Chapter 3 Arithmetic for Computers

Chulyun Kim

* This material is based on the lecture slides provided by Morgan Kaufmann

Outline

- Introduction
- Addition and Subtraction
- Multiplication
- Division
- Floating Points
- Concluding Remarks

Arithmetic for Computers

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

§3.1 Introduction

Integer Addition

- 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

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

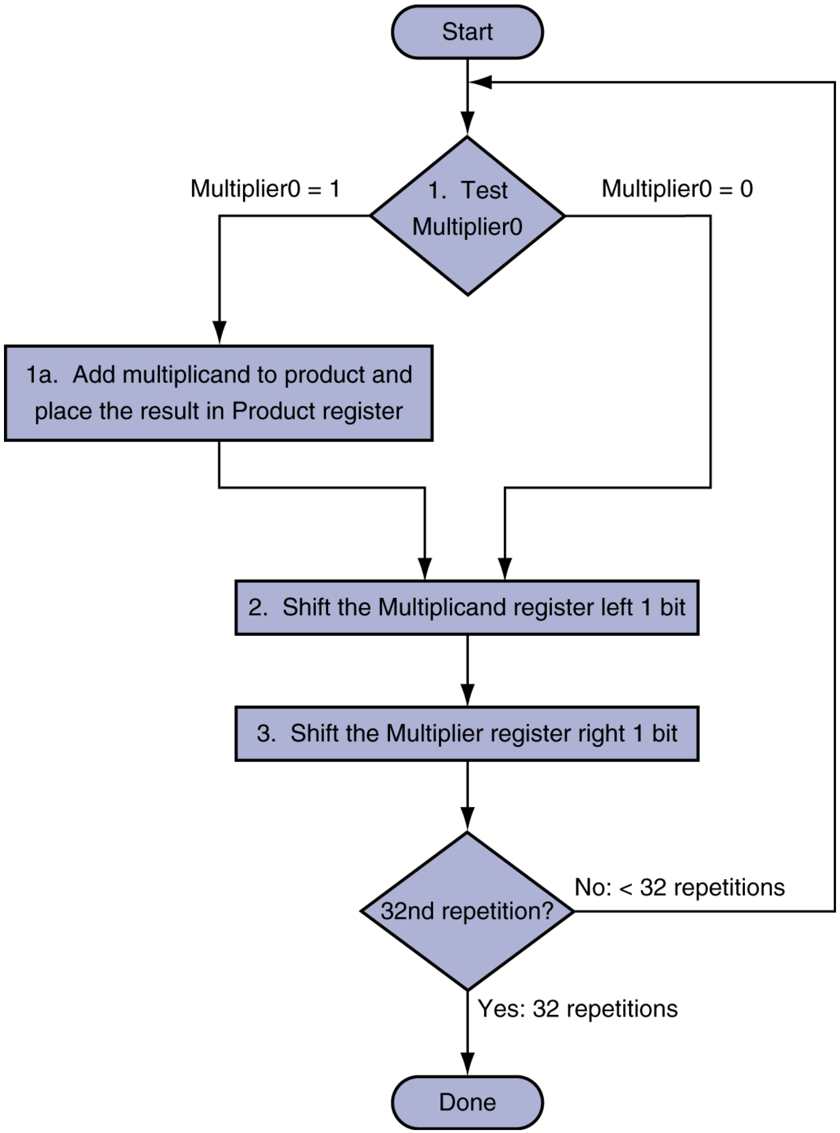
- Start with long-multiplication approach

