

# **Chapter 3**

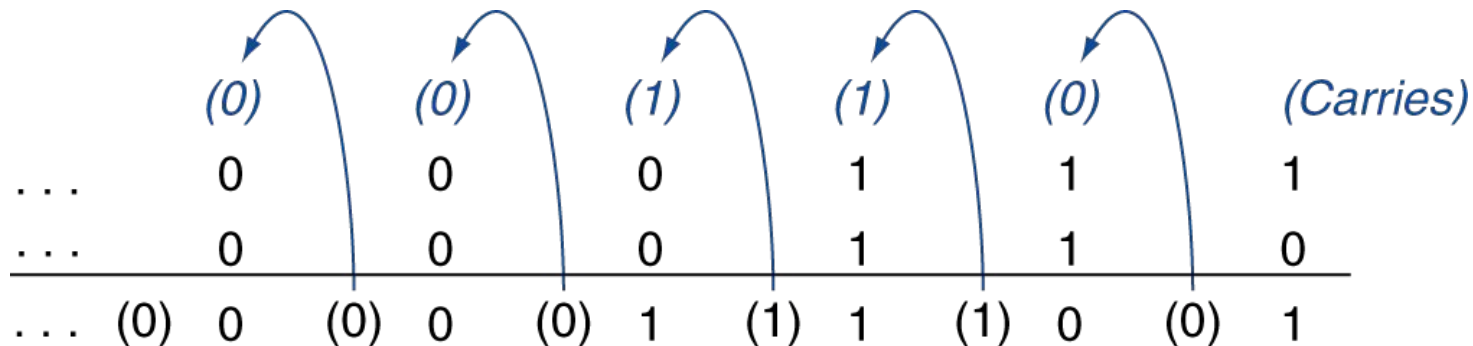
## **Arithmetic for Computers**

# Arithmetic for Computers

- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

# Integer Addition

## ■ Example: $7 + 6$



## ■ Overflow if result out of range

- Adding +ve and -ve operands, no overflow
- Adding two +ve operands
  - Overflow if result sign is 1
- Adding two -ve operands
  - Overflow if result sign is 0

# Integer Subtraction

- Add negation of second operand

- Example:  $7 - 6 = 7 + (-6)$

+7: 0000 0000 ... 0000 0111

-6: 1111 1111 ... 1111 1010

+1: 0000 0000 ... 0000 0001

- Overflow if result out of range
  - Subtracting two +ve or two -ve operands, no overflow
  - Subtracting +ve from -ve operand
    - Overflow if result sign is 0
  - Subtracting -ve from +ve operand
    - Overflow if result sign is 1

