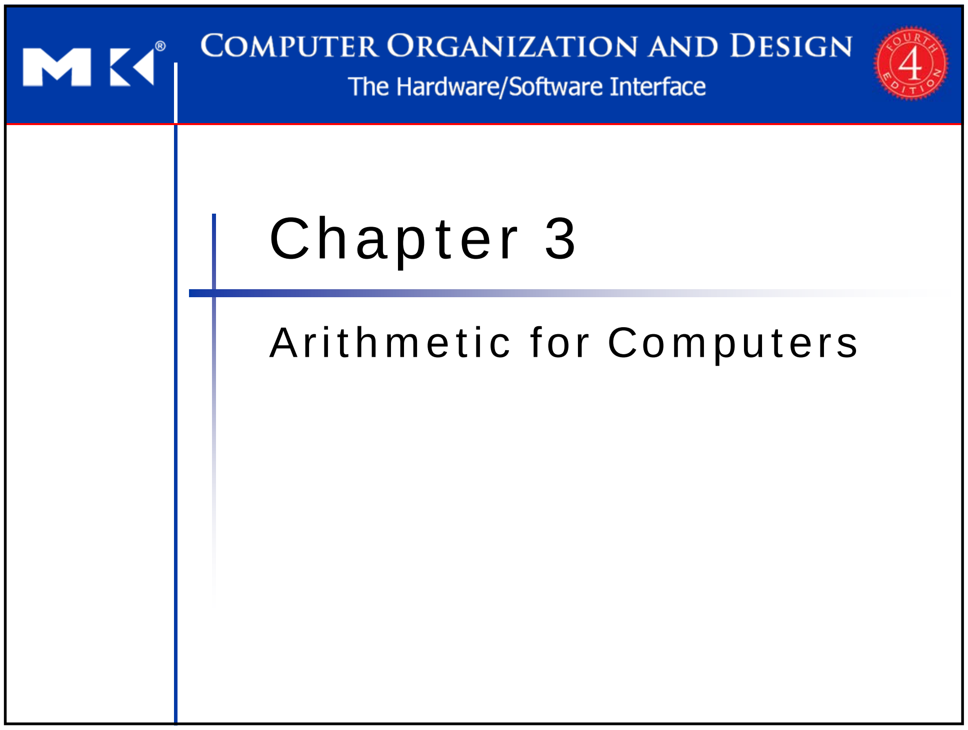
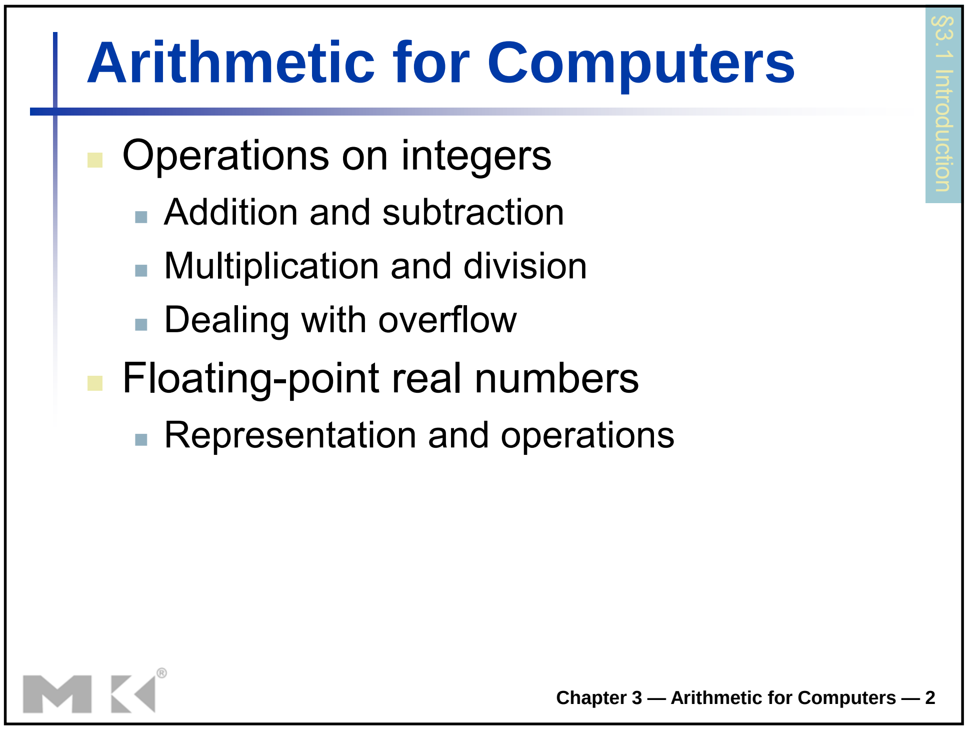


COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface

Chapter 3

Arithmetic for Computers

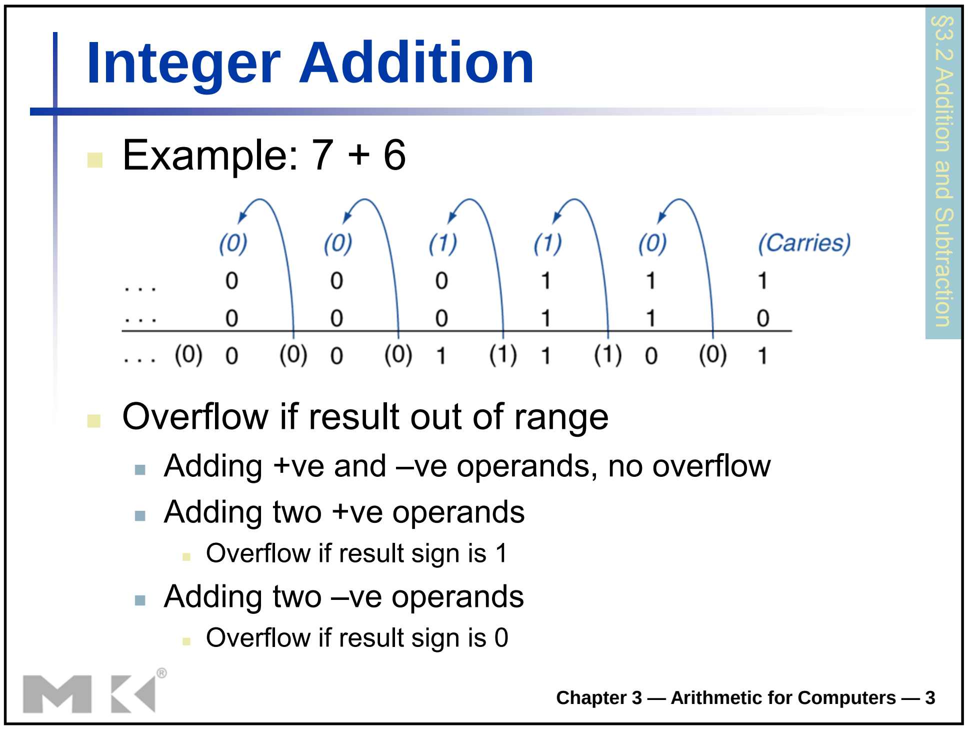


Arithmetic for Computers

§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

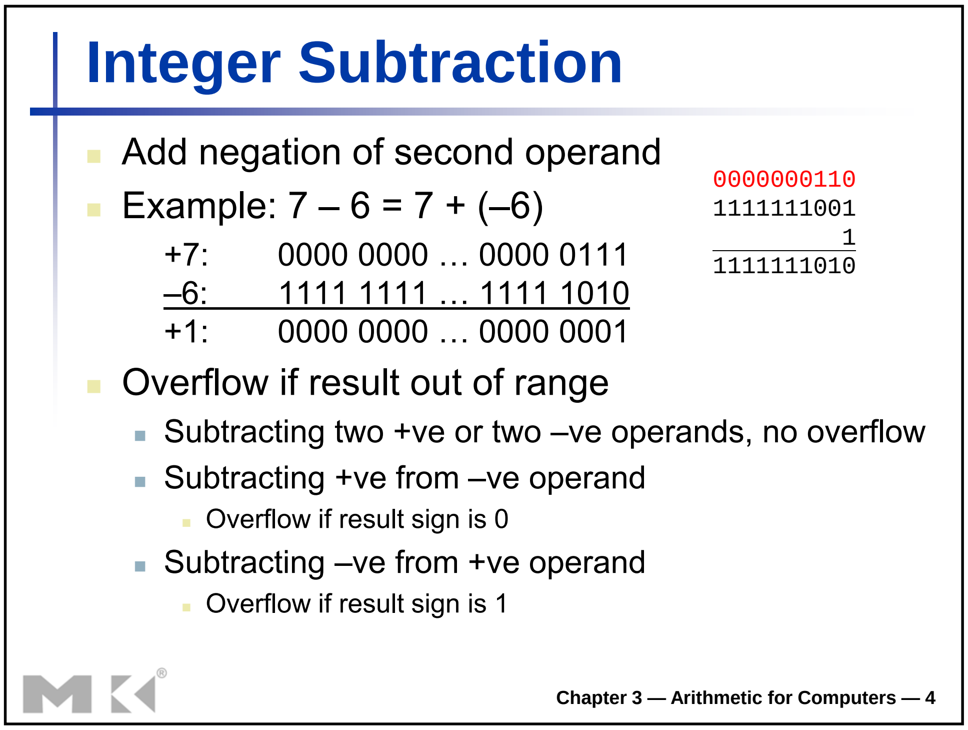
§3.2 Addition and Subtraction

- Example: 7 + 6

```
        (0)    (0)    (1)    (1)    (0)    (Carries)
. . .    0      0      0      1      1      1
