

Topics

- Integer Addition and Subtraction (3.2)
- Integer Multiplication (3.3)
- Integer Division (3.4)
- → **Floating point arithmetic (3.5)**

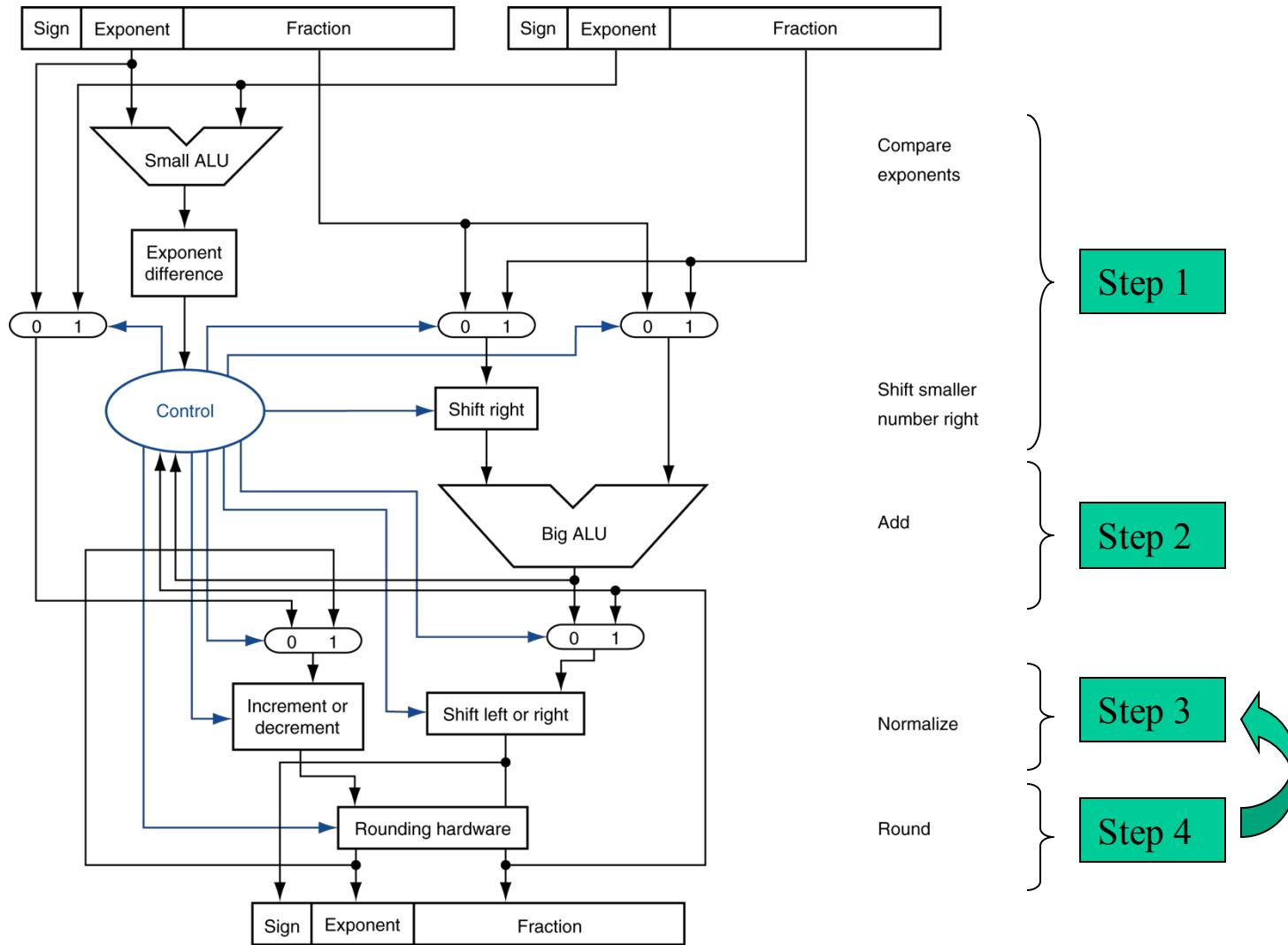
Floating-Point Addition

- Consider a 4-digit decimal example
 - $9.999 \times 10^1 + 1.610 \times 10^{-1}$
- 1. Align decimal points
 - Shift number with smaller exponent
 - $9.999 \times 10^1 + 0.016 \times 10^1$
- 2. Add significands
 - $9.999 \times 10^1 + 0.016 \times 10^1 = 10.015 \times 10^1$
- 3. Normalize result & check for over/underflow
 - 1.0015×10^2
- 4. Round and renormalize if necessary
 - 1.002×10^2

