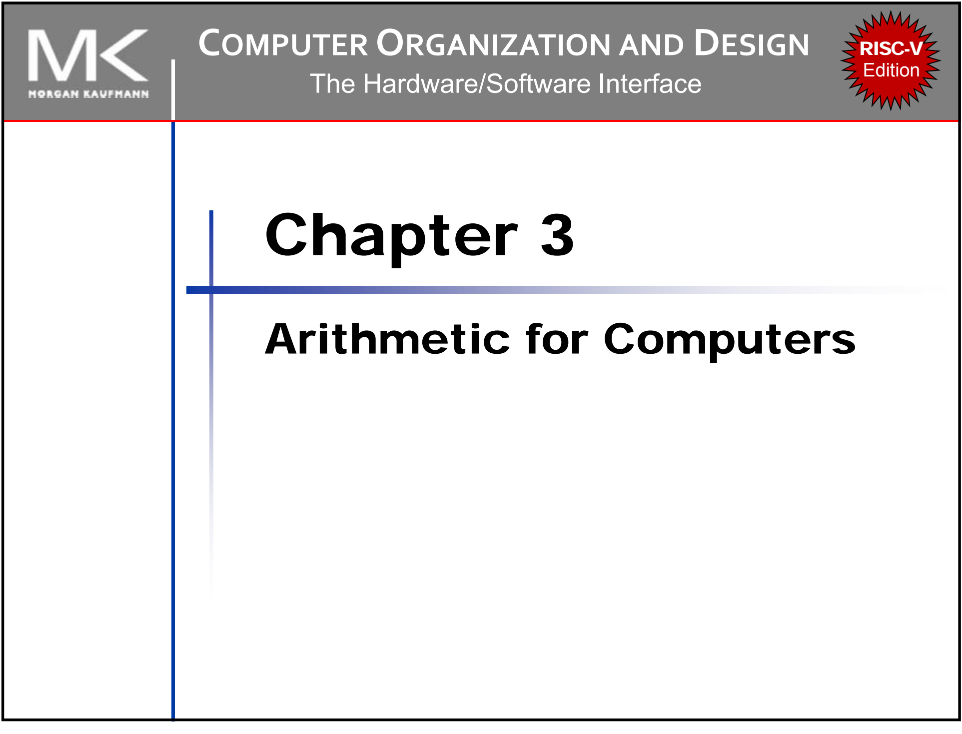


Arithmetic for Computers

§3.1 Introduction

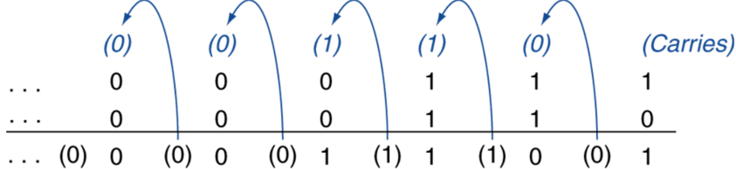
- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Chapter 3 — Arithmetic for Computers — 2