. . .    0      0      0      1      1      0
-----------------------------------------------
. . . (0) 0  (0) 0  (0) 1  (1) 1  (1) 0  (0) 1
```

- 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

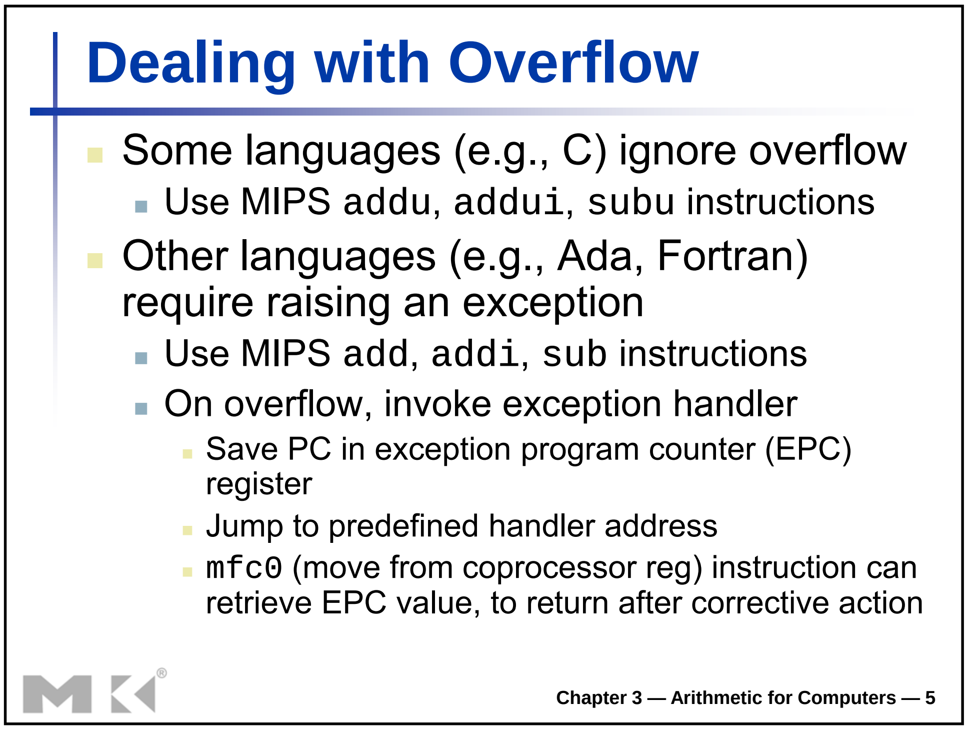
- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

```
+7:   0000 0000 … 0000 0111
–6:   1111 1111 … 1111 1010
+1:   0000 0000 … 0000 0001
```

```
0000000110
1111111001
         1
----------
1111111010
```