# Multiplication

- Start with long-multiplication approach

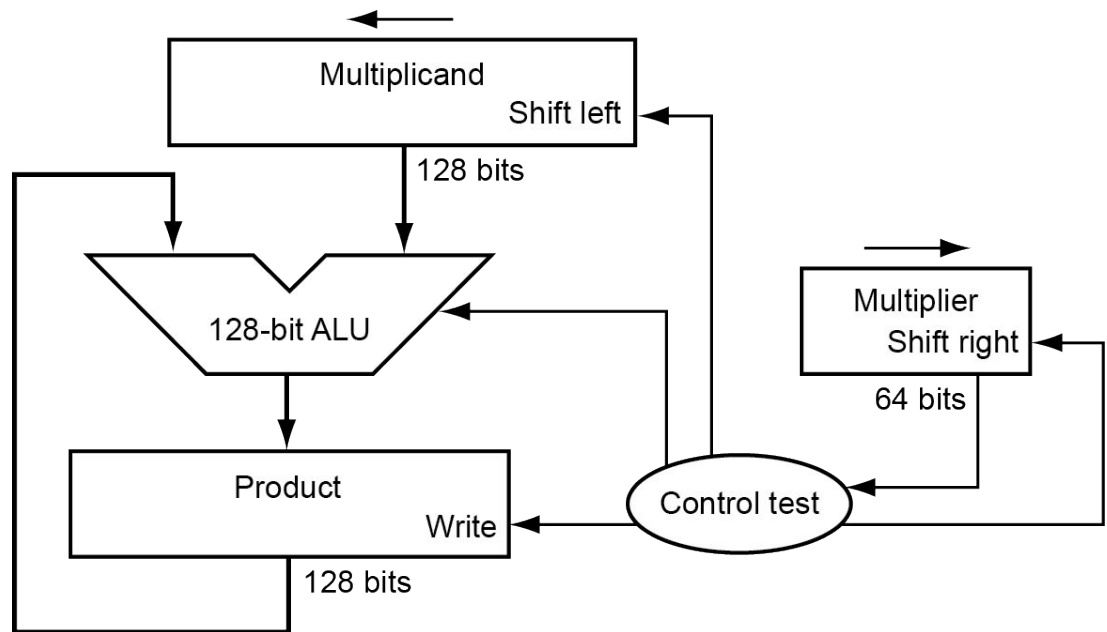
— multiplicand

— multiplier

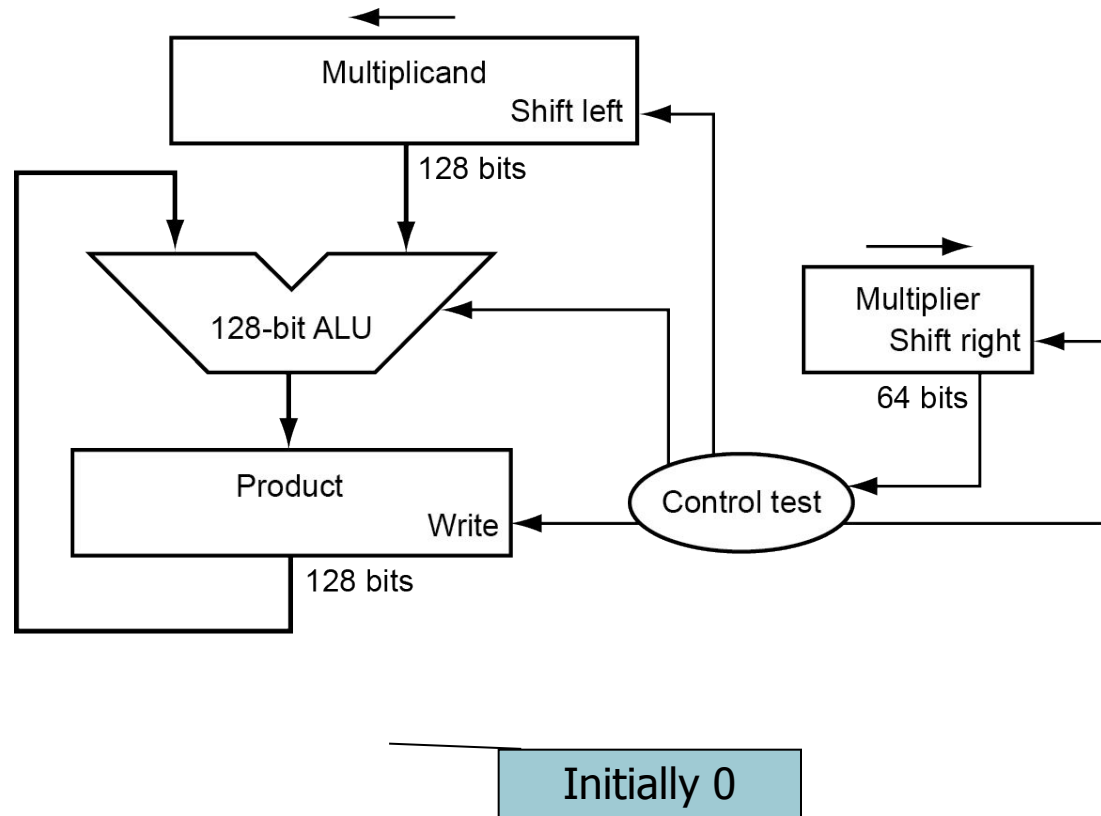