Integer Addition

§3.2 Addition and Subtraction

- 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

Chapter 3 — Arithmetic for Computers — 3

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

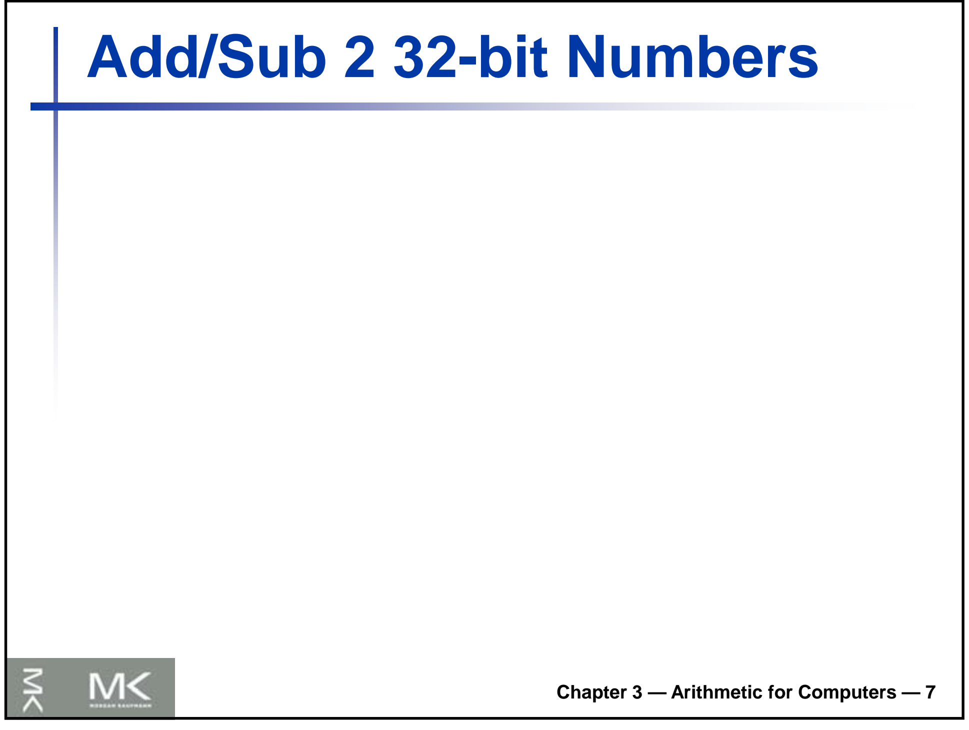
  +7: 0000 0000 … 0000 0111
  –6: 1111 1111 … 1111 1010
  +1: 0000 0000 … 0000 0001
- Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

Chapter 3 — Arithmetic for Computers — 4

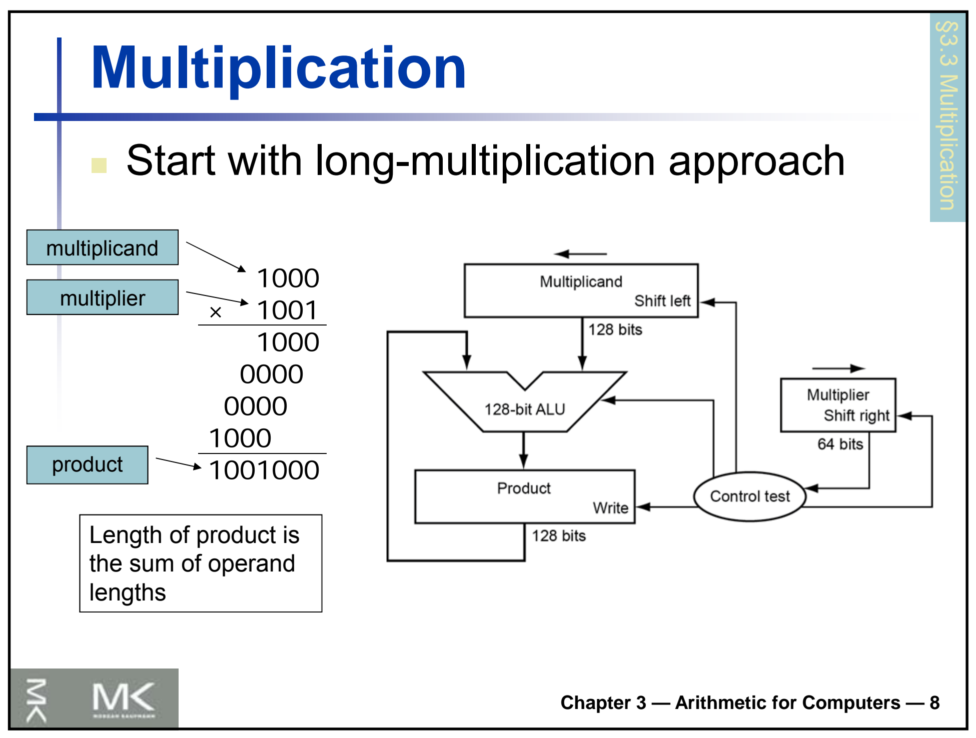
Arithmetic for Multimedia

- Graphics and media processing operates on vectors of 8-bit and 16-bit data
  - Use 64-bit adder, with partitioned carry chain
    - Operate on 8×8-bit, 4×16-bit, or 2×32-bit vectors
  - SIMD (single-instruction, multiple-data)
- Saturating operations
  - On overflow, result is largest representable value
    - c.f. 2s-complement modulo arithmetic
  - E.g., clipping in audio, saturation in video

