Out-of-order Floating Point Coprocessor for RISC-V ISA

Presentation by Claire Charron

Agenda

1. Out-of-order execution
2. Floating point
3. Coprocessor
4. RISC-V ISA

Agenda

1. ~~Out-of-order execution~~
2. Floating point
3. Coprocessor
4. RISC-V ISA

Agenda

1. ~~Out-of-order execution~~
2. RISC-V ISA
3. Floating point
4. Coprocessor

Agenda

1. Why? Part 1
2. RISC-V ISA
3. Floating point
4. Coprocessor
5. Why? Part 2

Why? Embedded systems.

Why? Robots.

RISC-V ISA

- RISC, obviously
- Completely open, documents viewable online
- Designed for hardware, not academia
  - Instruction formats reduce number of multiplexers
- Avoids preference for certain implementations
  - No branch delay slots
- 32-bit (RV32) or 64-bit (RV64)

RV32G

- RV32I: Integer base (required)
  - RV32M: Multiplication, division, modulus
  - RV32A: Atomic operations
  - **RV32F**: Single-precision floating-point
  - **RV32D**: Double-precision floating-point
  - RV32G: RV32IMAFD

R32FD

- 32 registers (double size for FD)
- IEEE floating-point standard
  - Sign bit, mantissa, exponent (scientific notation)
  - Conversion between single/double can be done with truncation

Instructions supported

- Add, subtract, multiply, divide
- Min, max
- Square root
- Fused multiply and accumulate (±a±bc)
- Conversion to integers
- Comparisons (<, =, ≥)
- Classification (±0, ±NaN, ±∞, etc.)

Processor

- Around 25, 100, or 180 MFLOPS depending on attached CPU
- Tested in hardware!