- 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

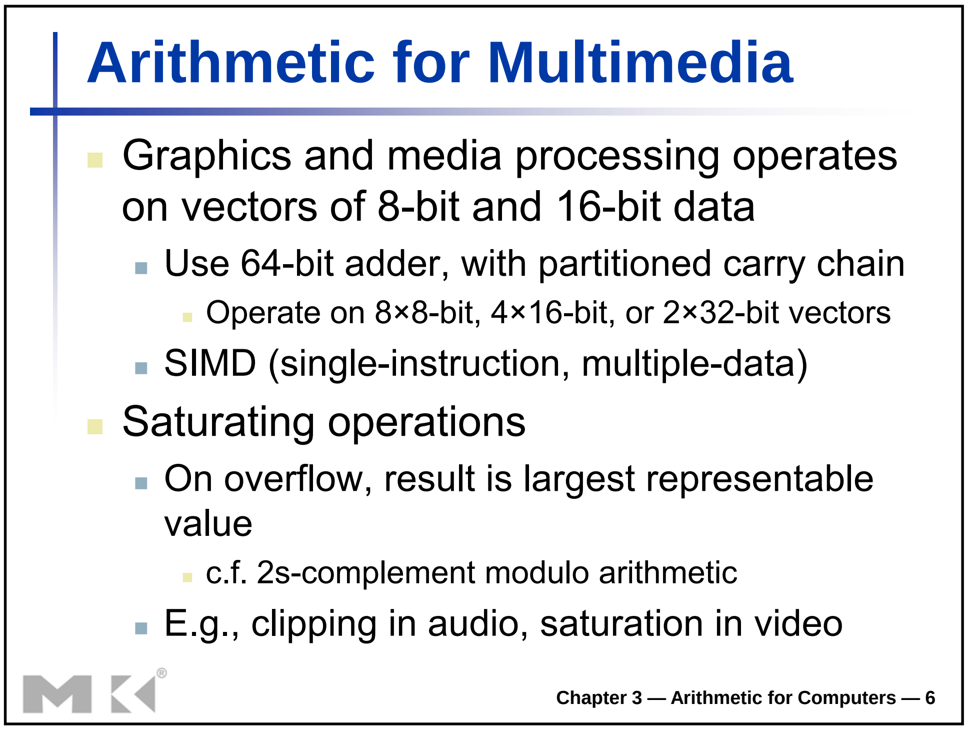
Chapter 3 — Arithmetic for Computers — 4



Dealing with Overflow

- Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui`, `subu` instructions
- Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi`, `sub` instructions
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register
    - Jump to predefined handler address
    - `mfc0` (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action

Chapter 3 — Arithmetic for Computers — 5



Arithmetic for Multimedia

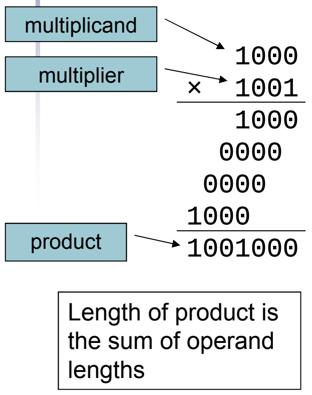
- Graphics and media processing operates on vectors of 8-bit and 16-bit data
  - Use 64-bit adder, with partitioned carry chain
    - Operate on 8×8-bit, 4×16-bit, or 2×32-bit vectors
  - SIMD (single-instruction, multiple-data)
- Saturating operations
  - On overflow, result is largest representable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

