IEEE Floating-Point Extensions for Containing Error in the RISC-V Architecture

Alexander Underwood, Tuan Nguyen, James Stine

VLSI Computer Architecture Research Group

Oklahoma State University

Stillwater, OK 74078 USA

{alexander.underwood, james.stine}@okstate.edu

Outline

- Introduction/Motivation
- Background
- Proposed Implementation
- Evaluation (Hardware and Software)
- Conclusion

Introduction

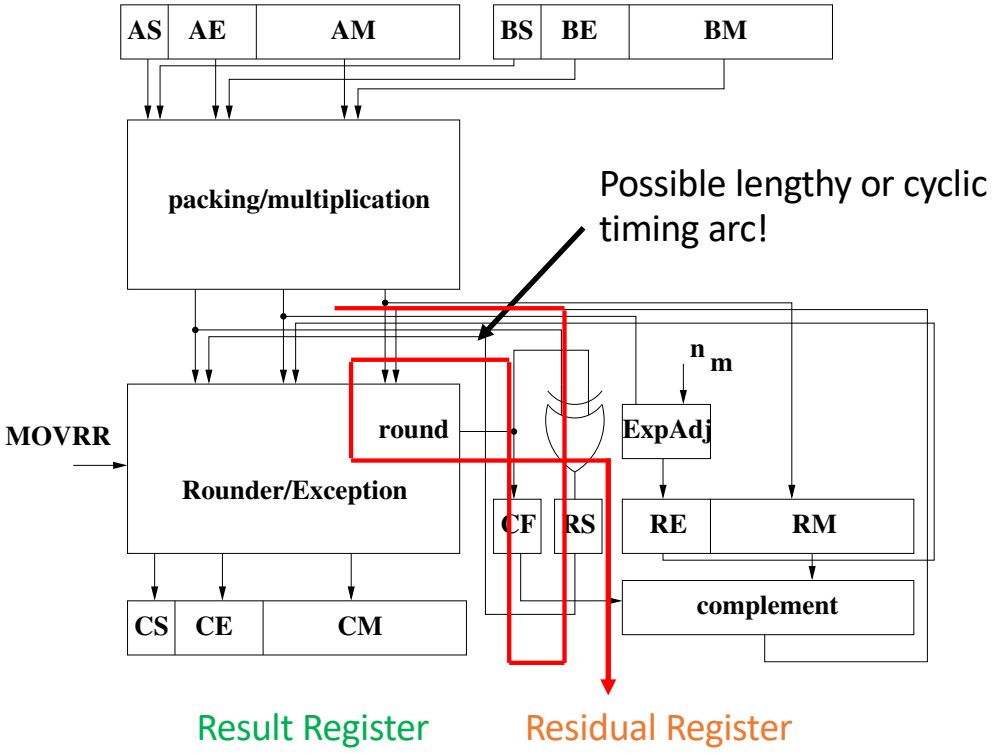
- Floating point operations are key in modern architectures.
- IEEE 754 has an inherent issue with precision due its dynamic range.
  - Rounding also eliminates information from the actual computation that can be disastrous if not careful.
- Software packages exist to make up for this precision loss, but are often very slow.
- There needs to be better mechanisms for containing error without sacrificing performance.

Native Pairs: The Residual Register

- Native Pairs were originally proposed by previous researchers but clarified by Douglas Priest in 1991.
  - Detailed algorithms presented in original article to gather arbitrary floating-point arithmetic.
  - Algorithms extension of ideas presented by W. Kahan from UCB.
- Main idea: Sequences of numbers can be used to obtain arbitrary precision
  - Second register that contains another floating point value of the same size.
- Paired with the standard result register for a floating point operation.
- Captures the portion of the result from the multiplier that would usually be discarded.

