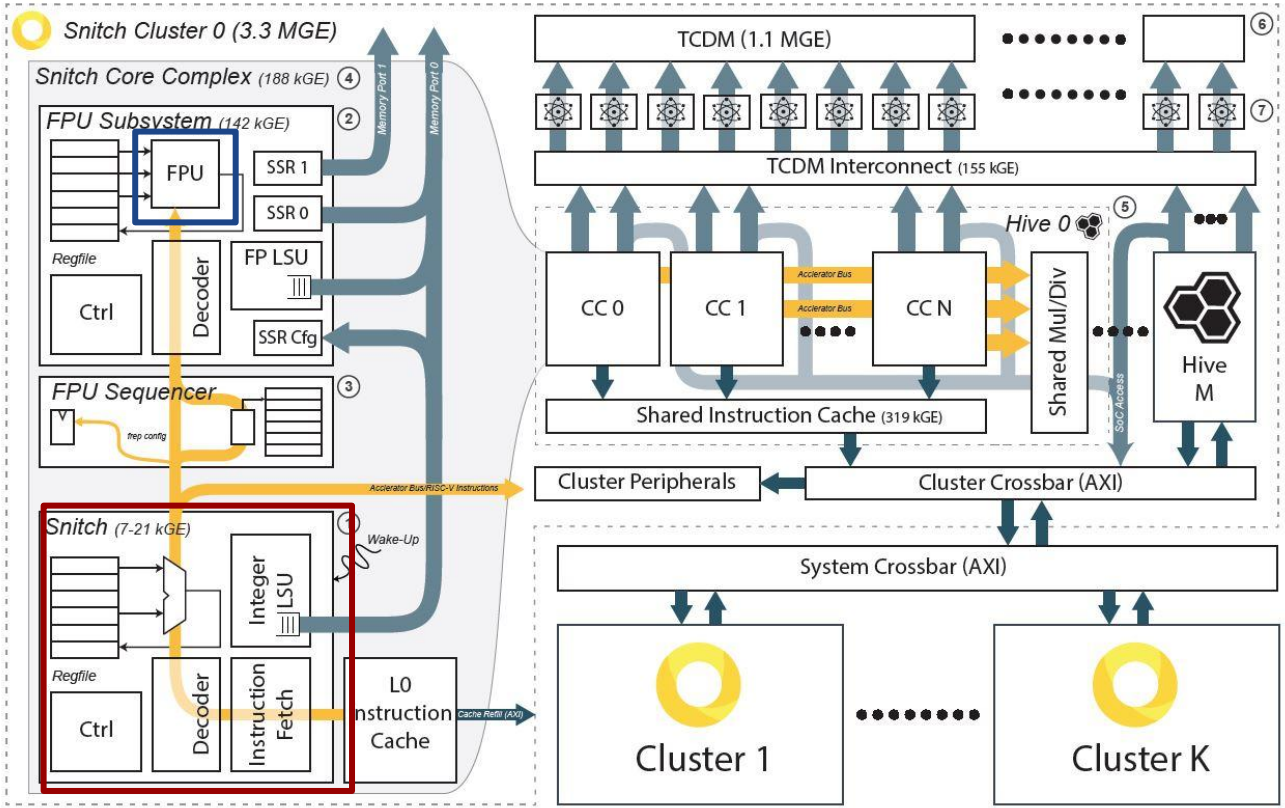
Tiny Floating-Point Unit

- Microcode
- Paper
- Merged implementation

SNITCH

Tiny FPU

Snitch



Zaruba, F., Schuiki, F., Hoefler, T., & Benini, L. (2020). Snitch: A 10 kGE Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads. arXiv preprint cs.AR/2002.10143.