Chapter 3 — Arithmetic for Computers — 6

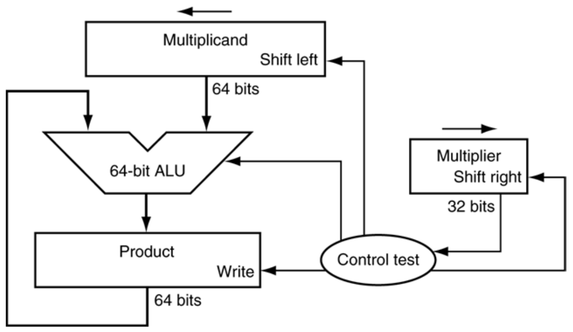
Multiplication

§3.3 Multiplication

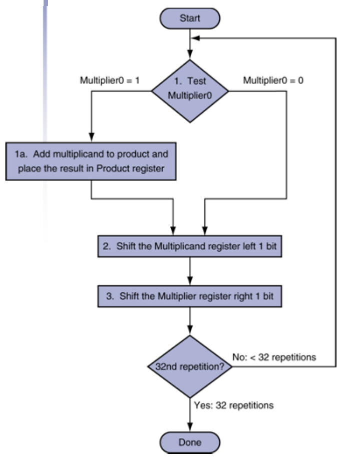
- Start with long-multiplication approach

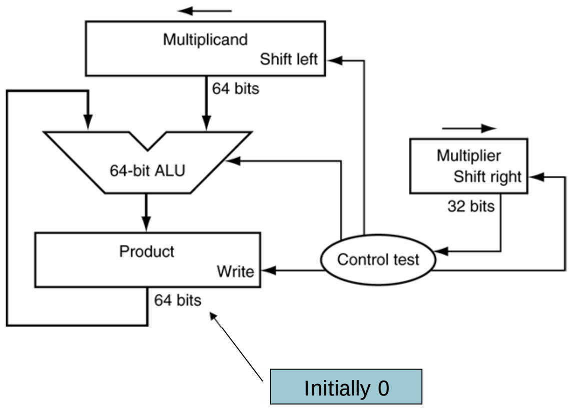


```
      1000
  ×   1001
  --------
      1000
     0000
    0000
   1000
  --------
   1001000
```



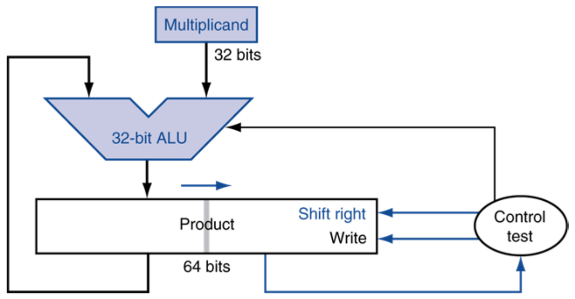
Multiplication Hardware


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: < 32 repetitions
Yes: 32 repetitions
Done