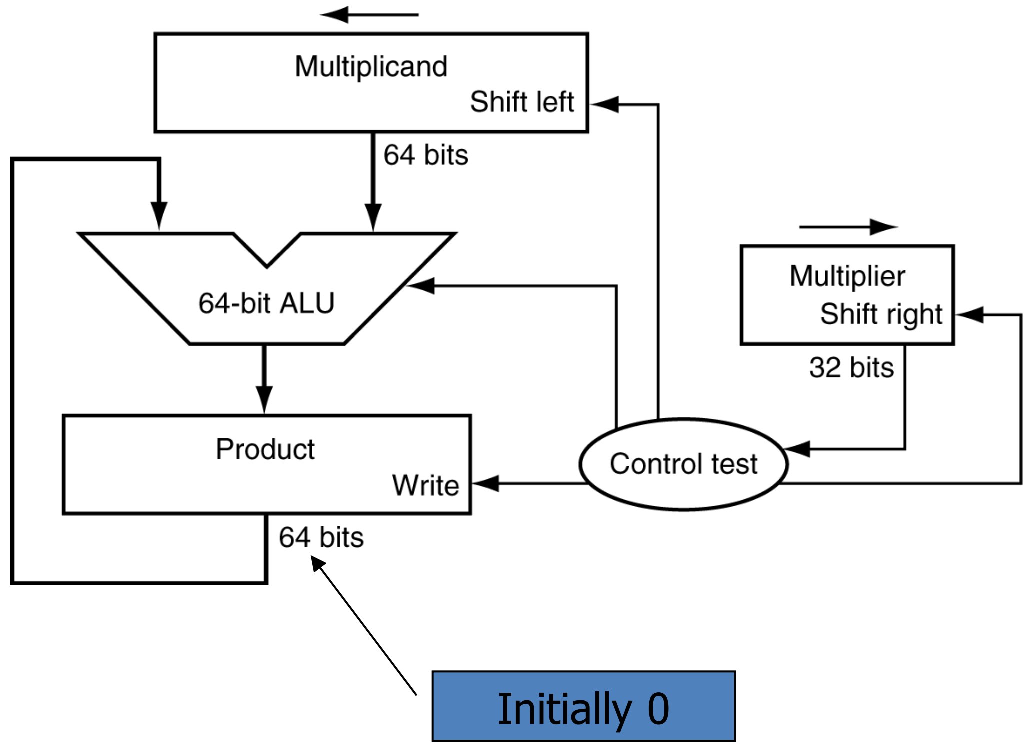
```
multiplicand      1000
multiplier    ×   1001
                 ------
                  1000
                 0000
                0000
               1000
               -------
product        1001000
```