Microcode

- Compile with **F/D extensions** enabled (e.g. fadd.s f1,f2,f3)
- Modify the decoder to forward the instructions to the Microcode module
- FSM generates the ALU operations needed to compute FP instructions on an INT datapath and forwards the inputs to the INT datapath
- The FSM has internal registers (“shadow register”) to store the operands and the partial results

Microcode

Microcoded instructions:

- Comparison: FEQ.S, FLT.S, FLE.S
- FADD.S, FSUB.S
- FMUL.S

Microcode

Shadow registers:

- Three 32-bit registers: op_a/op_b/partial_result (rounding)
- FSM state
- Tag: 5 bits
- Flags to handle denormals: 3 bits
- 4 bits more for FADD.S

Microcode

	MICROCODE	SEGGER SW EMULATION (avg)	TINY FPU
FEQ_S	**2 cycles**	**10 cycles**	**2 cycles**
FLT_S	**5 cycles (max)**	**11 cycles**	**2 cycles**
FLE_S	**5 cycles (max)**	**10 cycles**	**2 cycles**
FMUL_S	**18 cycles**	**39.3 cycles**	**18 cycles**
FADD_S	**~30 cycles**	**49.5 cycles**	**10-13 cycles**
FSUB_S	**FADD_S latency + 1 cycle**	**62.2 cycles**	**10-13 cycles**

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

Microcode

- 20x20 FP32 **MATMUL** (MUL + ADD)

matmul cycles	**FPU**	**Tiny FPU**	**uCoded**	**Opt. SW**	**libgcc**
f32, N20, LU4	68465	234250	367131	1029084	1995997
f64, N20, LU4	70853	355341	/	1819240	3209205

Microcode

TODO:

- FMADD_S
- area results
- area optimization (FP32)
- Microcode for FP64 instructions

Paper

Comparison:

1. **Snitch + FPU**
2. **Snitch + TinyFPU**
3. **Snitch (INT-only):**
    a. Optimized SW library
    b. libgcc

Contribution:

1. TinyFPU
2. Optimized SW library

Paper content

- Main focus: **Energy efficiency**
- Benchmark 1: 20x20 MATMUL + non-FP computation time
- Benchmark 2: application that rarely uses the FPU
- Energy vs inter-event time

Paper issues

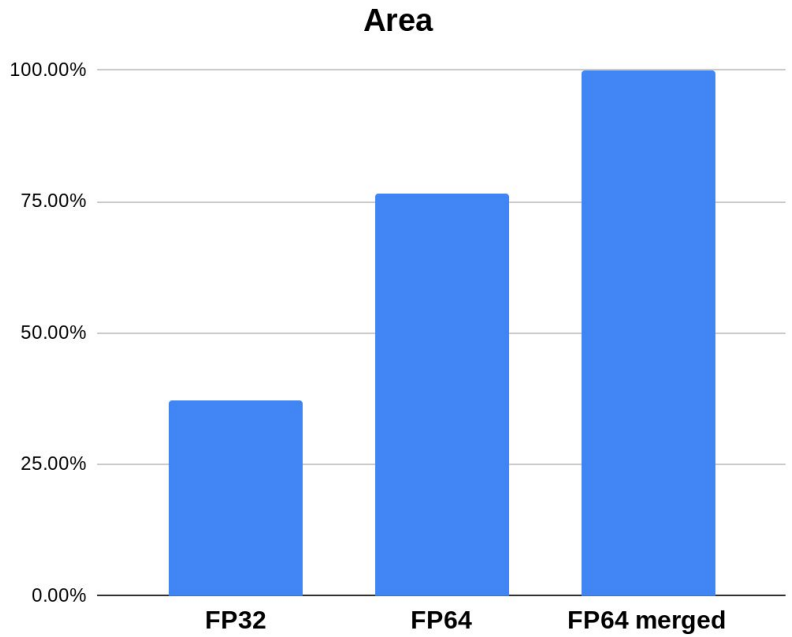
- **Merged implementation** (TinyFPU): to compute FP32 instructions on FP64 datapath
- **Power** results

TinyFPU - Merged Implementation

- D-extension requires support for single-precision FP
- Added support for FP32 instructions to the FP64 TinyFPU
- **FP32 instructions** computed on the **FP64 datapath**
- Same **latencies** as for the FP32-only TinyFPU

TinyFPU - Merged Implementation (Area Overhead)