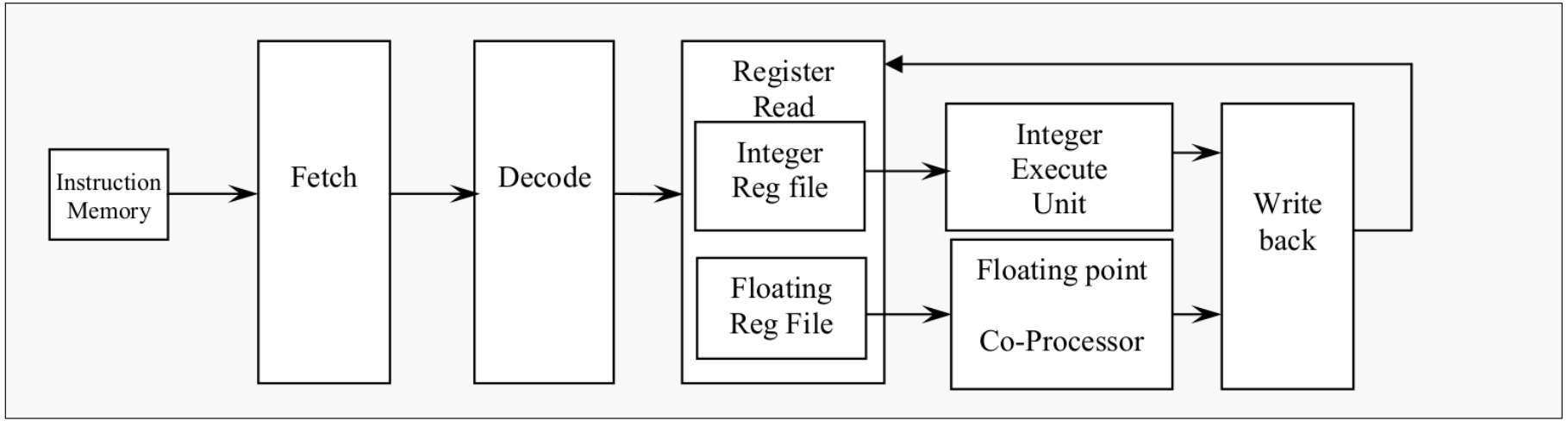


Fig. 1. Top Level Architecture For RISC V Processor

Coprocessor

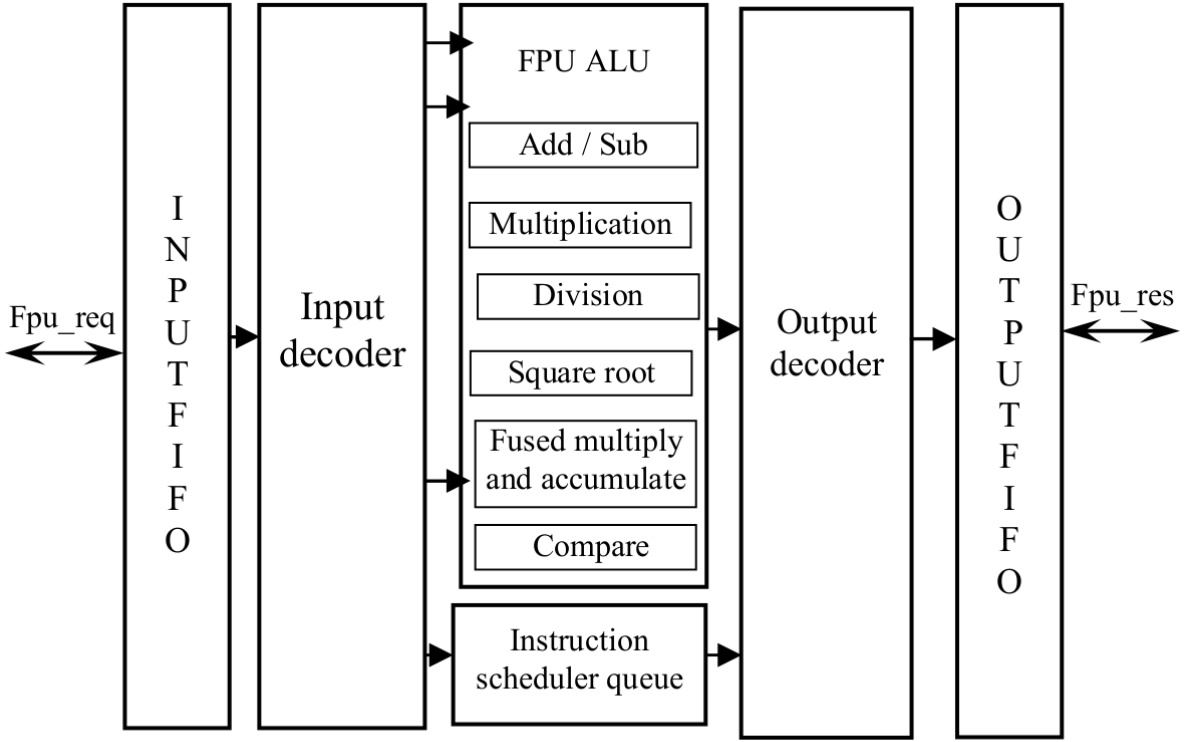


Fig. 2. Architecture of Out-of-order FPU Co-Processor

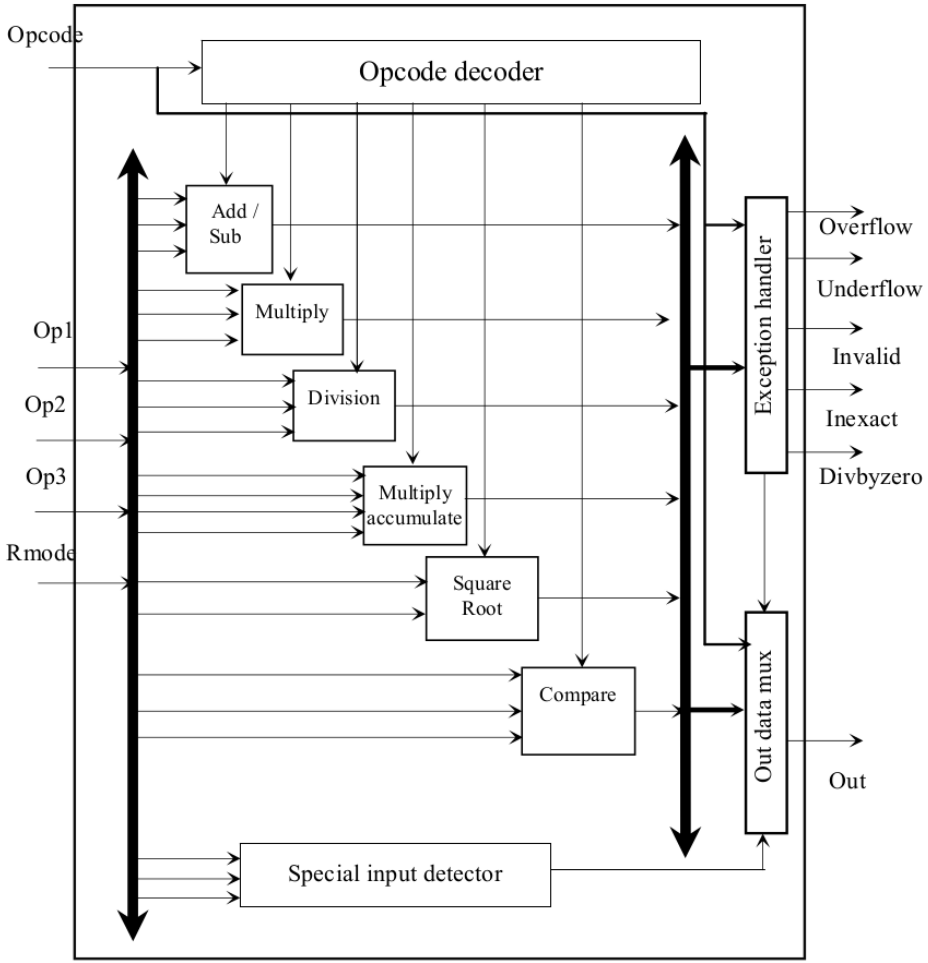


Fig. 4. Floating Point ALU Architecture

Preprocessing (in every pipeline)

- Decoding parts of float
  - Sign bit, mantissa, exponent
- Normalisation
  - e.g. 8×2^1 becomes 1×2^4

Postprocessing (again, every pipeline)

- Normalisation (again)
- Rounding
  - We usually leave extra bits at end of result to avoid rounding error
  - Very complicated

Pipeline breakdowns

1. How many stages (clock cycles)?
2. How many bits of rounding?
3. What do we do?

Addition

(5 stages, 3-bit buffer for rounding)

1. Swap the operands so that LHS is smaller.
2. Left-shift LHS so that its exponent is the same as RHS
3. Add/subtract the two mantissas depending on signs