Length of product is the sum of operand lengths

➔ $(2^m-1)*(2^n-1) < 2^{m+n}$

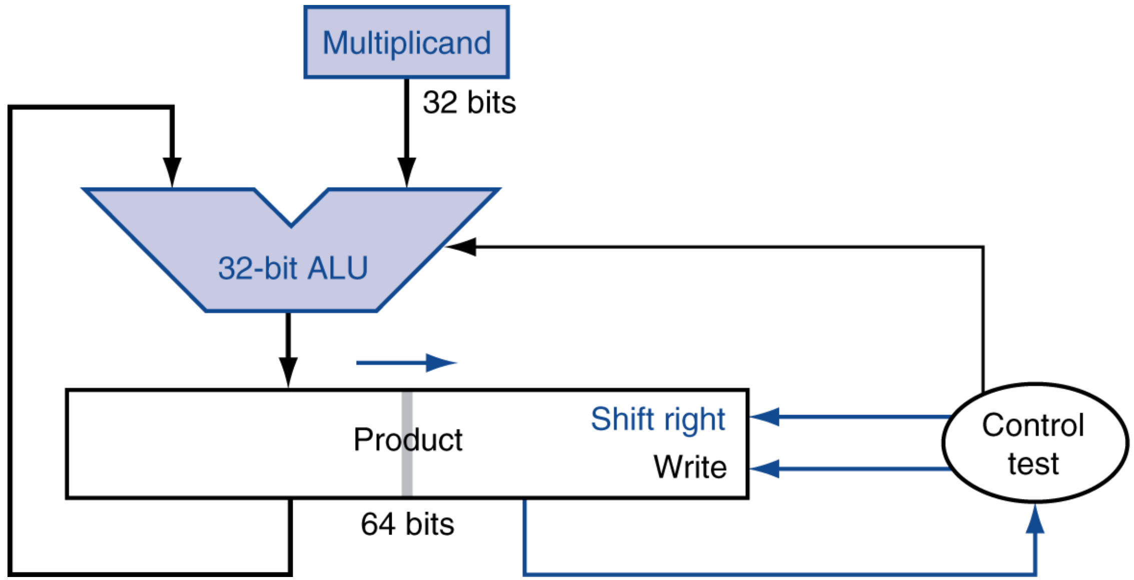
Multiplication Hardware





Optimized Multiplier

- Perform steps in parallel: add/shift

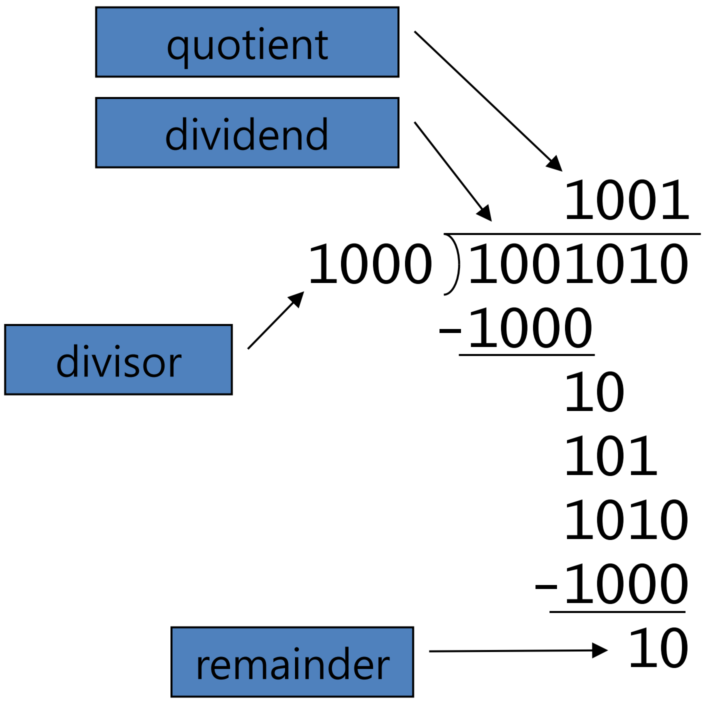


  - One cycle per partial-product addition
    - That's ok, if frequency of multiplications is low