- Target: RVFD
- Merged implementation instead of FP32 TinyFPU and FP64 TinyFPU



Next steps

- Simplify Snitch cluster
- Area and power analysis for the various implementations (Snitch+FastFPU, Snitch+TinyFPU, Snitch)

Optimized FP library

- Three flavours:
  - RV32**I**M **+ Denormals**
  - RV32**I**M No Denormals
  - RV32**E**M No Denormals

No Denormals

There is **no specification** about **no**n-supporting **denormals**.
Choices to be faster if we do not require high precision:

- **Opportunistic denormal** handling: some denormals are treated, others are flushed to zero. If they are treated, they are treated correctly.
- Multiplication: if the result would be “normal” only after the rounding, it is flushed to zero.
- Comparison functions: INFs are treated as NaNs.

Optimized FP library

- Three flavours:
  - RV32**I**M **+ Denormals** → **Verified with TestFloat (level 2, max number of inputs)**
  - RV32**I**M No Denormals
  - RV32**E**M No Denormals

} → **Verified with TestFloat (level 1, less inputs)**

Code size (RV32[I/E]MC)

Code Size (B)	Mine	Mine_nd_e	libgcc	SEGGER_nd	ARM
f32_add	414	392	786	410	352
f32_sub	8	8	816	10	4
f32_mul	374	172	538	178	360
f32_eq	38	38	86	44	8
f32_le	52	52	120	54	8
f32_lt	56	56	120	58	94
f64_add	750	656	1520	724	632
f64_sub	10	10	1538	10	4
f64_mul	560	288	1096	286	596
f64_eq	64	46	106	52	8
f64_le	104	84	166	70	8
f64_lt	110	88	166	70	122
Total	2540	1890	7058	1966	2196

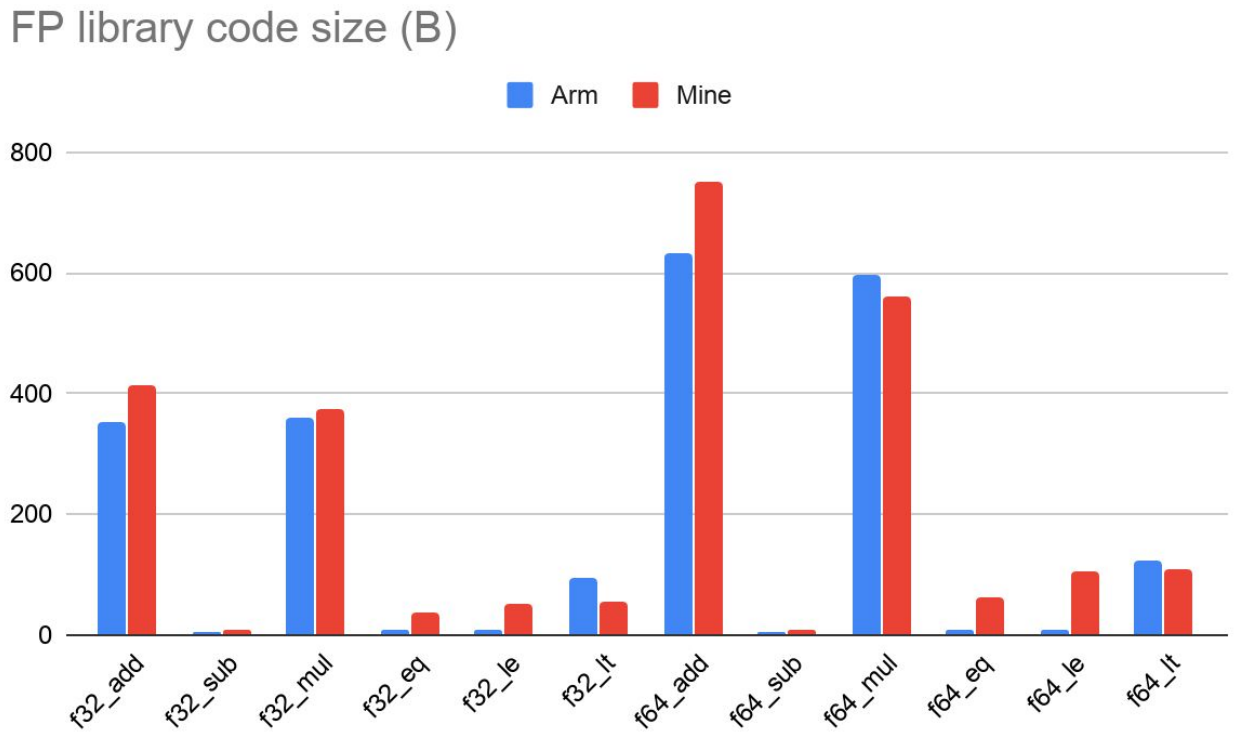
SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