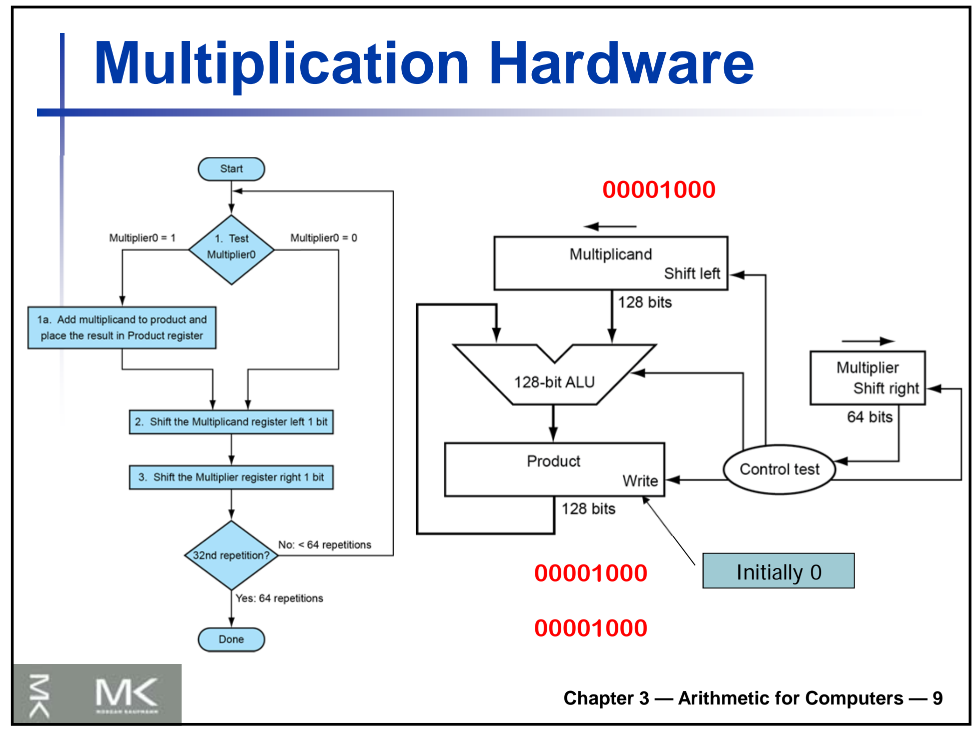
Adding 2 32-bit Numbers

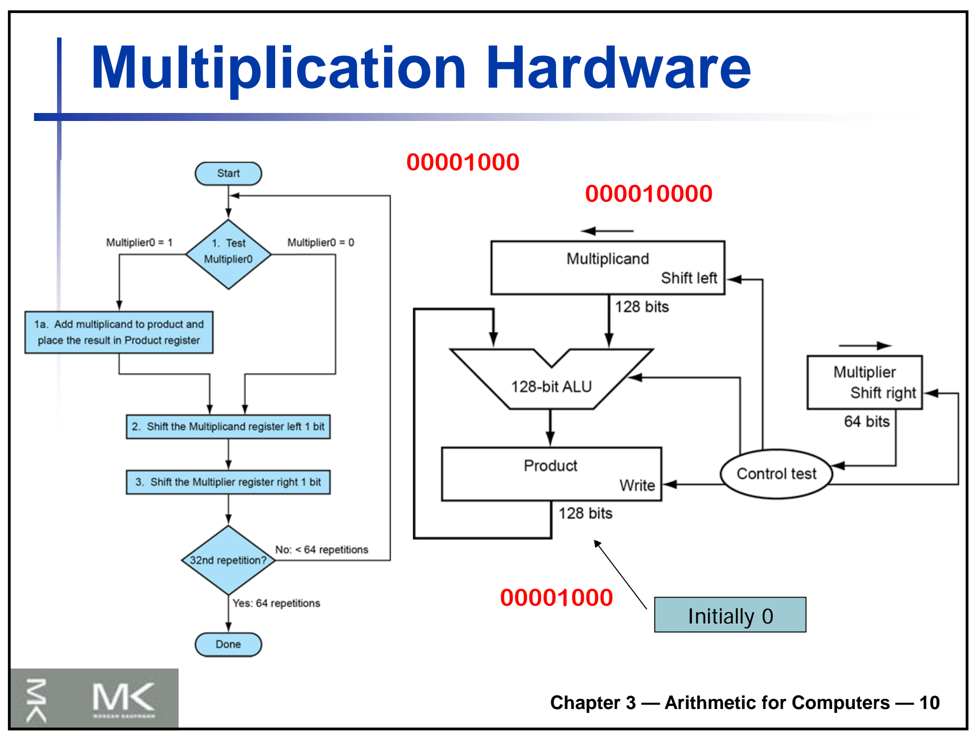