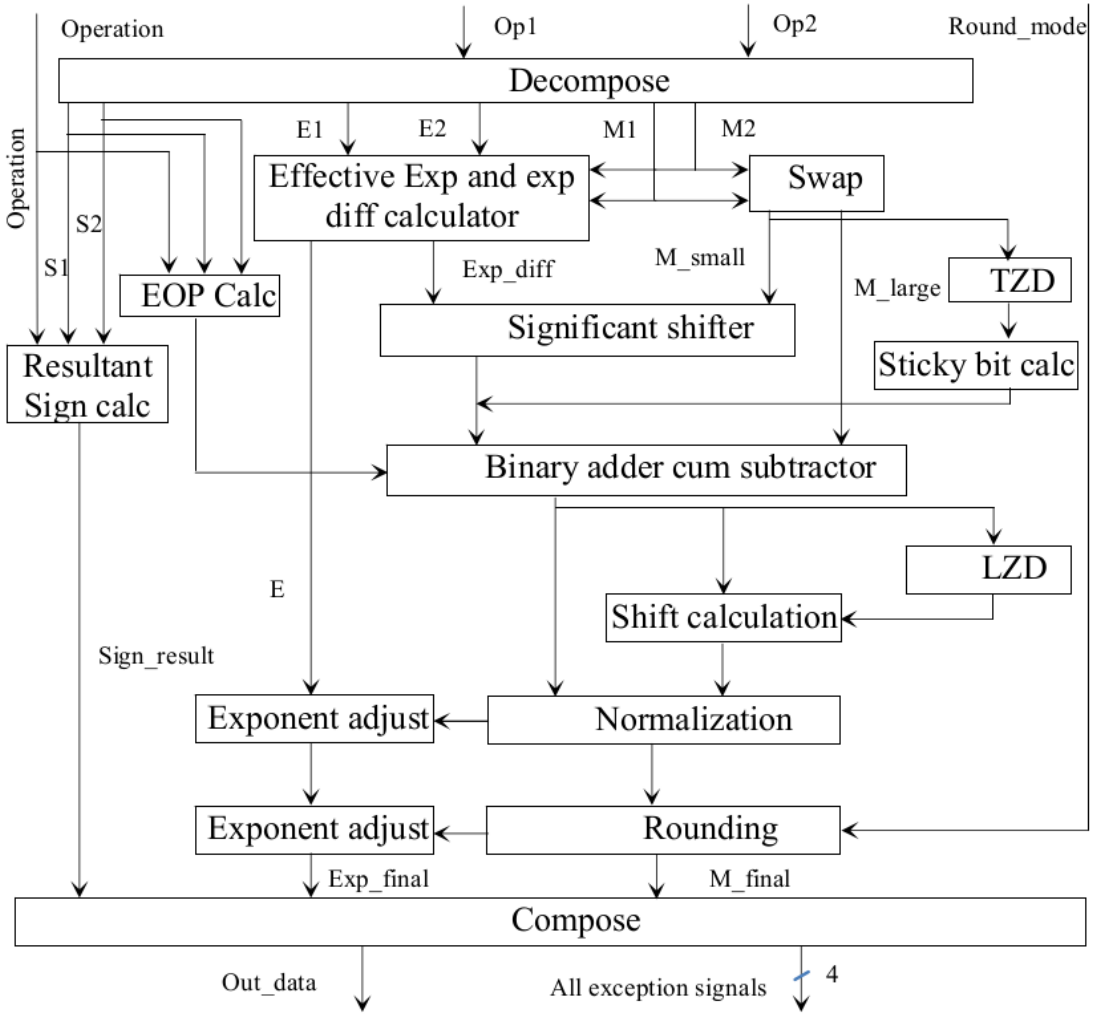
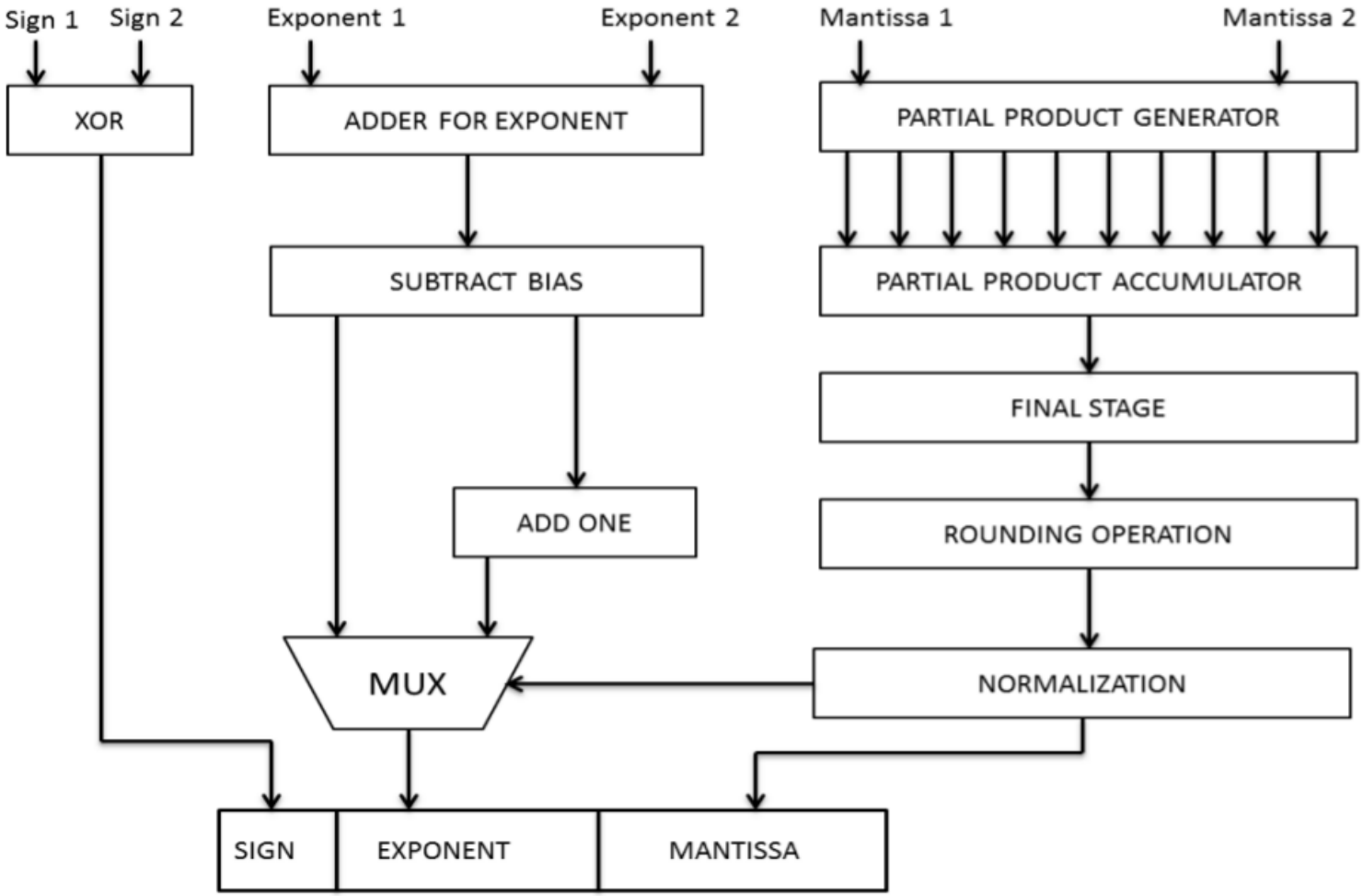