Code size (RV32[I/E]MC)

Code Size (B)	Mine	Mine_nd_e	libgcc	SEGGER_nd	ARM
f32_add	414	392	786	410	352
f32_sub	8	8	816	10	4
f32_mul	374	172	538	178	360
f32_eq	38	38	86	44	8*
f32_le	52	52	120	54	8*
f32_lt	56	56	120	58	94*
f64_add	750	656	1520	724	632
f64_sub	10	10	1538	10	4
f64_mul	560	288	1096	286	596
f64_eq	64	46	106	52	8*
f64_le	104	84	166	70	8*
f64_lt	110	88	166	70	122*
Total	**2540 (+15.7%)**	1890	7058	1966	**2196**

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

libgcc Arm vs. Mine

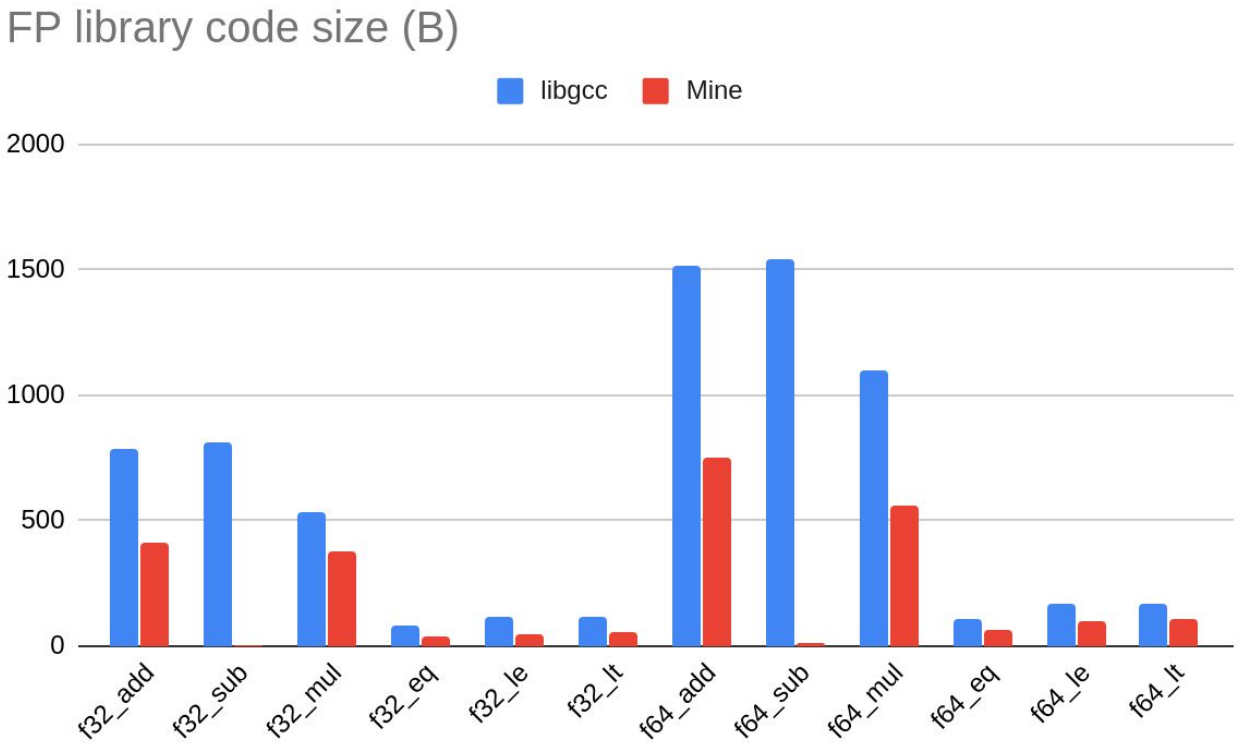


Code size (RV32[I/E]MC)