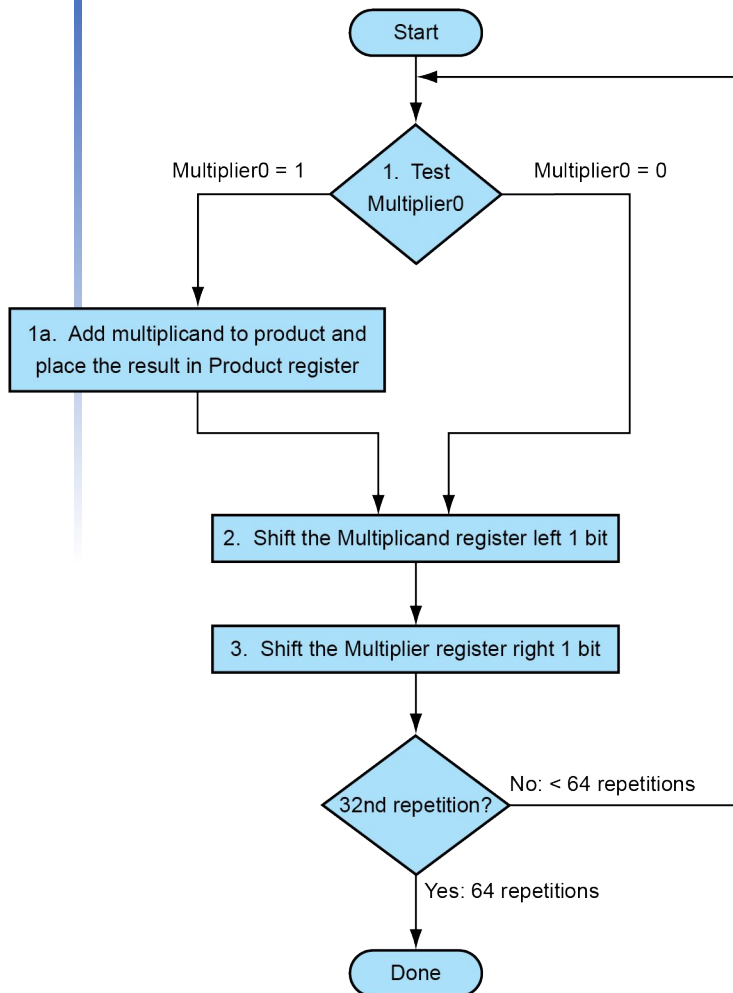
$$\begin{array}{r}
 1000 \\
 \times 1001 \\
 \hline
 1000 \\
 0000 \\
 0000 \\
 1000 \\
 \hline
 1001000
 \end{array}$$

