Report 16.12.20

16/12/2020

Matteo Perotti
Luca Bertaccini
Pasquale Davide Schiavone
Stefan Mach

Professor Luca Benini
Integrated Systems Laboratory
ETH Zürich

New instructions in RVfplib - Quantitative Analysis

- **New** RISC-V **instructions** from the **Code Size** reduction TG
- **Evaluate** their impact **on RVfplib**
- Manual inspection, since RVfplib is hard-coded in assembly

Source: https://github.com/riscv/riscv-code-size-reduction/tree/master/ISA%20proposals/Huawei

Instructions

We **exclude** the **memory operations**, as RVfplib does not use them.

- **32-bit**:
  - beqi, blti[u], bgei[u]
- **16-bit**:
  - c.mul (rs1 == rd)
  - c.lsbnot
  - c.neg, c.not
  - c.sext, c.zext
- **Others**:
  - shift + op
  - op + carry
  - clz

Source: https://github.com/riscv/riscv-code-size-reduction/tree/master/ISA%20proposals/Huawei

Functions

Representative subset RVfplib:

- **Arithmetic**
  - Add 32-64
  - Mul 32-64
  - Div 32-64
- **Conversion**
  - Double -> long long int
  - Long long int -> double
  - Double -> Float
  - Float -> Double
- **Comparison**
  - Less or equal 32-64

Methodology

- **Branch immediate**
  - Look for **li (2B) + branch (4B)**
- **c.mul**
  - Look for **mul** with **rd == rs1**
- **c.lsbnot**
  - Look for **A ^= 1 (xori a, a, 1)**
- **c.neg, c.not**
  - Look for **neg, not**
- **c.sext, c.zext**
  - Think about situations in which they can be useful
- **clz**
  - Look for explicit **CLZ sequences**

Methodology

- **Shift + Op**
  - Look for **shift immediate + generic ALU operation**
  - The sequence should be > 4 Byte, i.e. at least one between the shift immediate and the operation is not compressed

- **Op + Carry**
  - Look for **Add, Subtract, Shift** for which a **“carry”** can help

Code Size savings

Saving (B)	branch imm	c.neg*	c.not*	c.mul	clz	c.lsbnot	c.sext/c.zext	shift+op	op+carry
add32	2	4	4	0	0*	0	0	2	0
add64	4	12	10	0	~50**	0	0	14	34
mul32	0	0	0	0	0	0	0	4	2
mul64	0	2	4	0	0	0	0	26	14
div32	0	2	2	0	0	0	0	12	0
div64	2	2	0	0	0	0	0	34	12
lesf2	0	0	0	0	0	0	0	0	0
ledf2	0	0	0	0	0	0	0	4	0
fixdfdi	0	8	6	0	0	0	0	2	0
floatdidf	0	4	4	0	0	0	0	2	2
extend	0	0	0	0	0*	0	0	0	0
trunc	0	2	0	0	0	0	0	6	0

*No saving for the functions optimized for low code size. But great performance boost for severe numerical cancellations

**Up to now, 64-bit add does not have a slow reduced clz implementation

***Considering to compress ALL the neg, not, regardless of the RVC register subset limitation

c.neg, c.not

- The **proposed encodings** exploit only the **8 RVC registers**
- The **table** shows the **maximum possible savings**, hypothesizing that the instructions are compatible with all the registers, or at least with the first 16
- If this is not the case, the register allocation in RVfplib can be changed keeping into account these new instructions

Shift+Op

- **Expensive** instructions in terms of **encoding**
- Generic, it would be **not limited by** the **RVC** register **subset**, speeding up the computation (with a 1 cycle operation)
- **Not always the best solution**. When an operand is shifte and then used to perform multiple operations on different registers, split shift+op instructions provide the lower code size

Op+Carry

- **Add, Sub, Shift-by-one**
- Just **one instruction per type**, as updating the carry bit can be a default side effect of the normal RISC-V ALU operations
- Operations:
  - **Add** with **carry**, if present
  - **Subtract** with **borrow**, if present
  - **Shift** grabbing the previously-**shifted-out bit**