Adding a Residual Register

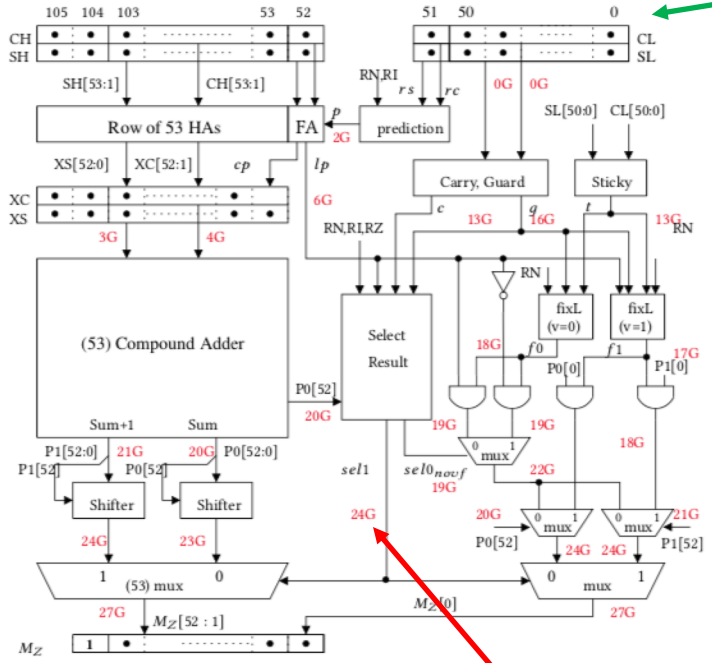
- Original architecture proposed by Dieter, Kaveti and Dietz in IEEE Computer Architecture Letters, 2007.
- Proposes hardware to allow native-pair arithmetic within a given functional unit.
- Avoids having to use fused-multiply-add (fma) units which are not always available.
- Adds a simple microarchitecture feature with relatively little performance penalty.
- New instruction in the ISA (`MOVRR`) to allow access to the residual register value if desired, otherwise works as standard IEEE 754 arithmetic.



[Dieter, Kaveti, Dietz, 2007]

Hardware Issues with Latency

IEEE 754 Rounding Unit for Multiplication



106-bit product is obtained during mantissa multiplication

- The problem with this microarchitecture proposal is that two floating-point mantissas (i.e., [1,2)) can possibly be greater than 2.
- This means waiting for the rounder unit to produce its completed result before the residual register can be properly obtained.
- Proposed solution is to replicate the rounder unit to alleviate bottleneck.

Linear Delay Gate Delays shown in Red

Overflow in mantissa multiplication is not detected until here almost towards end of rounding

Example : Residual Registers in C++

- The usefulness of a residual register can be demonstrated in C++.
- Programs were created to illustrate native-pair multiplication and its ability to add useful hardware for arbitrary precision.
- Normally, the result of a multiply is a single register solution.
- With a residual register, the output of a multiply spans 2 registers.