— product

Length of product is  
the sum of operand  
lengths

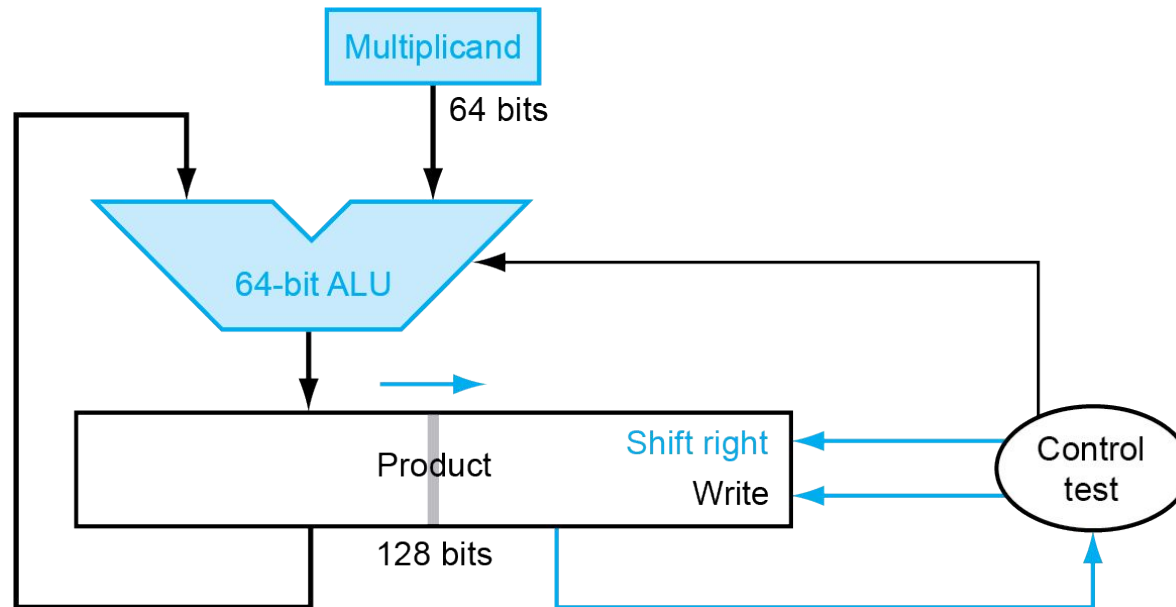


# Multiplication Hardware



# Optimized Multiplier

- Perform steps in parallel: add/shift



- One cycle per partial-product addition
  - That's ok, if frequency of multiplications is low

# RISC-V Multiplication

- Four multiply instructions:
  - `mul`: multiply
    - Gives the lower 64 bits of the product
  - `mulh`: multiply high
    - Gives the upper 64 bits of the product, assuming the operands are signed
  - `mulhu`: multiply high unsigned
    - Gives the upper 64 bits of the product, assuming the operands are unsigned
  - `mulhsu`: multiply high signed/unsigned
    - Gives the upper 64 bits of the product, assuming one operand is signed and the other unsigned
- Use `mulh` result to check for 64-bit overflow



# Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - $-2.34 \times 10^{56}$
  - $+0.002 \times 10^{-4}$
  - $+987.02 \times 10^9$
- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types float and double in C

normalized

not normalized



# Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of representations
  - Portability issues for scientific code
- Now almost universally adopted
- Two representations
  - Single precision (32-bit)
  - Double precision (64-bit)

# IEEE Floating-Point Format

single: 8 bits

double: 11 bits

single: 23 bits

double: 52 bits



$$x = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: sign bit (0  $\Rightarrow$  non-negative, 1  $\Rightarrow$  negative)
- Normalize significand:  $1.0 \leq |\text{significand}| < 2.0$ 
  - Always has a leading pre-binary-point 1 bit, so no need to represent it explicitly (hidden bit)
  - Significand is Fraction with the “1.” restored
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single: Bias = 127; Double: Bias = 1203

# Single-Precision Range

- Exponents 00000000 and 11111111 reserved
- Smallest value
  - Exponent: 00000001  
 $\Rightarrow$  actual exponent =  $1 - 127 = -126$
  - Fraction: 000...00  $\Rightarrow$  significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
  - exponent: 11111110  
 $\Rightarrow$  actual exponent =  $254 - 127 = +127$
  - Fraction: 111...11  $\Rightarrow$  significand  $\approx 2.0$
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

# Double-Precision Range

- Exponents 0000...00 and 1111...11 reserved
- Smallest value
  - Exponent: 000000000001  
 $\Rightarrow$  actual exponent =  $1 - 1023 = -1022$
  - Fraction: 000...00  $\Rightarrow$  significand = 1.0
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
  - Exponent: 111111111110  
 $\Rightarrow$  actual exponent =  $2046 - 1023 = +1023$
  - Fraction: 111...11  $\Rightarrow$  significand  $\approx 2.0$
  - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

# Floating-Point Precision

- Relative precision
  - all fraction bits are significant
  - Single: approx  $2^{-23}$ 
    - Equivalent to  $23 \times \log_{10} 2 \approx 23 \times 0.3 \approx 6$  decimal digits of precision
  - Double: approx  $2^{-52}$ 
    - Equivalent to  $52 \times \log_{10} 2 \approx 52 \times 0.3 \approx 16$  decimal digits of precision