Relative code size saving

- **Estimate** of the **saving** for the function subset
- **Manual inspection**
- Some **savings** are **overlapped,** i.e. they are **not cumulative**
- Total code size of the considered functions: **3.5 kB**

branch imm	c.neg	c.not	c.mul	clz	c.lsbnot	c.sext/c.zext	shift+op	op+carry
8 B	36 B	30 B	0 B	50 B	0 B	0 B	106 B	64 B
0,23%	1,01%	0,84%	0,00%	1,41%	0,00%	0,00%	2,98%	1,80%

Code size saving on total

Tiny-FPU + Zfinx

1. Zfinx compiler
2. Zfinx area results
3. Further improvements

Zfinx implementation

- Added **one read port** to the INT register file (fmadd)
- Created new FP_SS:
  - without **FP LSU**
  - without **FP register file**
- **Accelerator interface** used as for the shared MULDIV unit

Zfinx implementation

- Using **PULP compiler**, I ran a **simple program** composed by:
  - 2 `fmadd.s`, 1 `fadd.s`, 1 `fmul.s`
- Very **simple program** to prevent the compiler from using non-supported PULP instructions
- **Further tests** with **Zfinx compiler** to validate the design

Zfinx compiler

1. Clone RISC-V GNU Toolchain https://github.com/riscv/riscv-gnu-toolchain
2. Clone Zfinx riscv-gcc https://github.com/pz9115/riscv-gcc/tree/riscv-gcc-10.2.0-zfinx
3. Clone Zfinx riscv-binutils-gdb
   https://github.com/pz9115/riscv-binutils-gdb/commit/bc7715694e2b97ba55791e8ef7409377bd7fb1f
4. Replace riscv-gcc and riscv-binutils with the Zfinx version
5. Installed using:
   ./configure --prefix=<destination_directory> **--with-arch=rv32imafzfinx --with-abi=ilp32** --with-cmodel=medlow --enable-multilib
6. Errors while compiling the libraries:
   /tmp/cc2bPziV.s: Assembler messages:
   /tmp/cc2bPziV.s:94: Error: illegal operands `fmadd.s a3,a1,a2,a3

Zfinx compiler

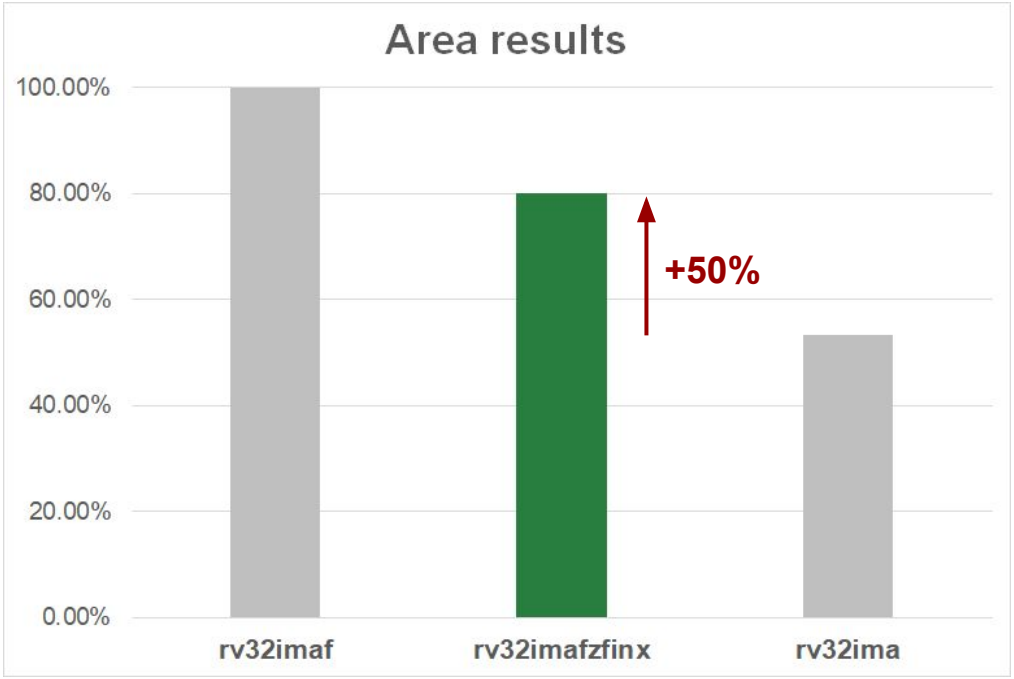
1. Previously installed using:

   ./configure --prefix=<destination_directory> --with-arch=rv32imafxzfinx --with-abi=ilp32 --with-cmodel=medlow --enable-multilib

2. Compiling code with: -march=rv32imafzfinx -mabi=ilp32

3. Error when compiling `fmadd.s`

Area Results



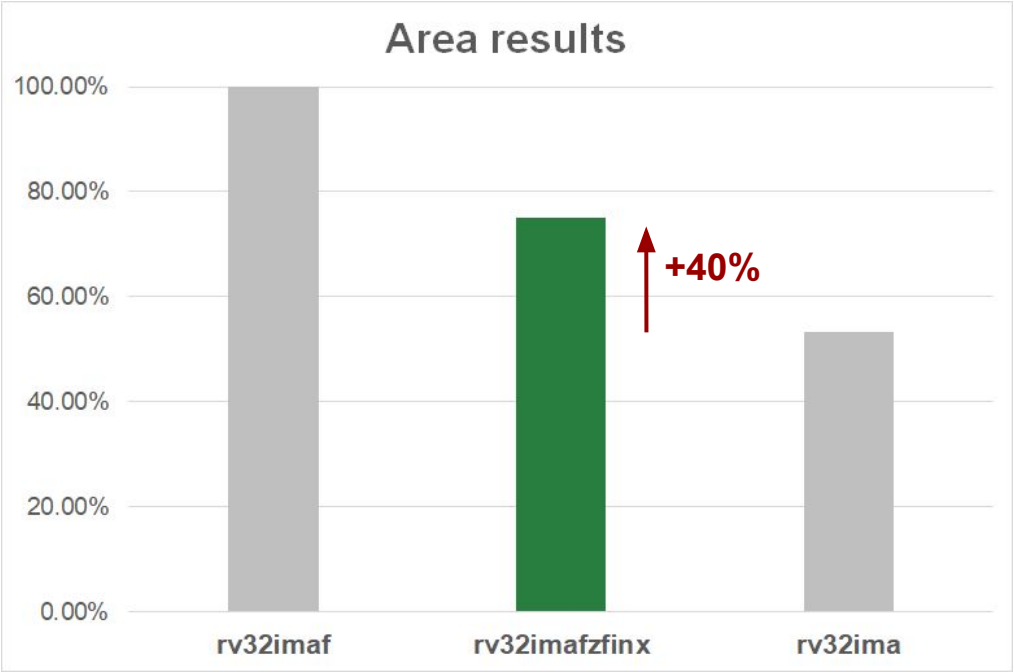
Area results

- Comparing Snitch integer core + muldiv unit [+ FP subsystem]
- **Zfinx** version is **1.5x larger** than the integer-only version of Snitch (muldiv included)
- The Zfinx implementation is **20% smaller** than the non-Zfinx Snitch implementation including a FP32 Tiny-FPU
- Removing the FP register led to a **50% smaller FP subsystem**

Area results

- Zfinx version adds ~17.5 kGE of overhead
- Tiny-FPU occupies ~9.5 kGE
- Spill registers wrapped around the FPU occupy ~3.4 kGE
- INT regfile gets ~10% bigger due to the new read port

Area Results - no FPU spill registers



Further improvements

- In the FP subsystem, spill registers are wrapped around the FPU to cut the combinational path
- When we add TinyFPU, this register are a duplication of TinyFPU internal registers
- We could remove the input spill registers already (25% of the area of FP32 Tiny-FPU) → ~1.43x larger
- We could remove the output spill registers adding one cycle of latency (removing another 10% of the area of FP32 Tiny-FPU) → **~1.4x larger**