

Divide: Paper & Pencil

Divisor 1000	$ \begin{array}{r} 1001 \\ 1001010 \\ \underline{-1000} \\ 10 \\ 101 \\ 1010 \\ \underline{-1000} \\ 10 \end{array} $	Quotient Dividend Remainder
--------------	--	---

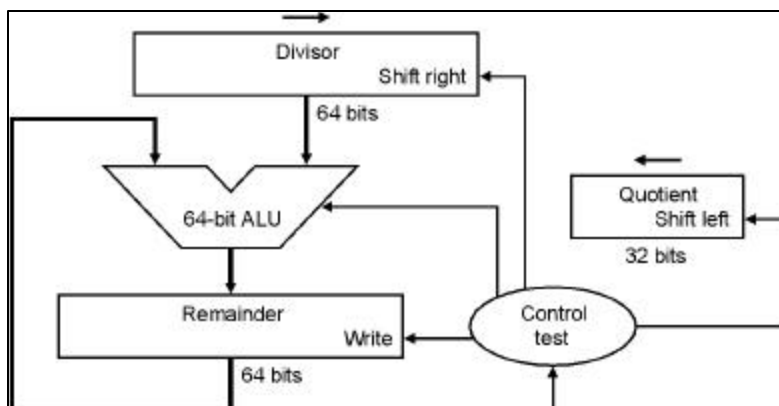
- See how big a number can be subtracted, creating quotient bit on each step
 - Binary => 1 * divisor or 0 * divisor
- Dividend = Quotient x Divisor + Remainder

CSE 141

Allan Snively

DIVIDE HARDWARE Version 1

- 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg



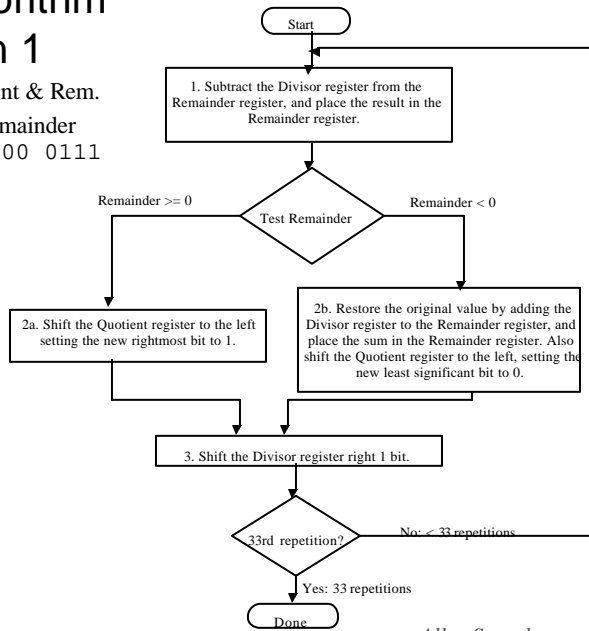
CSE 141

Allan Snively

Divide Algorithm Version 1

- Takes $n+1$ steps for n -bit Quotient & Rem.

Quotient	Divisor	Remainder
0000	0011 0000	0000 0111

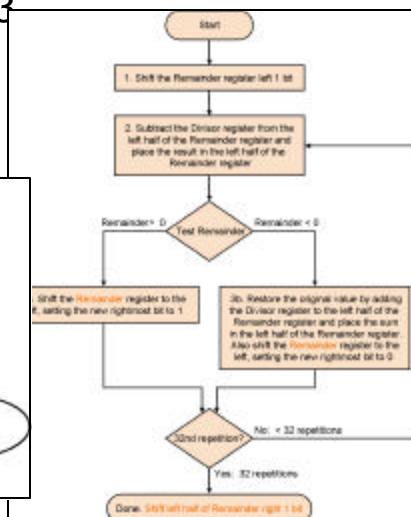
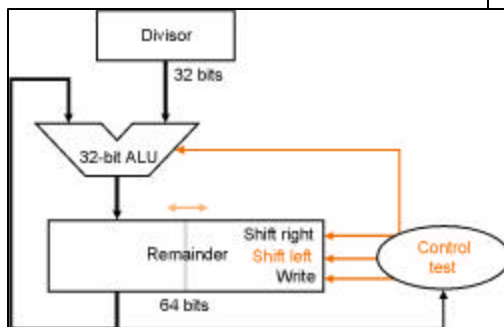


CSE 141

Allan Snively

DIVIDE HARDWARE Version 3

- 32-bit Divisor reg, 32-bit ALU, 64-bit Remainder reg, (Q-bit Quotient reg)



CSE 141

Allan Snively

Observations on Divide Version 3

- Same Hardware as Multiply: just need ALU to add or subtract, and 63-bit register to shift left or shift right
- Hi and Lo registers in MIPS combine to act as 64-bit register for multiply and divide
- Signed Divides: Simplest is to remember signs, make positive, and complement quotient and remainder if necessary
 - Note: Dividend and Remainder must have same sign
 - Note: Quotient negated if Divisor sign & Dividend sign disagree

So Far

- Can do logical, add, subtract, multiply, divide, ...
- But.....
 - what about fractions?
 - what about really large numbers?