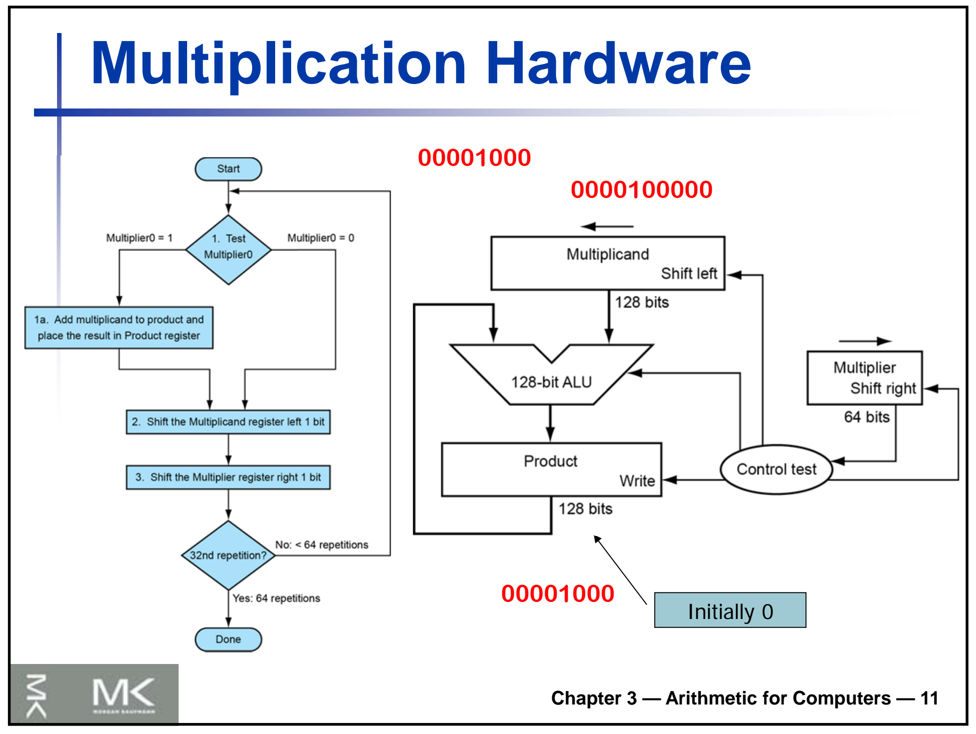
Add/Sub 2 32-bit Numbers
Chapter 3 — Arithmetic for Computers — 7