Fig. 5. Architecture for Floating Point Adder / Subtractor

Multiplication

(9 stages, 1-bit buffer for rounding)

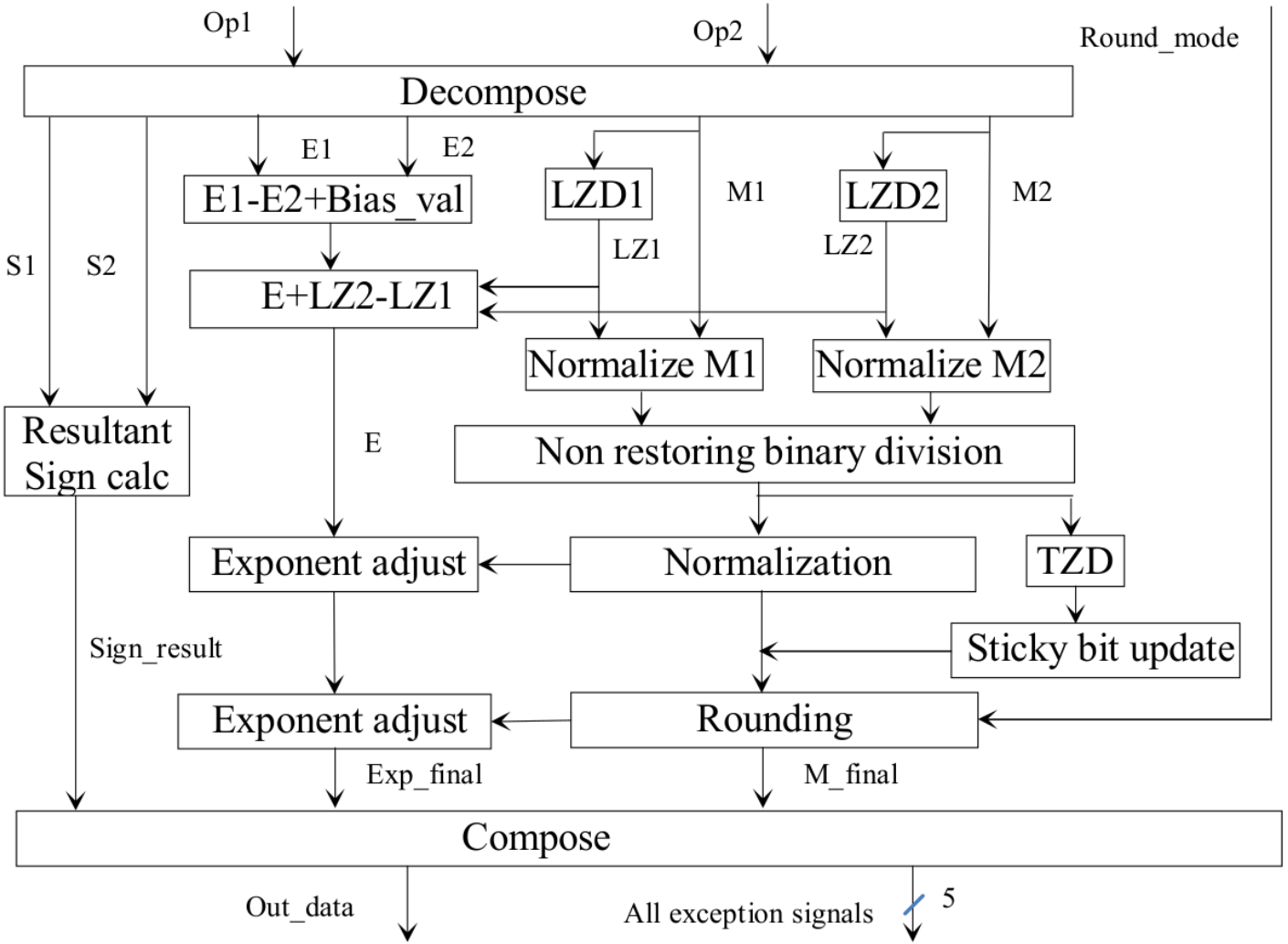
1. XOR signs
2. Add exponents
3. Multiply mantissas as if they were integers
    a. Split operands into N-bit chunks
    b. Have an N×N multiplication table
    c. Standard pen-and-paper algorithm in base-2^N