Optimized Multiplier

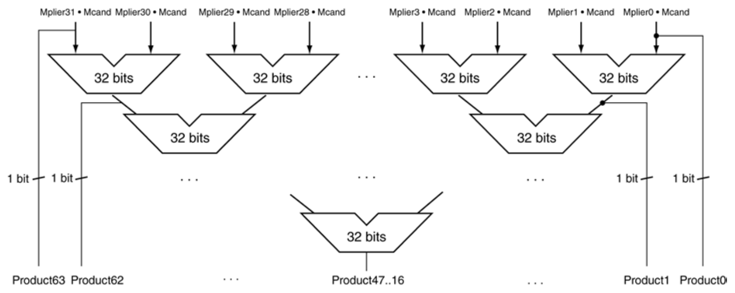
- Perform steps in parallel: add/shift



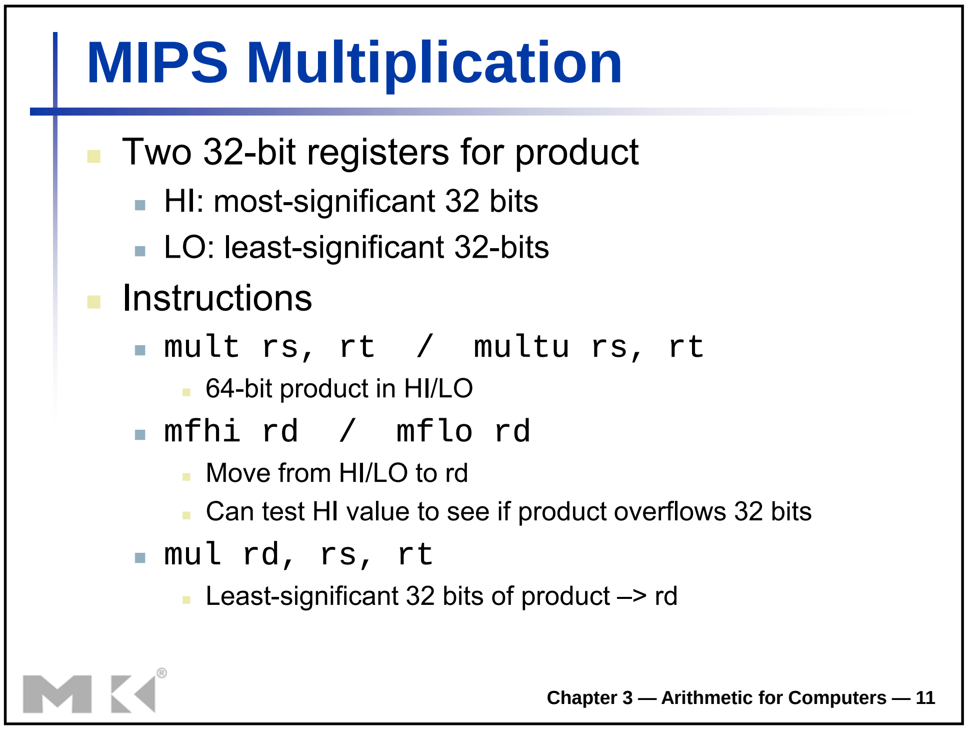
- One cycle per partial-product addition
  - That’s ok, if frequency of multiplications is low

Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



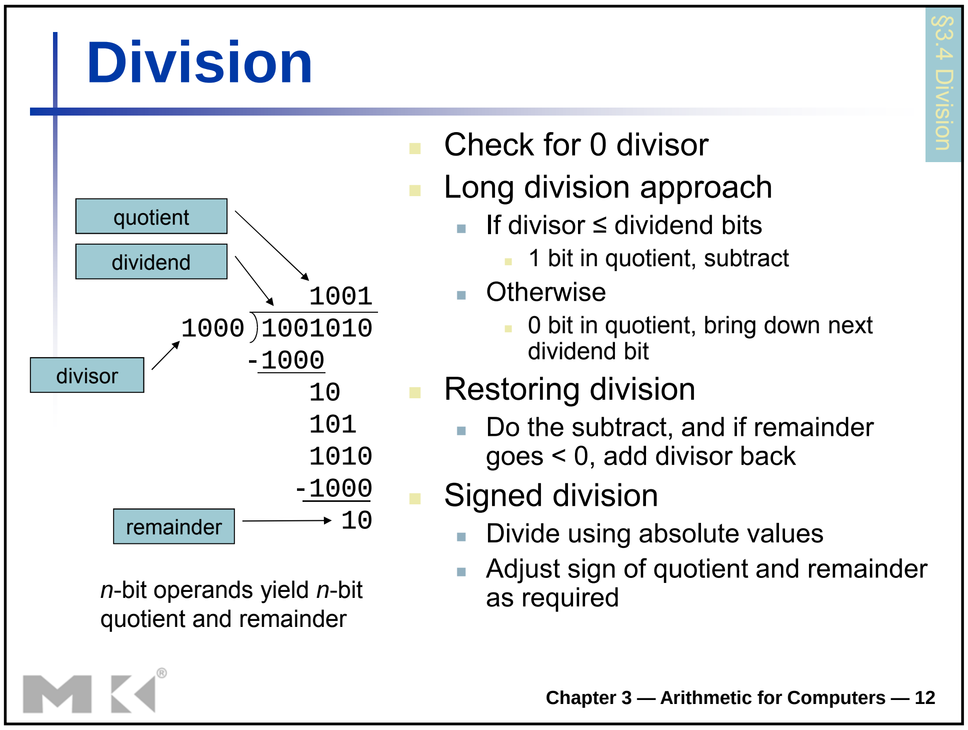
- Can be pipelined
  - Several multiplication performed in parallel



MIPS Multiplication

- Two 32-bit registers for product
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instructions
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd

Chapter 3 — Arithmetic for Computers — 11



Division

§3.4 Division

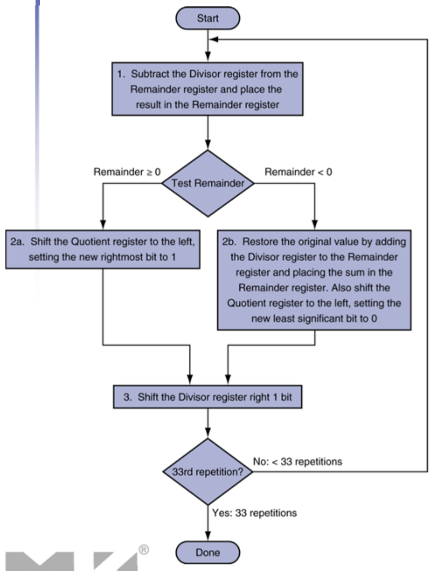
```
quotient          1001
dividend
        1000 )1001010
divisor      -1000
                10
                101
                1010
               -1000
remainder  ->     10
```

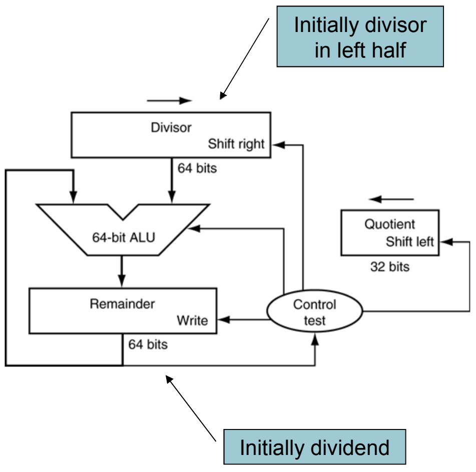
n-bit operands yield *n*-bit quotient and remainder

- Check for 0 divisor
- Long division approach
  - If divisor ≤ dividend bits
    - 1 bit in quotient, subtract
  - Otherwise
    - 0 bit in quotient, bring down next dividend bit
- Restoring division
  - Do the subtract, and if remainder goes < 0, add divisor back
- Signed division
  - Divide using absolute values
  - Adjust sign of quotient and remainder as required

Chapter 3 — Arithmetic for Computers — 12

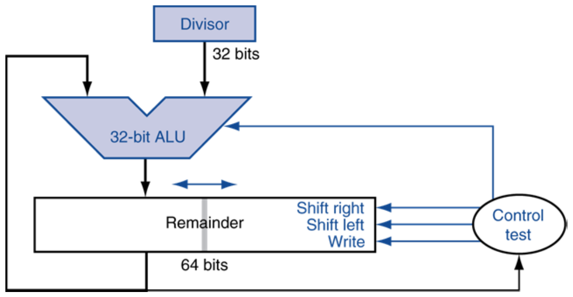
Division Hardware


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
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done