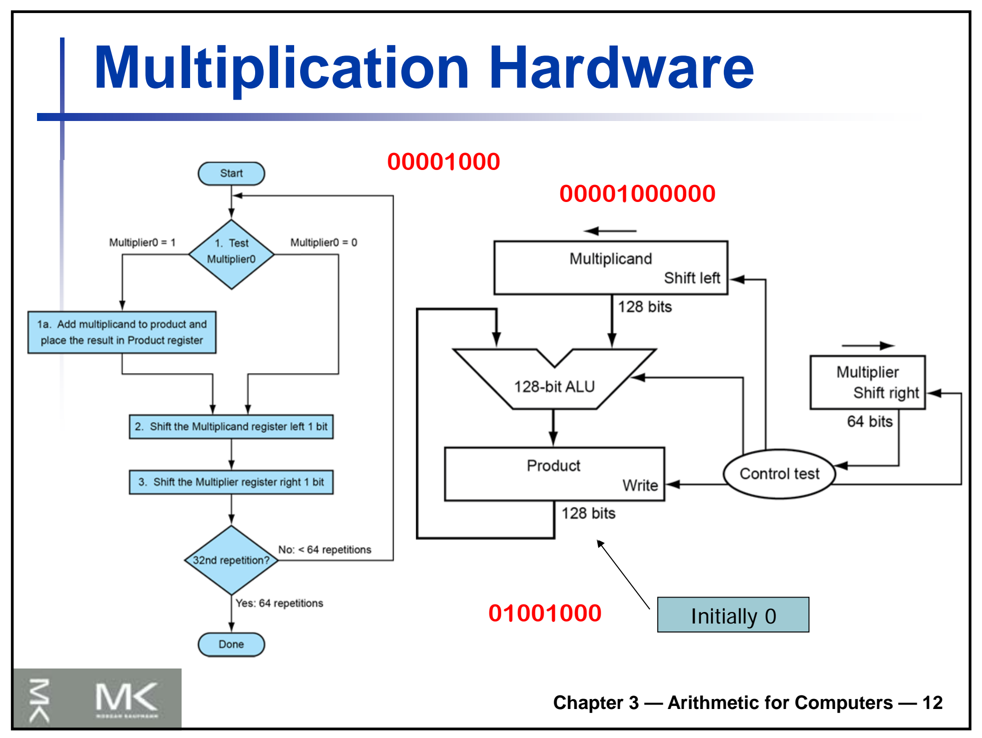


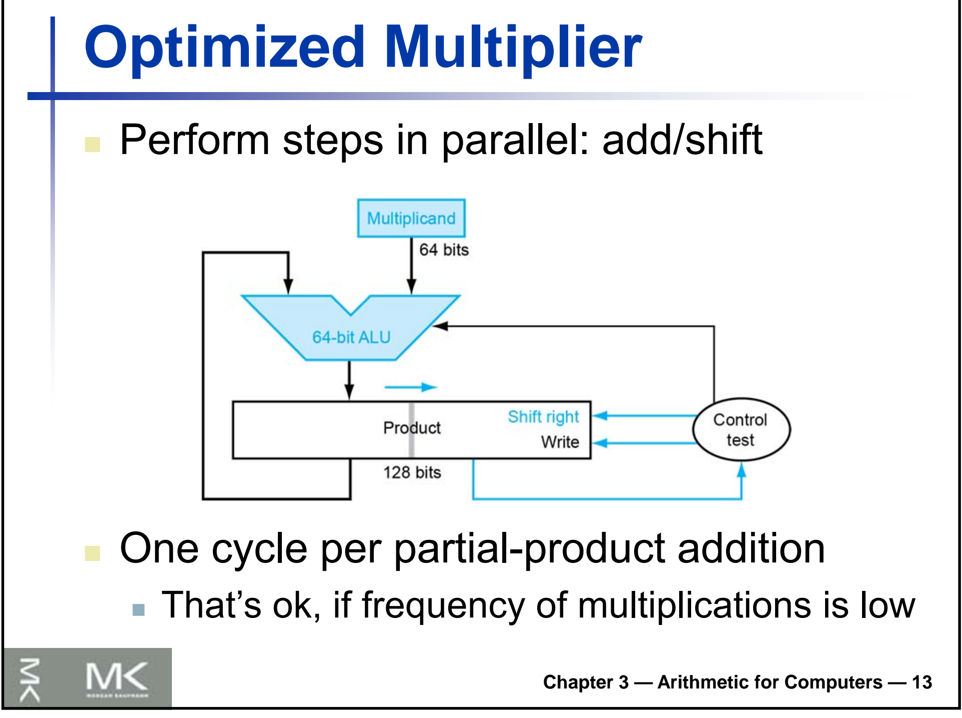
Multiplication
Start with long-multiplication approach
multiplicand
multiplier
product
1000
× 1001
1000
0000
0000
1000
1001000
Length of product is the sum of operand lengths
Multiplicand
Shift left
128 bits
128-bit ALU
Product
Write
128 bits
Multiplier
Shift right
64 bits
Control test
§3.3 Multiplication
Chapter 3 — Arithmetic for Computers — 8