Code Size (B)	Mine	Mine_nd_e	libgcc	SEGGER_nd	ARM
f32_add	414	392	786	410	352
f32_sub	8	8	816	10	4
f32_mul	374	172	538	178	360
f32_eq	38	38	86	44	8
f32_le	52	52	120	54	8
f32_lt	56	56	120	58	94
f64_add	750	656	1520	724	632
f64_sub	10	10	1538	10	4
f64_mul	560	288	1096	286	596
f64_eq	64	46	106	52	8
f64_le	104	84	166	70	8
f64_lt	110	88	166	70	122
Total	**2540 (-64%)**	1890	**7058**	1966	2196

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

libgcc RISC-V vs. Mine

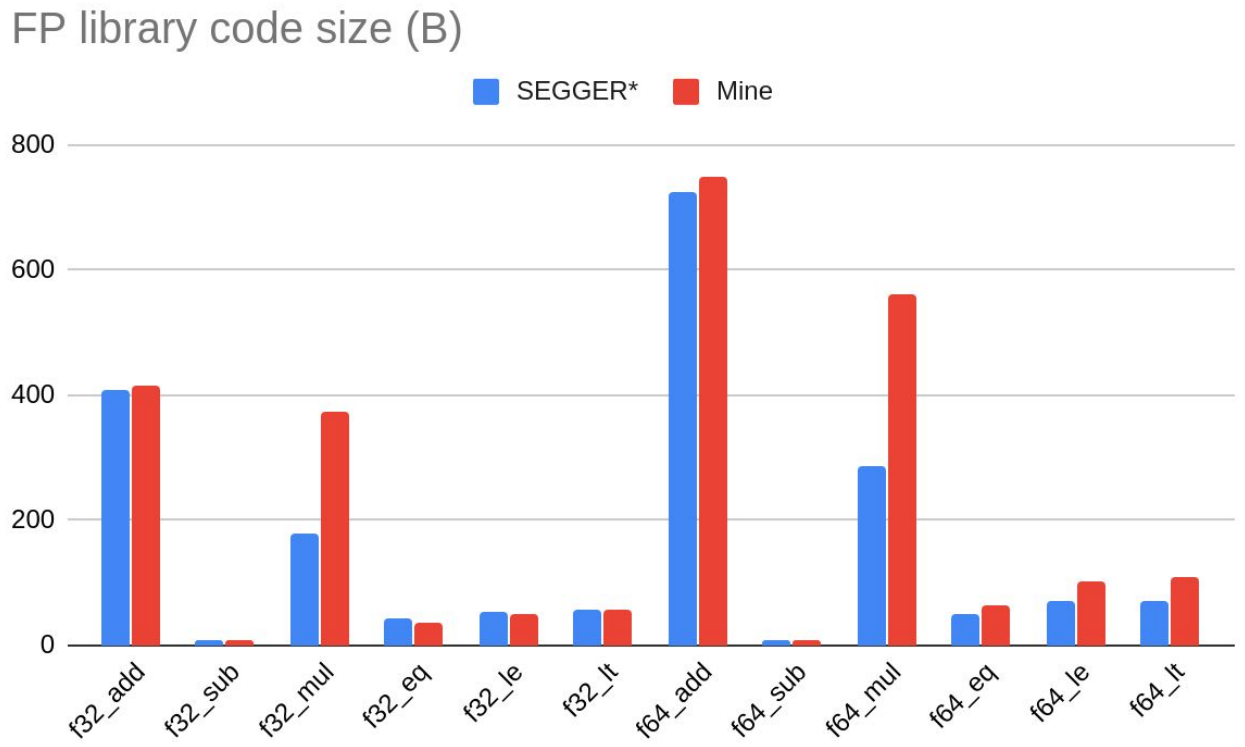


Code size (RV32[I/E]MC)