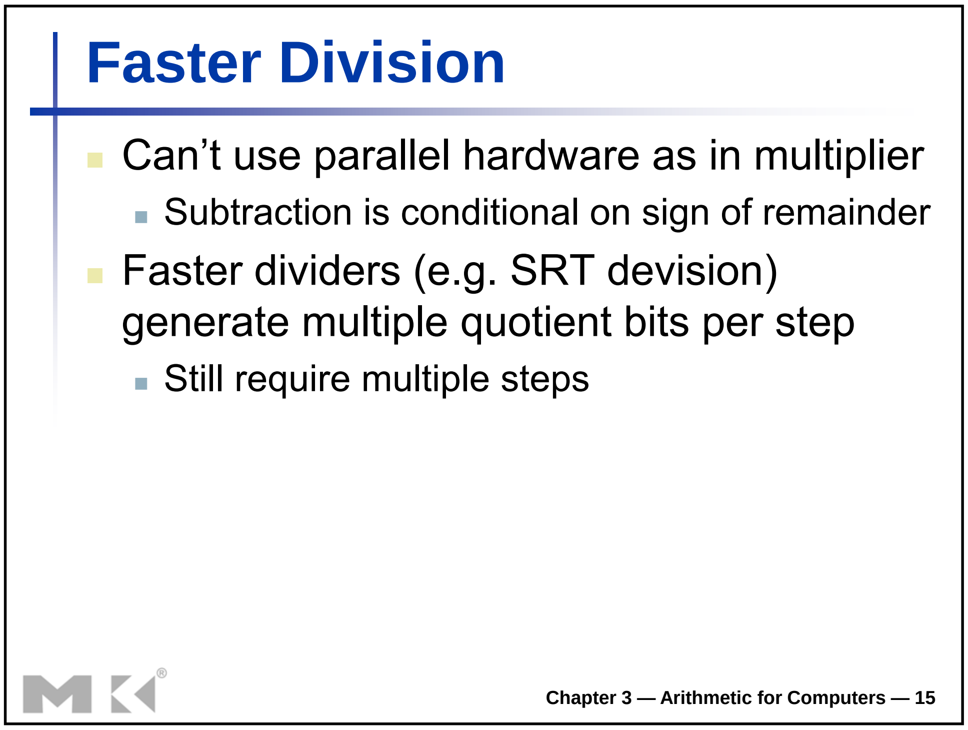
Chapter 3 — Arithmetic for Computers — 13

Optimized Divider



- One cycle per partial-remainder subtraction
- Looks a lot like a multiplier!
  - Same hardware can be used for both

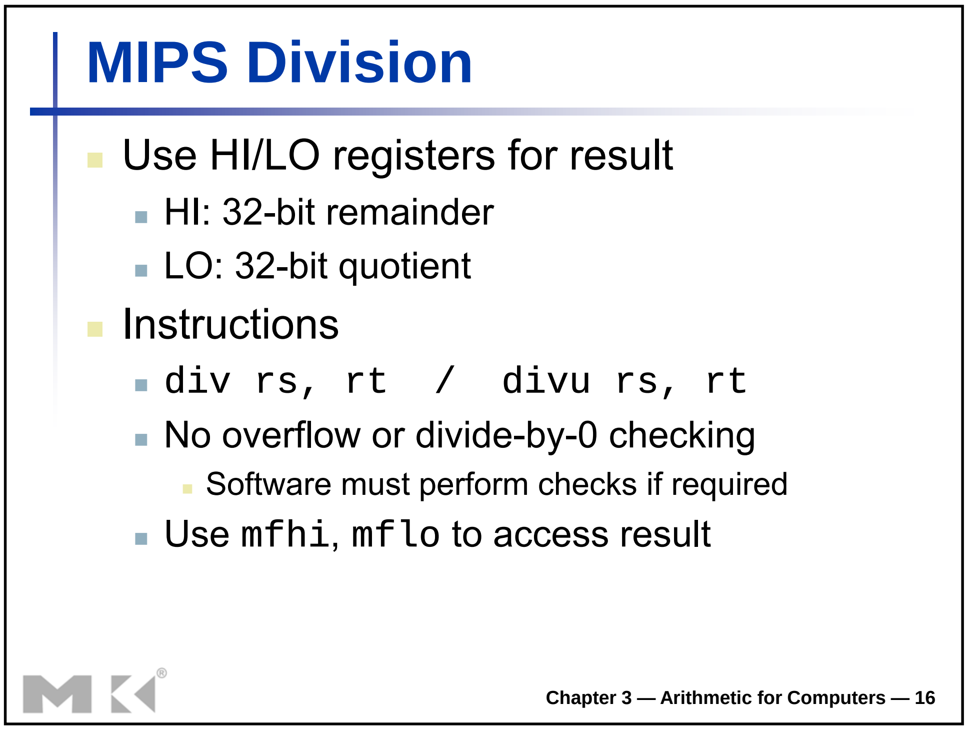
Chapter 3 — Arithmetic for Computers — 14



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

Chapter 3 — Arithmetic for Computers — 15



MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - `div rs, rt  /  divu rs, rt`
  - No overflow or divide-by-0 checking
    - Software must perform checks if required
  - Use `mfhi`, `mflo` to access result

Chapter 3 — Arithmetic for Computers — 16