Sign 1
Sign 2
Exponent 1
Exponent 2
Mantissa 1
Mantissa 2
XOR
ADDER FOR EXPONENT
PARTIAL PRODUCT GENERATOR
SUBTRACT BIAS
PARTIAL PRODUCT ACCUMULATOR
FINAL STAGE
ADD ONE
ROUNDING OPERATION
MUX
NORMALIZATION
SIGN
EXPONENT
MANTISSA

Division

(31 stages, 1-bit buffer for rounding)

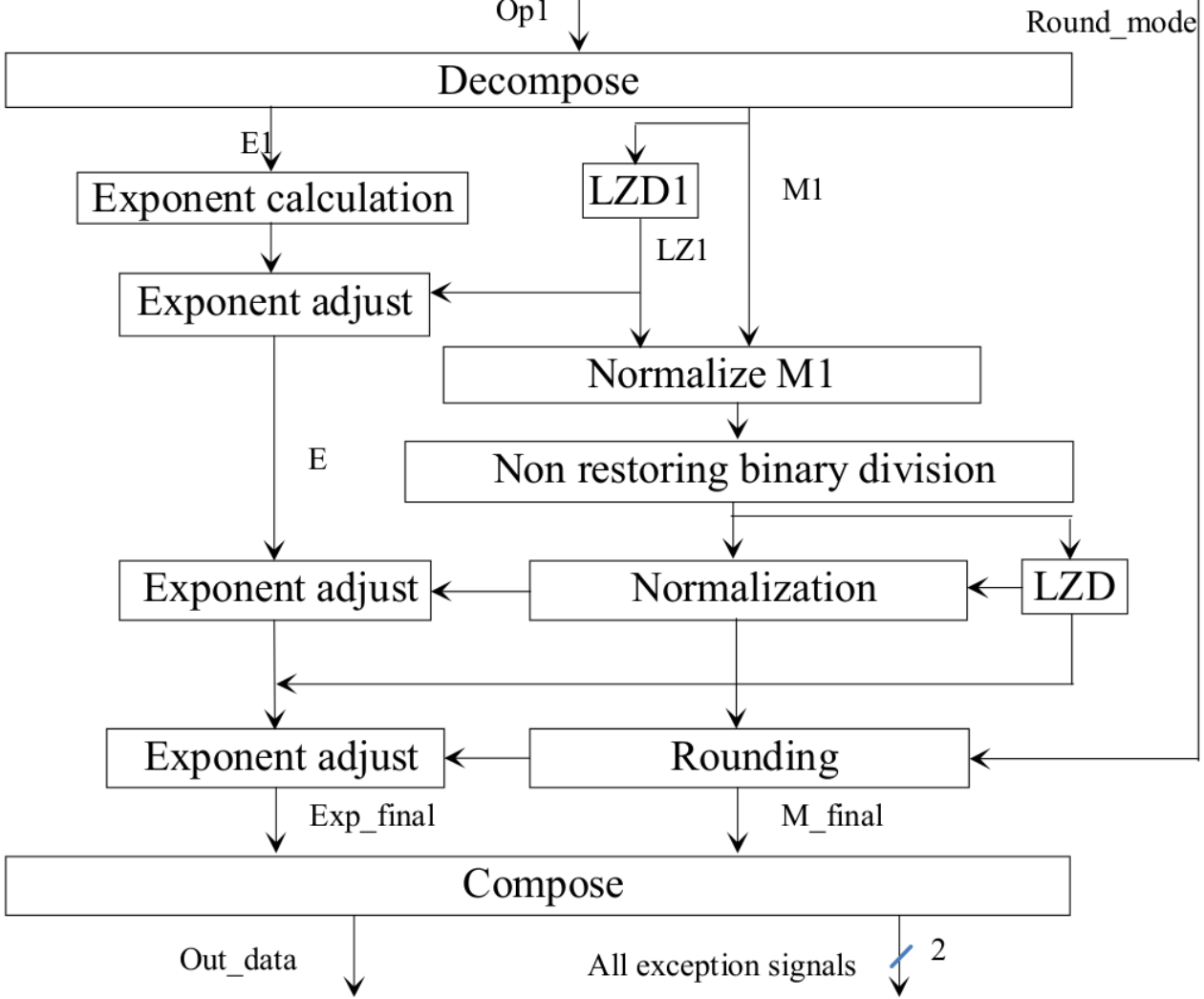
1. XOR signs
2. Subtract exponents
3. Divide mantissas as if they were integers
    a. Non-restoring binary division
    b. Done with repeated subtraction and left-shifts; look it up