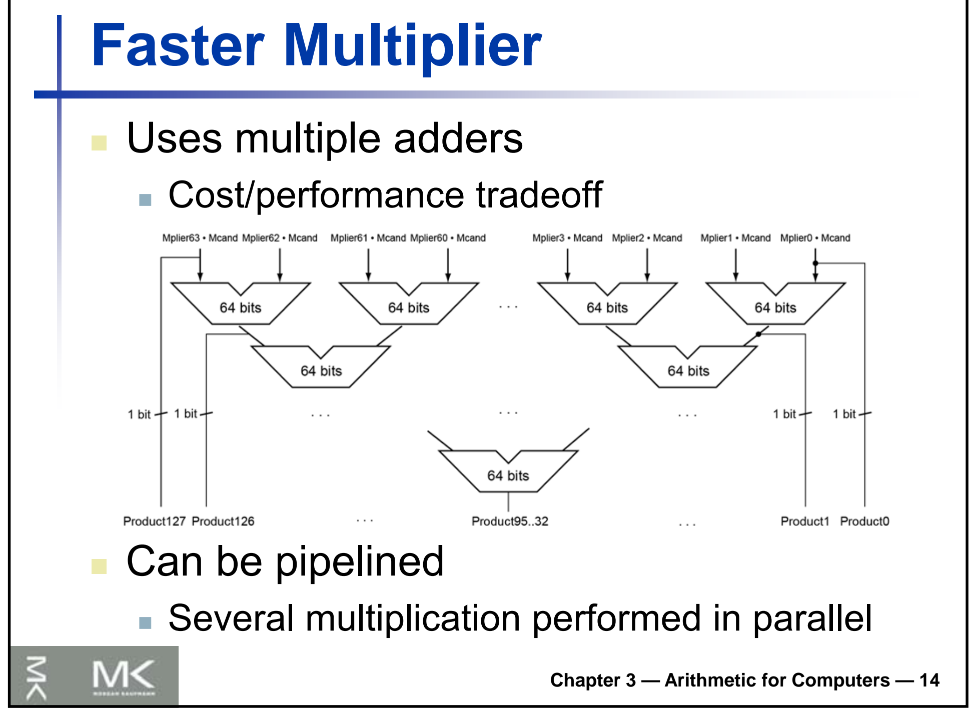

Multiplication Hardware
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done
00001000
Multiplicand
Shift left
128 bits
128-bit ALU
Multiplier
Shift right
64 bits
Product
Write
Control test
128 bits
00001000
Initially 0
00001000
Chapter 3 — Arithmetic for Computers — 9


Multiplication Hardware
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done
00001000
000010000
Multiplicand
Shift left
128 bits
128-bit ALU
Multiplier
Shift right
64 bits
Product
Write
Control test
128 bits
00001000
Initially 0
Chapter 3 — Arithmetic for Computers — 10


Multiplication Hardware
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done
00001000
0000100000
Multiplicand
Shift left
128 bits
128-bit ALU
Multiplier
Shift right
64 bits
Product
Write
Control test
128 bits
00001000
Initially 0
Chapter 3 — Arithmetic for Computers — 11


Multiplication Hardware
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done
00001000
00001000000
Multiplicand
Shift left
128 bits
128-bit ALU
Multiplier
Shift right
64 bits
Product
Write
Control test
128 bits
01001000
Initially 0
Chapter 3 — Arithmetic for Computers — 12