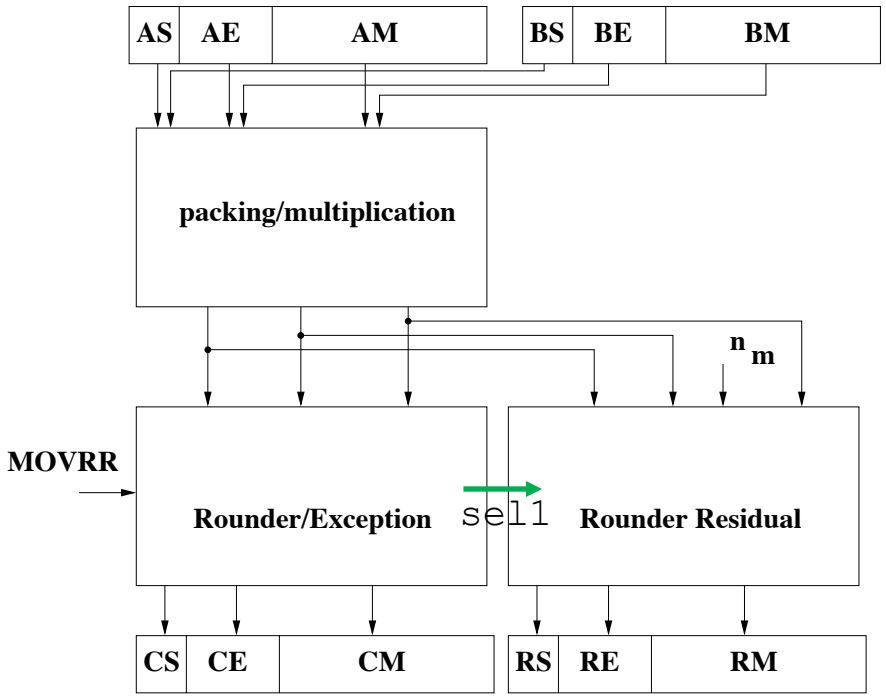
Demonstration in C++


```
x = 0x4010_0000_0000_0002 (4.0000000000000177635683940025046 46…)
y = 0x4000_0000_0000_0001 (2.0000000000000004440892098500626161…)
z = x * y

Using Quad Precision Library:
z = 0x4002_0000_0000_0000_3000_0000_0000_0200
    (8.000000000000005329070518200752182894337227847742 91
       1728565282786229673206435109023004770278930664 0625)

Using Residual Register:
z[1] = 0x4020_0000_0000_0003 (8.000000000000005329070518200751 3…)
z[2] = 0x39b0_0000_0000_0000 (7.88609052210118054117285652…E-31)
z[1] + z[2] = 8.000000000000005329070518200752182894337227
              847747291172856528278622967320643510902300 4
              7702789306640625
```

Architecture Modifications



- Duplicate a 2nd rounder portion to get residual register without having to wait again.
- Rounder Residual unit can compute with a compound adder (A+B, A+B+1, A+B+2, A+B+3) which is adequate for residual unit.
  - A+B+3 is utilized in case the need for TC in Priest's algorithm.
- Rounder unit within normal IEEE 754 still produces `sel1` signal to determine if complement is needed.
- Verified using System Verilog RTL with IEEE 754 validated floating-point multiplier unit.
- Can we perform core microarchitecture simulation for floating-point arithmetic to assess its performance?

gem5 (http://www.gem5.org)

- Runs real workloads and useful to analyze workloads that real users care about!
- Has a comprehensive model library for memory and I/O devices.
  - Modified for RISC-V as well as other ISAs.
- Extensively utilized for computer architecture research.
- Developed by University of Michigan and University of Wisconsin-Madison and freely available and open-source.
- Problem is that IEEE 754 arithmetic is only modified for use with ordinary compiler-given arithmetic operations (i.e., `Fd = Fs1*Fs2;`).
  - Solution: Adapt *gem5* to add true floating-point support.

Implementation of MOVRR for gem5

- New register RREG added to miscellaneous register list.
- Important to emphasize that RISC-V has binary128 capabilities, but using the residual register and algorithms from D. Priest that arbitrary precision floating-point arithmetic can be done with both hardware and software support.
- `MOVRR` instruction stores RREG in destination register (additions to F/D/Q ISA extension for RISC-V)
- RREG value is generated during a floating point multiply call.

```
movrr({{
    Fd_bits = xc->tcBase()->readMiscReg(RREG);
}});
```

MOVRR in RISC-V Toolchain

- Add MOVRR to the toolchain for assembly.
- No C/C++ language support – must use instruction from inside in asm block using a pragma definition.

```
asm volatile
(
    "movrr %[dest]"
    : [dest] "=f" (result)
);
```

SoftFloat Introduction

- Implemented by John Hauser at UCB based on W. Kahan's paranoia work.
  - http://www.jhauser.us/arithmetic/SoftFloat.html
- Complete implementation of IEEE 754 arithmetic in an efficient and compact library (also generates testvectors for hardware validation).
- Supports 16, 32, 64, 80, and 128-bit floating-point formats.
- Floating-point operations take place in the library similar to how they would on floating-point hardware.
- Allows access to intermediary floating-point states for measuring statistics or extracting values.

