

CSE 2021 Computer Organization

Chapter 3

Arithmetic for Computers

Arithmetic for Computers

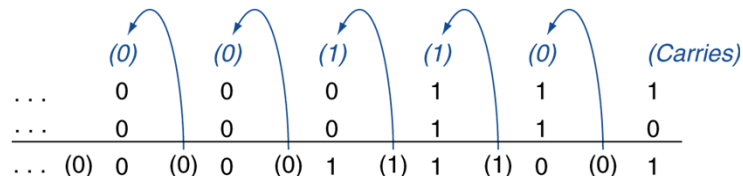
- Operations on integers
 - Addition and subtraction
 - Multiplication and division
 - Dealing with overflow
- Floating-point real numbers
 - Representation and operations

CSE 2021 Computer Organization

Arithmetic Operations on Integers

Integer Addition & Subtraction

- Addition example: $7 + 6$



- Subtraction example: $7 - 6 = 7 + (-6)$

- Add negation of second operand

```

+7:  0000 0000 ... 0000 0111
-6:  1111 1111 ... 1111 1010
+1:  0000 0000 ... 0000 0001

```

2's complement

Addition of Signed Numbers

More examples below are shown for 4-bit 2's complement arithmetic.

$$\begin{array}{r} 1. \quad (+5) \quad 0101 \\ +(+2) \quad +0010 \\ \hline (+7) \quad 0111 \end{array}$$

$$\begin{array}{r} 2. \quad (-5) \quad 1011 \\ +(+2) \quad +0010 \\ \hline (-3) \quad 1101 \end{array}$$

$$\begin{array}{r} 3. \quad (+5) \quad 0101 \\ +(-2) \quad +1110 \\ \hline (+3) \quad 1 \ 0011 \\ \text{ignore the carry} \end{array}$$

$$\begin{array}{r} 4. \quad (-5) \quad 1011 \\ +(-2) \quad +1110 \\ \hline (-7) \quad 1 \ 1001 \\ \text{ignore the carry} \end{array}$$

Overflow

- Example: 7 + 6 (each number in signed 4-bit)

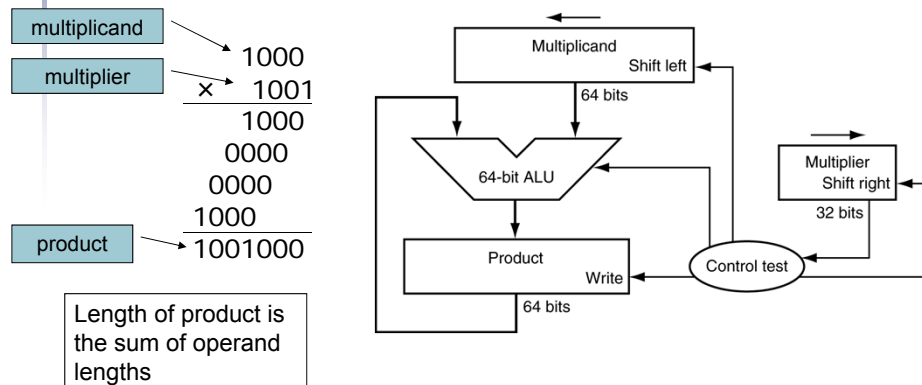
$$\begin{array}{r} + 7: \quad 0111 \\ + 6: \quad 0110 \\ \hline +13: \quad 1101 \rightarrow -3 \\ \text{Overflow} \end{array}$$

- Overflow if result out of range

Operation	Operand A	Operand B	Result Indicating overflow
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0

Multiplication

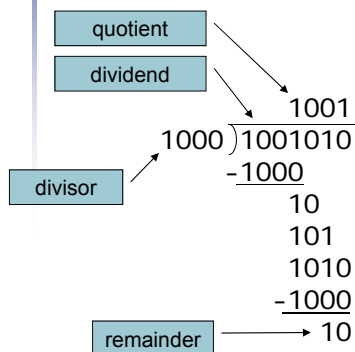
- Start with long-multiplication approach



MIPS Multiplication

- Two 32-bit registers for product
 - HI: most-significant 32 bits
 - LO: least-significant 32-bits
- Instructions
 - `mult 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 \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

MIPS Division

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

CSE 2021 Computer Organization

Floating Point

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
 - $\pm 1.xxxxxx_2 \times 2^{yyyy}$
- Types `float` and `double` in C

Floating Point Standard

- Defined by IEEE Std 754-1985
- Developed in response to divergence of 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

S	Exponent	Fraction
---	----------	----------

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

- 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 ≈ 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: 00000000001
 \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: 11111111110
 \Rightarrow actual exponent = $2046 - 1023 = +1023$
 - Fraction: 111...11 \Rightarrow significand ≈ 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

Activity 1

- Represent $(-0.75)_{10}$ in single and double precision of IEEE 754 binary representation

Activity 2

- What number is represented by the single-precision float

11000000101000...00

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
 - 1.0015×10^2
- 4. Round and renormalize if necessary
 - 1.002×10^2

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

FP Instructions in MIPS

- FP hardware is coprocessor 1
 - Adjunct processor that extends the ISA
- Separate FP registers
 - 32 single-precision: \$f0, \$f1, ... \$f31
 - Paired for double-precision: \$f0/\$f1, \$f2/\$f3, ...
- 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
 - lwc1, ldc1, swc1, sdc1
 - e.g., ldc1 \$f8, 32(\$sp)

FP Instructions in MIPS

- Single-precision arithmetic
 - add. s, sub. s, mul. s, div. s
 - e.g., add. s \$f0, \$f1, \$f6
- Double-precision arithmetic
 - add. d, sub. d, mul. d, div. d
 - e.g., mul. d \$f4, \$f4, \$f6
- Single- and double-precision comparison
 - c. xx. s, c. xx. d (xx is eq, lt, le, ...)
 - Sets or clears FP condition-code bit
 - e.g. c. lt. s \$f3, \$f4
- Branch on FP condition code true or false
 - bc1t, bc1f
 - e.g., bc1t TargetLabel

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

Acknowledgement

- The slides are adapted from Computer Organization and Design, 4th Edition, by David A. Patterson and John L. Hennessy, 2008, published by MK (Elsevier)