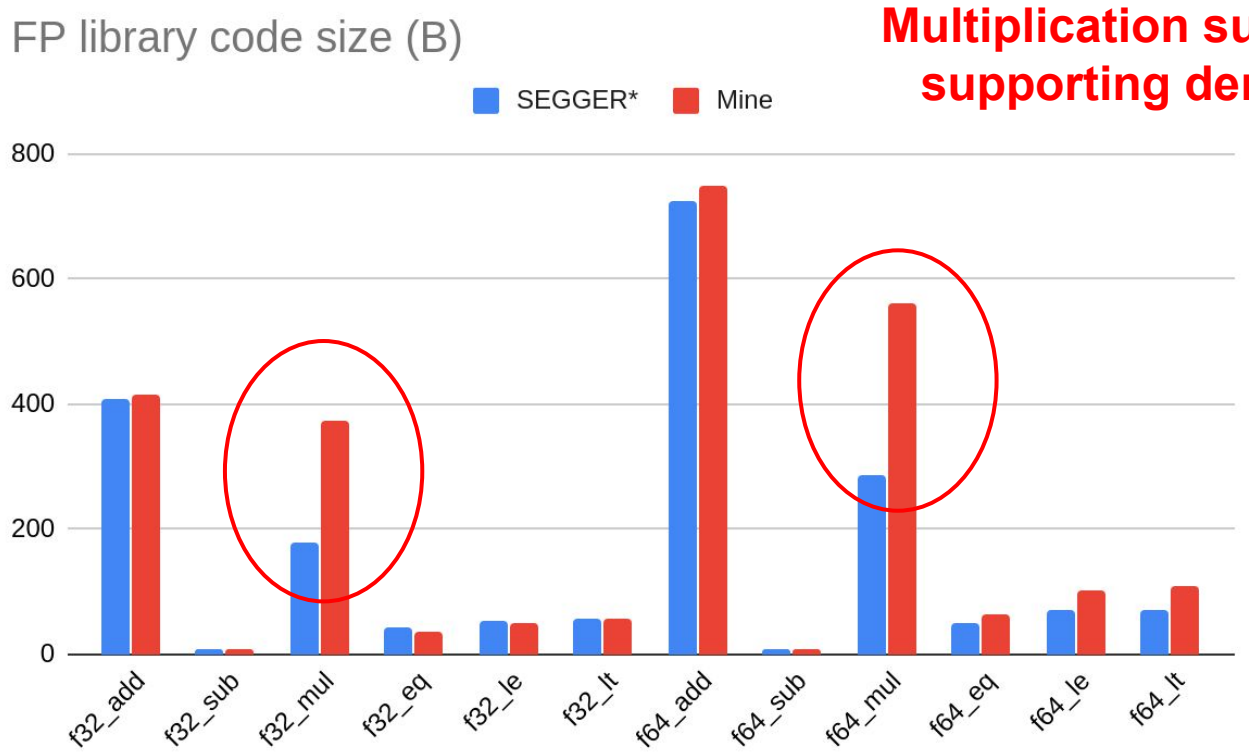
Code Size (B)	Mine	Mine_nd_e	libgcc	SEGGER_nd	ARM
f32_add	414	392	786	410	352
f32_sub	8	8	816	10	4
f32_mul	374	172	538	178	360
f32_eq	38	38	86	44	8
f32_le	52	52	120	54	8
f32_lt	56	56	120	58	94
f64_add	750	656	1520	724	632
f64_sub	10	10	1538	10	4
f64_mul	560	288	1096	286	596
f64_eq	64	46	106	52	8
f64_le	104	84	166	70	8
f64_lt	110	88	166	70	122
Total	**2540 (+29%)**	1890	7058	**1966**	2196