Op1
Op2
Round_mode
Decompose
E1
E2
E1-E2+Bias_val
LZD1
M1
LZD2
M2
S1
S2
LZ1
LZ2
E+LZ2-LZ1
Normalize M1
Normalize M2
Resultant Sign calc
E
Non restoring binary division
Exponent adjust
Normalization
TZD
Sign_result
Sticky bit update
Exponent adjust
Rounding
Exp_final
M_final
Compose
Out_data
All exception signals
5

Square root

(31 stages, 1-bit buffer for rounding)

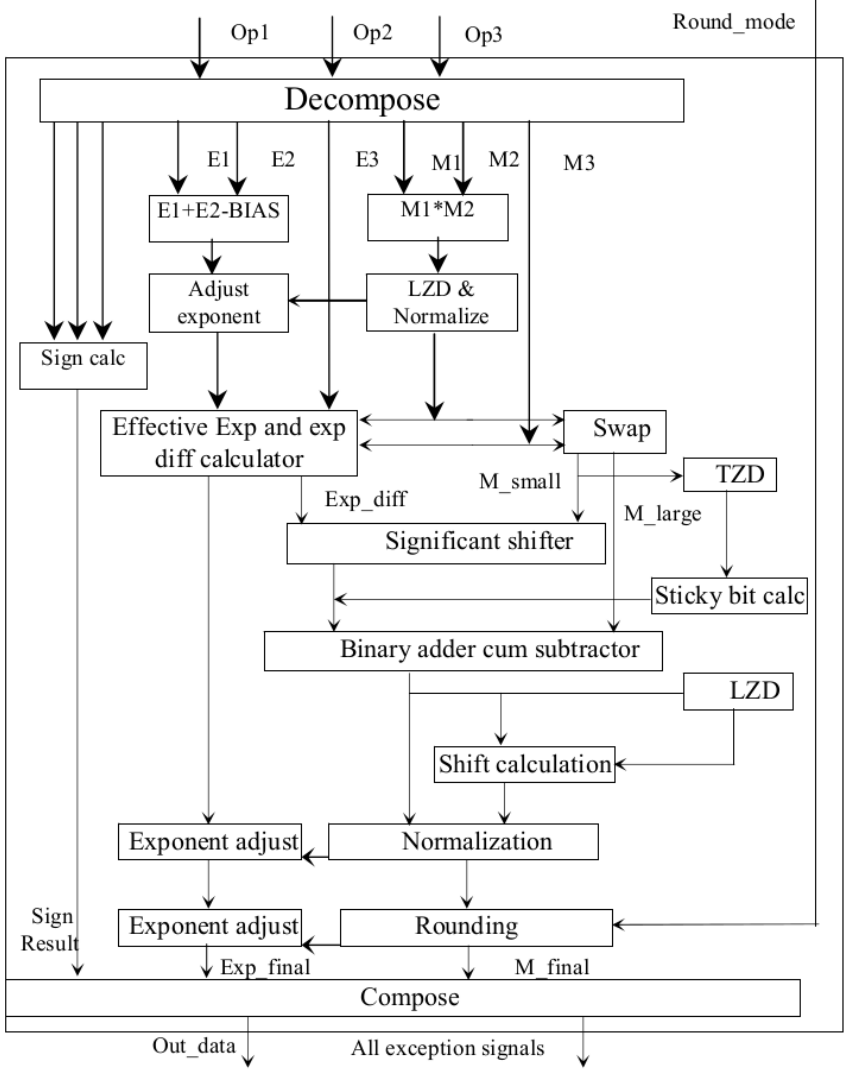
1. Divide exponent by two
    a. If exponent is odd, also have to modify mantissa
2. Find leading first bit, do a bunch of bit shifts
    a. Non-restoring square root
    b. Very similar to division
    c. Again, look it up