Floating-Point Addition

- Now consider a 4-bit binary example
 - $1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2}$ ($0.5 + -0.4375$)
- 1. Align binary points
 - Shift number with smaller exponent
 - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- 2. Add significands
 - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- 3. Normalize result & check for over/underflow
 - $1.000_2 \times 2^{-4}$, with no over/underflow
- 4. Round and renormalize if necessary
 - $1.000_2 \times 2^{-4}$ (no change) = 0.0625

FP Adder Hardware



FP Adder Hardware

- **Much more complex than integer adder**
- **Doing it in one clock cycle would take too long**
 - Much longer than integer operations
 - Slower clock would penalize all instructions
- **FP adder usually takes several cycles**
 - Can be pipelined

Floating-Point Multiplication (Decimal)

- Consider a 4-digit decimal example
 - $(1.110 \times 10^{10}) \times (9.200 \times 10^{-5})$
- 1. Add exponents
 - For biased exponents, subtract bias from sum
 - New exponent = $10 + (-5) = 5$
- 2. Multiply significands
 - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^5$
- 3. Normalize result & check for over/underflow
 - 1.0212×10^6
- 4. Round and renormalize if necessary
 - 1.021×10^6
- 5. Determine sign of result from signs of operands
 - $+1.021 \times 10^6$

Floating-Point Multiplication (Binary)

Now consider a 4-digit binary example

$$(1.000_2 \times 2^{-1}) \times (-1.110_2 \times 2^{-2}) \Leftrightarrow (0.5 \times -0.4375)$$

1. Add exponents

Unbiased: $-1 + -2 = -3$

Biased: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$

2. Multiply significands

$$1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$$

3. Normalize result & check for over/underflow

$1.110_2 \times 2^{-3}$ (no change) with no over/underflow

4. Round and renormalize if necessary

$1.110_2 \times 2^{-3}$ (no change)

5. Determine sign: +ve \times -ve \Rightarrow -ve

$$-1.110_2 \times 2^{-3} = -0.21875$$

FP Arithmetic Hardware

- **FP multiplier is of similar complexity to FP adder**
 - But uses a multiplier for significands instead of an adder
- **FP arithmetic hardware usually does**
 - Addition, subtraction, multiplication, division, reciprocal, square-root
 - FP \leftrightarrow integer conversion
- **Operations usually take several cycles**
 - Can be pipelined

FP Instructions in RISC-V

- **32 Separate FP registers**

- f0, f1, ... ,f31
- Each 64-bit wide for double precision
- Single-precision values stored in the lower 32 bits
- In the 1980s, FP processor (along with FP registers) was put on a second chip, called coprocessor
- Since the early 1990s, microprocessors have integrated FP on chip

- **FP instructions operate only on FP registers**

- Programs generally don't do integer ops on FP data, or vice versa

- **FP load and store instructions**

- flw, fsw, fld, fsd
 - e.g., flw f3, 32(sp)
- Base registers for addresses are still integer registers

FP Instructions in RISC-V

- **Single-precision arithmetic**

- fadd.s, fsub.s, fmul.s, fdiv.s, fsqrt.s
 - e.g., fadd.s f0, f1, f6

- **Double-precision arithmetic**

- fadd.d, fsub.d, fmul.d, fdiv.d, fsqrt.d
 - e.g., fmul.d f4, f4, f6

- **Single- and double-precision comparison**

- feq.s, flt.s, fle.s
- feq.d, flt.d, fle.d
- Set an integer register to 0 if the comparison is false and 1 if it is true
- Use beq, bne to branch on comparison result
- e.g.,
flt.s x5, f4, f5
beq x5, zero, L1

Coding Time

- **Download floats.s**