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

SEGGER (No Denormal) vs. Mine



SEGGER (No Denormal) vs. Mine



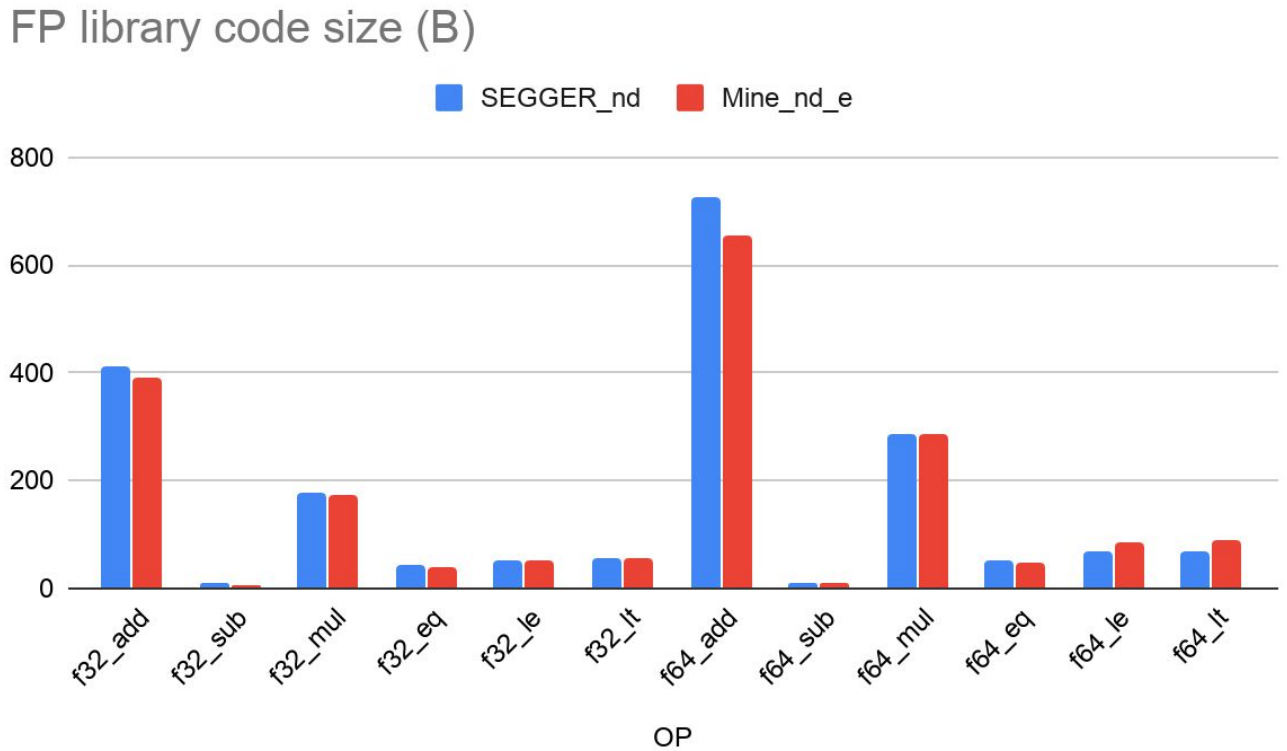
Multiplication suffers from supporting denormals!

Code size (RV32[I/E]MC)

Code Size (B)	Mine	Mine_nd_e	libgcc	SEGGER_nd	ARM
f32_add	414	392	786	410	352
f32_sub	8	8	816	10	4
f32_mul	374	172	538	178	360
f32_eq	38	38	86	44	8
f32_le	52	52	120	54	8
f32_lt	56	56	120	58	94
f64_add	750	656	1520	724	632
f64_sub	10	10	1538	10	4
f64_mul	560	288	1096	286	596
f64_eq	64	46	106	52	8
f64_le	104	84	166	70	8
f64_lt	110	88	166	70	122
Total	2540	**1890 (-4%)**	7058	**1966**	2196

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

SEGGER (No Denormal) vs. Mine (No Denormal RV32EMC)



Performance

Experiment setup:

- **Random in**puts within **(-1, 1)**
- **CV32E40P** core with **1-cycle latency SRAMs**
- Measured **avg. cycle count** for each function
- HW counter **mcycle**