Op1
Round_mode
Decompose
E1
Exponent calculation
LZD1
M1
LZ1
Exponent adjust
Normalize M1
E
Non restoring binary division
Exponent adjust
Normalization
LZD
Exponent adjust
Rounding
Exp_final
M_final
Compose
Out_data
All exception signals
2

Fused multiply/accumulate (FMA)

(10 stages, 3-bit buffer for rounding)

1. Very similar to multiplication and addition together
2. Extra adding is done in one extra stage


Op1
Op2
Op3
Round_mode
Decompose
E1
E2
E3
M1
M2
M3
E1+E2-BIAS
M1*M2
Adjust exponent
LZD & Normalize
Sign calc
Effective Exp and exp diff calculator
Swap
M_small
TZD
Exp_diff
M_large
Significant shifter
Sticky bit calc
Binary adder cum subtractor
LZD
Shift calculation
Exponent adjust
Normalization
Sign Result
Exponent adjust
Rounding
Exp_final
M_final
Compose
Out_data
All exception signals

Other instructions

- Compare, classify, convert all happen in a single stage

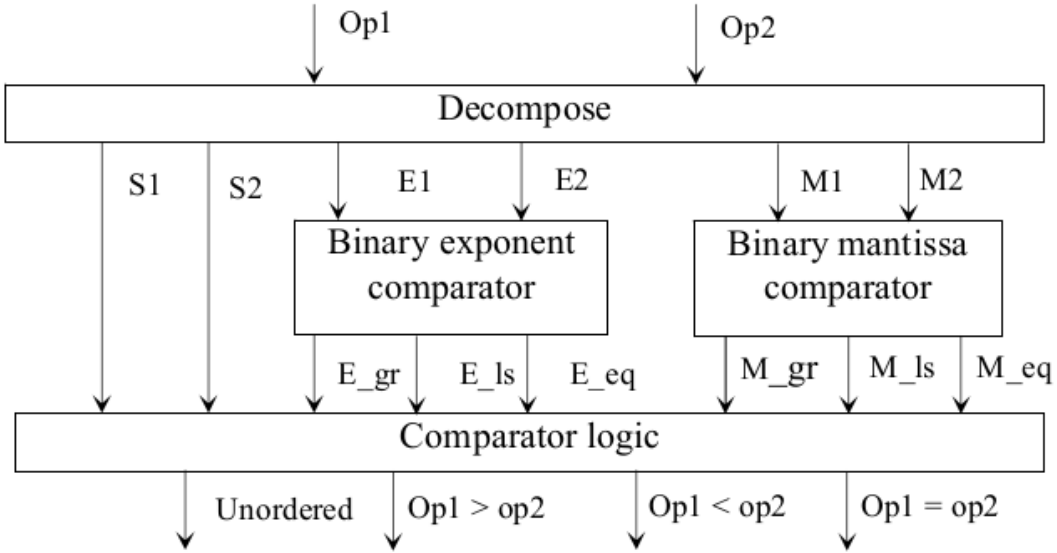


Fig. 10. Architecture For Comparator

Why? Part 2

- RISC-V is good for embedded systems because it’s open, RISC, doesn’t enforce hardware designs
- Floating-point math, although it takes a lot of hardware, happens in fewer cycles than integer emulation and saves power overall
- With FMA, addition is basically free

Questions?

- Patil, Vinayak et al. "Out Of Order Floating Point Coprocessor For RISC V ISA." International Symposium on VLSI Design and Test 19 (2015): n. pag. Web. 02 Mar 2016.
- “The RISC-V Instruction Set Manual.” RISC-V Foundation. RISC-V Foundation, 6 May 2014. Web. 18 Apr 2016. <http://riscv.org/specifications>