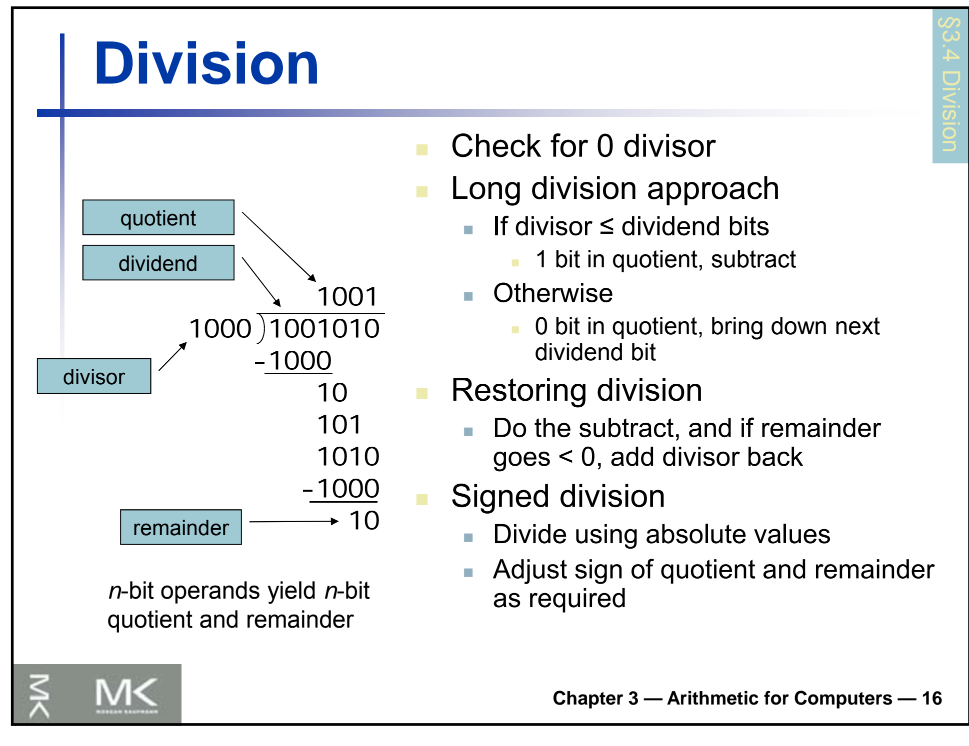

Optimized Multiplier
Perform steps in parallel: add/shift
Multiplicand
64 bits
64-bit ALU
Product
Shift right
Write
Control test
128 bits
One cycle per partial-product addition
That's ok, if frequency of multiplications is low
Chapter 3 — Arithmetic for Computers — 13

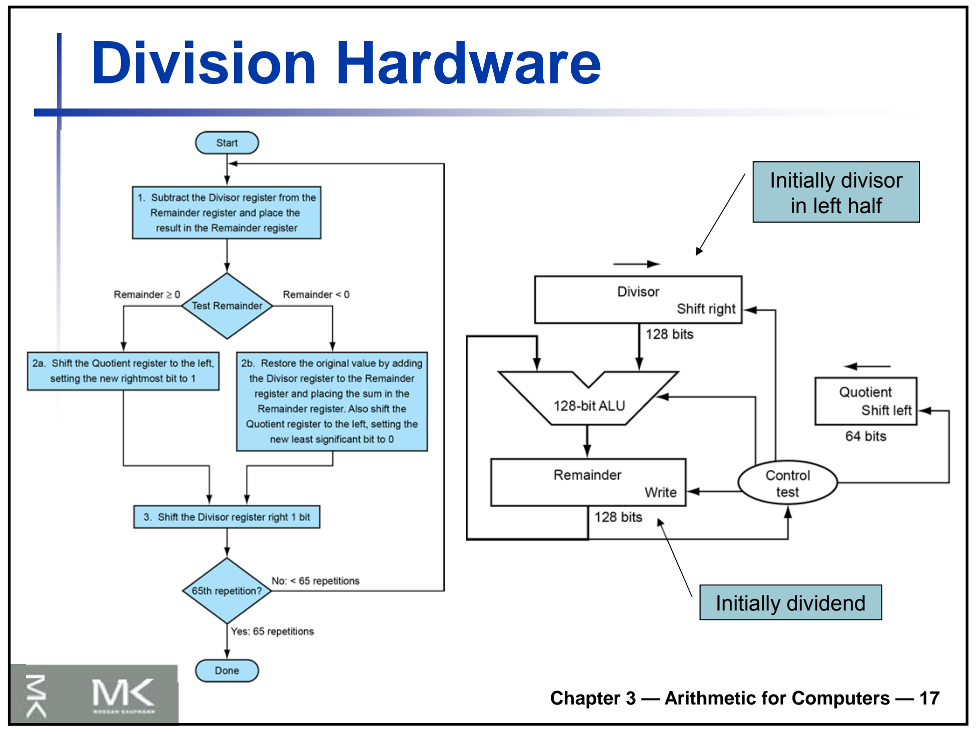

Faster Multiplier
Uses multiple adders
Cost/performance tradeoff
Mplier63 • Mcand Mplier62 • Mcand
Mplier61 • Mcand Mplier60 • Mcand
Mplier3 • Mcand Mplier2 • Mcand
Mplier1 • Mcand Mplier0 • Mcand
64 bits
1 bit
Product127 Product126
Product95..32
Product1 Product0
Can be pipelined
Several multiplication performed in parallel
Chapter 3 — Arithmetic for Computers — 14