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

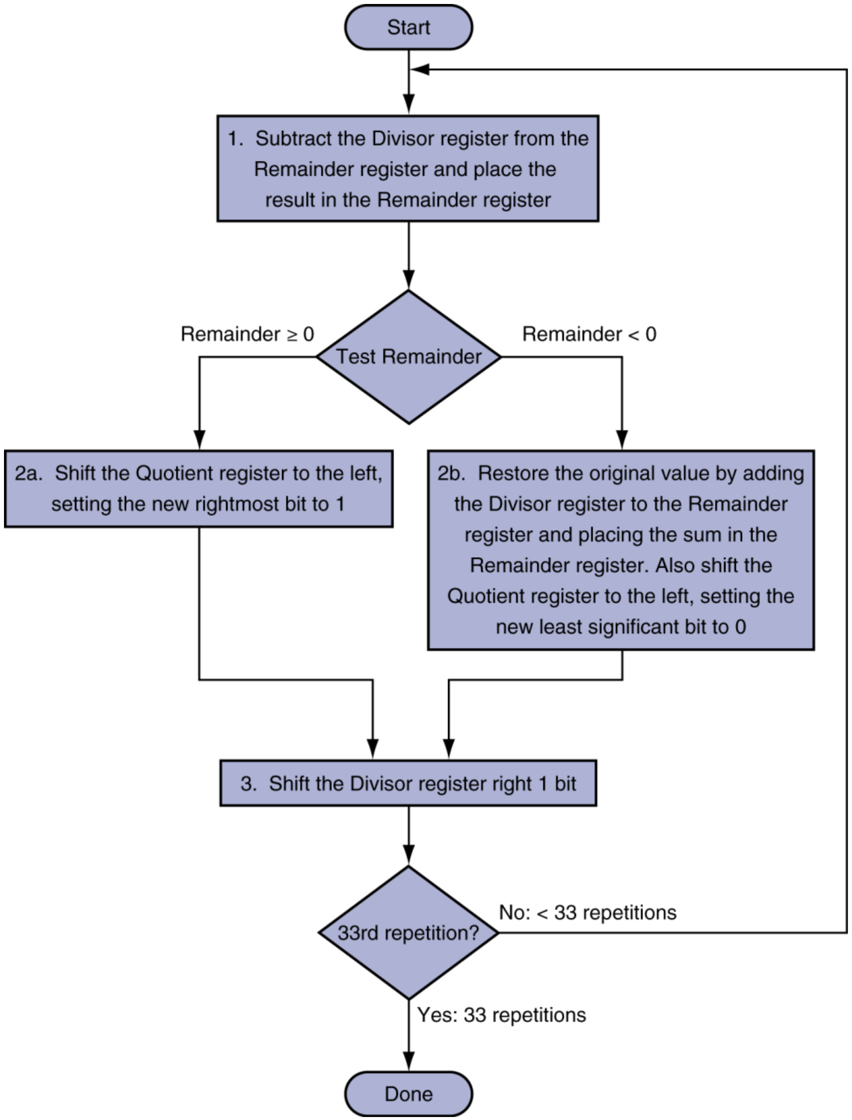
Division

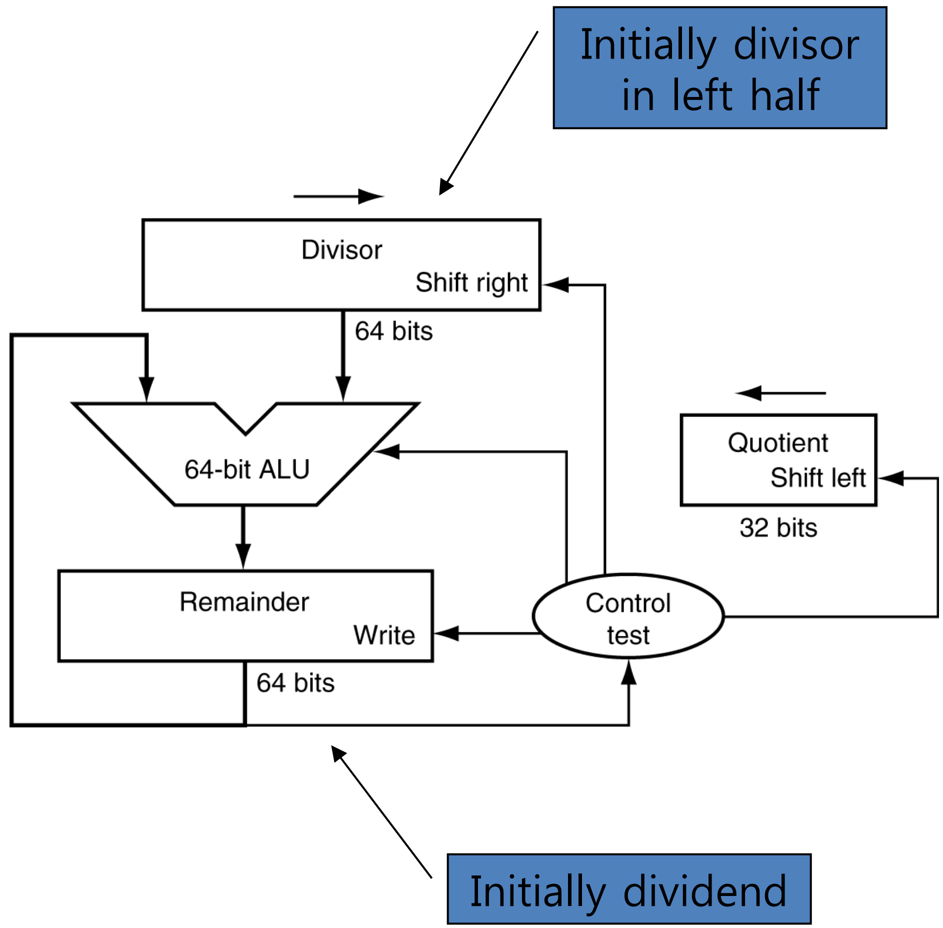


n-bit operands yield *n*-bit quotient and remainder

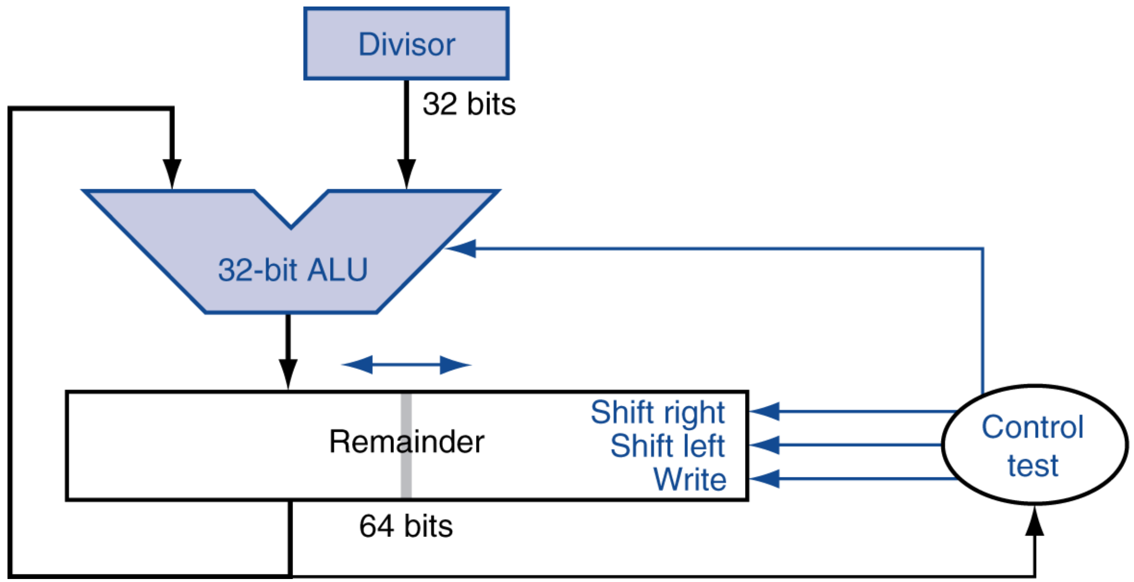
- Check for 0 divisor
- Long division approach
  - If divisor ≤ dividend bits
    - 1 bit in quotient, subtract
  - Otherwise
    - 0 bit in quotient, bring down next dividend bit
- Restoring division
  - Do the subtract, and if remainder goes < 0, add divisor back
- Signed division
  - Divide using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

Division Hardware





Optimized Divider



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

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

Floating Point

- Representation for non-integral numbers
  - Including very small and very large numbers
- Like scientific notation
  - -2.34×10^{56} ← normalized
  - $+0.002 \times 10^{-4}$ ← not normalized
  - $+987.02 \times 10^{9}$ ← not normalized
- In binary
  - $\pm 1.xxxxxxx_2 \times 2^{yyyy}$
- Types `float` and `double` in C

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of repr esentations
  - 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
S	Exponent	Fraction

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

- S: sign bit (0 ⇒ non-negative, 1 ⇒ negative)
- Normalize significand: 1.0 ≤ |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

Floating-Point Example

- Represent –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: $-1 + 127 = 126 = 01111110_2$
    - Double: $-1 + 1023 = 1022 = 01111111110_2$
- Single: 1011111101000…00
- Double: 1011111111101000…00

Floating-Point Example

- What number is represented by the single-precision float

  11000000101000… 00

  – S = 1

  – Fraction = $01000\ldots00_2$

  – Exponent = $10000001_2 = 129$

- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129-127)}$

  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

Interpretation of Data

- 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
- MIPS ISA
  - Core instructions: 54 most frequently used
    - 100% of SPECINT, 97% of SPECFP
  - Other instructions: less frequent