Integer Arithmetic

Jinkyu Jeong (jinkyu@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Where Are We?
Abstraction Layers in Modern Systems

Scope of this course

* Sources: CS 252 lecture notes from Prof. Kubiatowicz (UC Berkeley).
  Coursera lecture notes for HW/SW Interface from Profs. Borriello and Ceze (Univ. of Washington).
Introduction

• Bits are just bits
  – No inherent meaning: conventions define relationship between bits and numbers
  – $n$-bit binary numbers: $0 \sim 2^n - 1$ decimal numbers

• Of course it gets more complicated
  – Numbers are finite (overflow)
  – Negative numbers
  – Fractions and real numbers
  – Precision and accuracy
  – Error propagation, ...
Arithmetic for Computers: An Overview

• Operations on integers
  – Textbook: P&H 3.1-3.4 (for both 4th and 5th Editions)
  – Addition and subtraction
  – Multiplication and division
  – Dealing with overflow

• Floating-point real numbers
  – Textbook: P&H 3.5
  – Representation and operations
Integer Addition & Subtraction
Integer Addition

• Example: $7 + 6$

```
. . . 0 0 0 1 1 1 1
. . . 0 0 0 1 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 \ldots 0000 0111
  
  $-6$: 1111 1111 \ldots 1111 1010
  
  $+1$: 0000 0000 \ldots 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**
Simple Adder ($n$-bit ripple-carry adder)
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
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
Integer Multiplication
Multiplication

• Start with long-multiplication approach

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
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
Integer Division
Division

- Check for 0 divisor
- Long division approach
  - If divisor \( \leq \) 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

\[
\begin{array}{c}
\text{dividend} \\
\hline
1000 & 1001010 \\
-1000 & \hline
10 & 101 \\
1010 & \hline
-1000 & \hline
10 & \hline
\end{array}
\]

\( n \)-bit operands yield \( n \)-bit quotient and remainder
Division Hardware

1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register

Remainder \geq 0

Test Remainder

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

No: \leq 33 repetitions

Yes: 33 repetitions

Done

Initially divisor in left half

Initially dividend

Divisor

Shift right

64 bits

64-bit ALU

Remainder

Write

64 bits

Control test

Quotient

Shift left

32 bits
Optimized Divider

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

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