Adding SoftFloat to gem5

Benefits

- Allows access to intermediate values during floating point operations.
- Enables easy modifications to the way the floating point operations are performed.
- Has the ability to integrate true arithmetic exploration within *gem5*.
- Can be extended for other important arithmetic ideas as well as IEEE 754 arithmetic extensions (e.g., binary16, binary128).

Disadvantages

- Slightly longer simulation times due to more complex floating point operations.

Using SoftFloat in gem5

Standard double precision multiply implementation

```
fmul_d({{
  /* ... */

  Fd = Fs1*Fs2;

}}, FloatMultOp);
```

SoftFloat double precision multiply

```
fmul_d({{
  /* ... */

  float64_t fs1_sf;
  float64_t fs2_sf;
  float64_t fd_sf;

  std::memcpy(&fs1_sf, &Fs1, sizeof Fs1);
  std::memcpy(&fs2_sf, &Fs2, sizeof Fs2);

  fd_sf = f64_mul(fs1_sf, fs2_sf);

  std::memcpy(&Fd, &fd_sf, sizeof fd_sf);

}}, FloatMultOp);
```

SPEC Benchmarking in RISC-V gem5

- Several of the SPEC benchmarks from both the 2006 and 2017 set will run on gem5 in syscall emulation mode.
  - The benchmarks also compile using an up-to-date build of the RISC-V GNU toolchain.
- Profile the benchmark performance in RISC-V using the functional unit tracking and memory usage statistics.

SPEC Benchmarking gem5 Specifications

CPU Type	DerivO3CPU
L1d Cache Size	64kB
L1i Cache Size	32kB
Memory Type	DDR4_2400_8x8
Memory Size	16 GB
Memory Channels	1

gem5 RISC-V Architecture Details

SPEC06 Benchmarking Data

	444.namd	470.lbm
Simulated Seconds	17.55	10.26
Real Time Elapsed	28h 11m 11s	7h 59m 6s
# of Simulated Cycles	35,102,640,085	20,519,304,925
Total Function Calls	47,413,825,438	6,610,717,022
FloatMULT Calls	4,414,971,460 (9.31%)	1,273,446,080 (19.26%)

SPEC17 Benchmarking Data

	508.namd_r	519.lbm_r	619.lbm_s	644.nab_s
Simulated Seconds	10.03	1482.95	75.94	2.64
Real Time Elapsed	18h 47m 9s	55d 7h 13m 33s	37h 17m 52s	4h 57m 10s
# of Simulated Cycles	20,068,429,170	2,965,909,270,356	151,877,716,470	5,270,577,837
Total Function Calls	34,198,662,665	1,595,256,157,701	52,256,329,490	8,446,078,443
FloatMULT Calls	3,294,084,157 (9.63%)	326,750,716,512 (20.48%)	9,024,897,856 (17.27%)	1,192,676,971 (14.12%)

Value-Added Item

- Usefulness of gem5 is important for computer architecture research.
- Documented additions of SoftFloat as well as other architecture uses of gem5 at website.
  - May be useful along with other documentation available on the Internet.
- https://vlsiarch.ecen.okstate.edu/gem5/
- Suggestions, comments, modifications are welcome!

Hardware assessment

- Synthesized design using Synopsys® Design Compiler™ with topographical mode with cmos32soi IBM/GF technology.
- ARM®-standard cells with compiled Milkyway™ Galaxy™ database.
- Power numbers asses with 46,464 random vectors through a VCD file.

Item	Previous	Modified	
Area [um2]	17,757	18,907	+6.48%
Delay [ps]	361	361	
Power [mW]	30.59	31.30	+2.32%

Conclusion

- Implemented residual register using the RISC-V ISA and C++/gem5/HDL.
- Hardware gives extra information to the user for possible using residual register, variable precision or sticky accumulation options.
- Added new `movrr` instruction to RISC-V
- Added SoftFloat library to RISC-V gem5.
  - Key to adding innovation in functional units for arithmetic in gem5 and RISC-V.
  - Could be useful for machine learning or other ideas that utilize modification of FPU.
- Profile applicable uses using SPEC benchmarks in gem5 using the RISC-V ISA.