# Floating-Point Example

- Represent  $-0.75$ 
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - $S = 1$
  - Fraction =  $1000\dots00_2$
  - Exponent =  $-1 + \text{Bias}$ 
    - Single:  $-1 + 127 = 126 = 01111110_2$
    - Double:  $-1 + 1023 = 1022 = 011111111110_2$
- Single:  $10111111101000\dots00$
- Double:  $101111111111101000\dots00$

# Floating-Point Example

- What number is represented by the single-precision float

11000000101000...00

- $S = 1$

- Fraction =  $01000...00_2$

- Exponent =  $10000001_2 = 129$

- $$\begin{aligned} x &= (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)} \\ &= (-1) \times 1.25 \times 2^2 \\ &= -5.0 \end{aligned}$$



# Denormal Numbers

- Exponent = 000...0  $\Rightarrow$  hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-\text{Bias}}$$

- Smaller than normal numbers
  - allow for gradual underflow, with diminishing precision

- Denormal with fraction = 000...0

$$x = (-1)^S \times (0 + 0) \times 2^{-\text{Bias}} = \pm 0.0$$

— Two representations of 0.0!

# Infinites and NaNs

- Exponent = 111...1, Fraction = 000...0
  - $\pm$ Infinity
  - Can be used in subsequent calculations, avoiding need for overflow check
- Exponent = 111...1, Fraction  $\neq$  000...0
  - Not-a-Number (NaN)
  - Indicates illegal or undefined result
    - e.g., 0.0 / 0.0
  - Can be used in subsequent calculations

# 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
  - $1.0015 \times 10^2$
- 4. Round and renormalize if necessary
  - $1.002 \times 10^2$

# Floating-Point Addition

- Now consider a 4-digit 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

# Floating-Point Multiplication

- 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
  - $1.0212 \times 10^6$
- 4. Round and renormalize if necessary
  - $1.021 \times 10^6$
- 5. Determine sign of result from signs of operands
  - $+1.021 \times 10^6$

# Floating-Point Multiplication

- Now consider a 4-digit binary example
  - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$  ( $0.5 \times -0.4375$ )
- 1. Add exponents
  - Unbiased:  $-1 + -2 = -3$
  - Biased:  $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- 2. Multiply significands
  - $1.000_2 \times 1.110_2 = 1.1102 \Rightarrow 1.110_2 \times 2^{-3}$
- 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
  - $\text{FP} \leftrightarrow \text{integer}$  conversion
- Operations usually takes several cycles
  - Can be pipelined

# FP Instructions in RISC-V

- Separate FP registers: f0, ..., f31
  - double-precision
  - single-precision values stored in the lower 32 bits
- FP instructions operate only on FP registers
  - Programs generally don't do integer ops on FP data, or vice versa
  - More registers with minimal code-size impact
- FP load and store instructions
  - flw, fld
  - fsw, fsd



# FP Instructions in RISC-V

- Single-precision arithmetic
  - fadd.s, fsub.s, fmul.s, fdiv.s, fsqrt.s
    - e.g., fadds.s f2, f4, f6
- Double-precision arithmetic
  - fadd.d, fsub.d, fmul.d, fdiv.d, fsqrt.d
    - e.g., fadd.d f2, f4, f6
- Single- and double-precision comparison
  - feq.s, flt.s, fle.s
  - feq.d, flt.d, fle.d
  - Result is 0 or 1 in integer destination register
    - Use beq, bne to branch on comparison result
- Branch on FP condition code true or false
  - B.cond

# Concluding Remarks

- Bits have no inherent meaning
  - Interpretation depends on the instructions applied
- Computer representations of numbers
  - Finite range and precision
  - Need to account for this in programs

# Concluding Remarks

- ISAs support arithmetic
  - Signed and unsigned integers
  - Floating-point approximation to reals
- Bounded range and precision
  - Operations can overflow and underflow