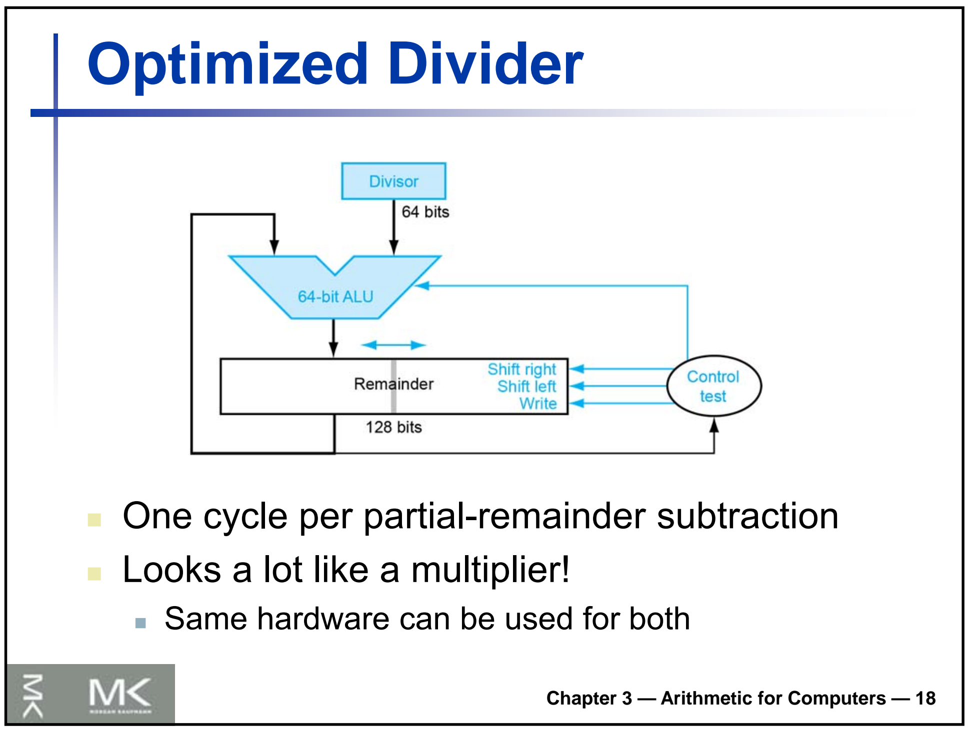
RISC-V Multiplication

- 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

Chapter 3 — Arithmetic for Computers — 15




Division Hardware
Start
1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register
Test Remainder
Remainder ≥ 0
Remainder < 0
2a. Shift the Quotient register to the left, setting the new rightmost bit to 1
2b. Restore the original value by adding the Divisor register to the Remainder register and placing the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0
3. Shift the Divisor register right 1 bit
65th repetition?
No: < 65 repetitions
Yes: 65 repetitions
Done
Initially divisor in left half
Divisor
Shift right
128 bits
128-bit ALU
Quotient
Shift left
64 bits
Remainder
Write
Control test
128 bits
Initially dividend
Chapter 3 — Arithmetic for Computers — 17


Optimized Divider
Divisor
64 bits
64-bit ALU
Remainder
Shift right
Shift left
Write
Control test
128 bits
One cycle per partial-remainder subtraction
Looks a lot like a multiplier!
Same hardware can be used for both
Chapter 3 — Arithmetic for Computers — 18

Faster Division

- Can't use parallel hardware as in multiplier
  - Subtraction is conditional on sign of remainder
- Faster dividers (e.g. SRT devision) generate multiple quotient bits per step
  - Still require multiple steps

RISC-V Division

- Four instructions:
  - div, rem: signed divide, remainder
  - divu, remu: unsigned divide, remainder
- Overflow and division-by-zero don't produce errors
  - Just return defined results
  - Faster for the common case of no error