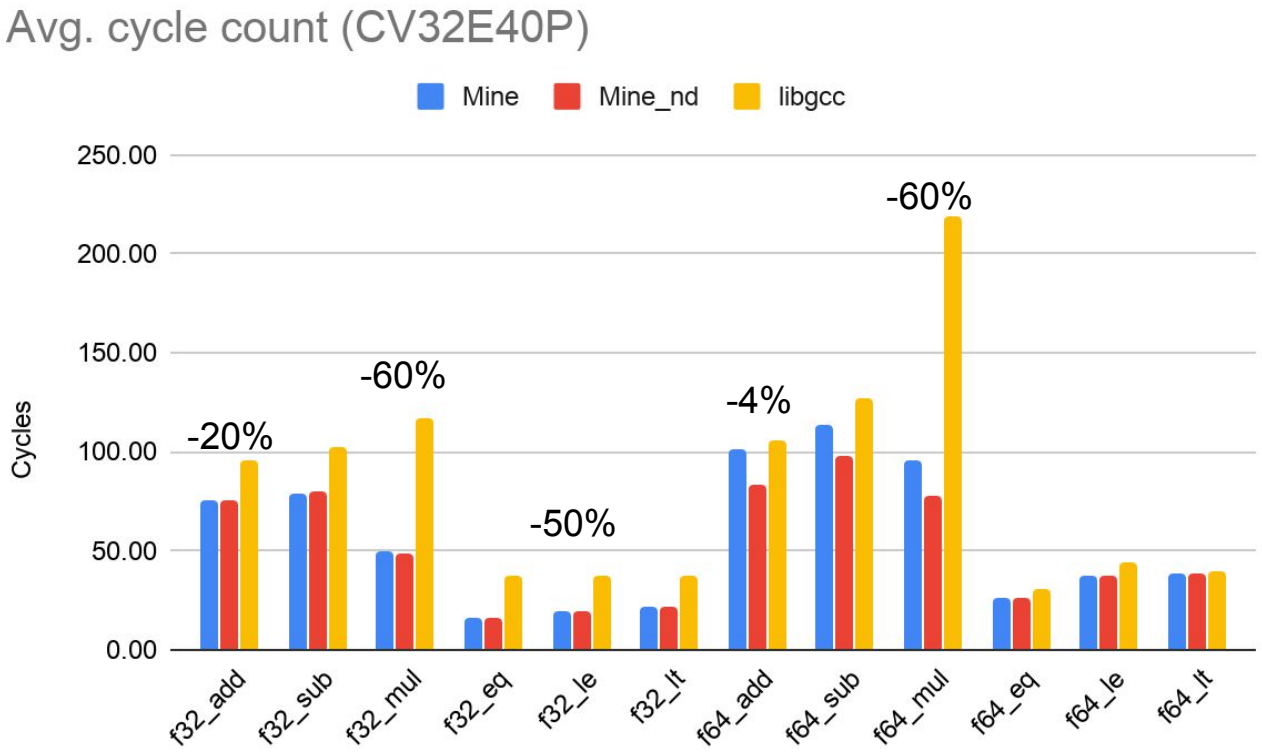
Performance (preliminary)

OP	Mine	Mine_nd_e	libgcc	SEGGER_nd
f32_add	75.10	75.10	95.30	49.5**
f32_sub	79.20	80.20	103.00	62.2**
f32_mul	50.20	48.20	117.20	39.3**
f32_eq	16.00	16.00	37.80	10**
f32_le	20.00	20.00	37.80	10**
f32_lt	22.00	22.00	37.80	11**
f64_add	101.30	83.60	106.00	62.8**
f64_sub	114.10	97.50	126.70	82.8**
f64_mul	96.10	78.10	218.50	75.0**
f64_eq	26.00	26.00	31.10	19**
f64_le	37.00	37.00	44.20	19**
f64_lt	39.00	39.00	40.00	20**

SEGGER data taken from: https://www.segger.com/products/development-tools/runtime-library/technology/floating-point-library/

** Different experiment setup, unfair comparison

Performance (preliminary)



Further

- Automatic verification also for no-denormal implementations
- Find a benchmark that uses FP “rarely enough”
- Test performance for no-denormal rv32e implementations