Binary Fractions

$$1011_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$$

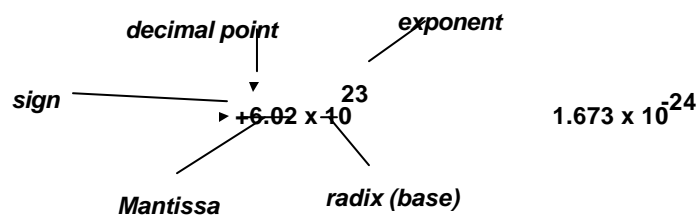
so...

$$101.011_2 = 1x2^2 + 0x2^1 + 1x2^0 + 0x2^{-1} + 1x2^{-2} + 1x2^{-3}$$

e.g.,

$$.75 = 3/4 = 3/2^2 = 1/2 + 1/4 = .11$$

Recall Scientific Notation

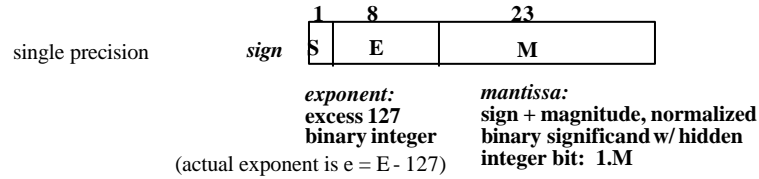


Issues:

- Arithmetic (+, -, *, /)
- Representation, Normal form
- Range and Precision
- Rounding
- Exceptions (e.g., divide by zero, overflow, underflow)
- Errors
- Properties (negation, inversion, if $A = B$ then $A - B = 0$)

Floating-Point Numbers

Representation of floating point numbers in IEEE 754 standard:



$$N = (-1)^S 2^{E-127} (1.M) \quad 0 < E < 255$$

$$\begin{aligned}
 0 &= 0\ 00000000\ 0 \dots 0 & -1.5 &= 1\ 01111111\ 10 \dots 0 \\
 325 &= 101000101\ X\ 2^8 = 1.01000101\ X\ 2^8 \\
 &= 0\ 10000111\ 0100010100000000000000 \\
 .02 &= .0011001101100\dots\ X\ 2^3 = 1.1001101100\dots\ X\ 2^{-3} \\
 &= 0\ 01111100\ 1001101100\dots
 \end{aligned}$$

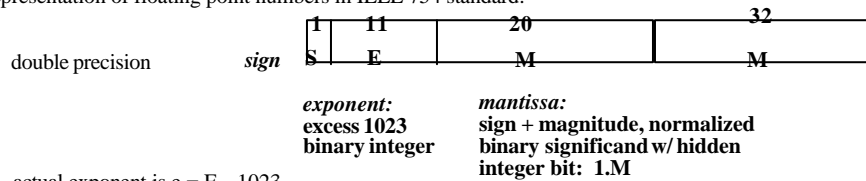
- range of about 2×10^{-38} to 2×10^{38}
- always normalized (so always leading 1, thus never shown)
- special representation of 0 (E = 00000000) (why?)
- can do integer compare for greater-than, sign

CSE 141

Allan Snively

Double Precision Floating Point

Representation of floating point numbers in IEEE 754 standard:



$$N = (-1)^S 2^{E-1023} (1.M) \quad 0 < E < 2048$$

- 52 (+1) bit mantissa
- range of about 2×10^{-308} to 2×10^{308}

CSE 141

Allan Snively

Floating Point Addition

- How do you add in scientific notation?

$$9.962 \times 10^4 + 5.231 \times 10^2$$

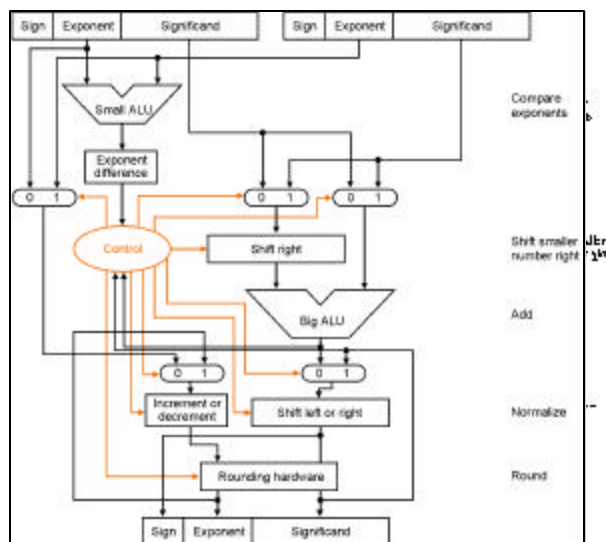
- Basic Algorithm

1. Align
2. Add
3. Normalize
4. Round

CSE 141

Allan Snively

FP Addition Hardware



CSE 141

Allan Snively

Floating Point Multiplication

- How do you multiply in scientific notation?

$$(9.9 \times 10^4)(5.2 \times 10^2) = 5.148 \times 10^7$$

- Basic Algorithm

1. Add exponents
2. Multiply
3. Normalize
4. Round
5. Set Sign

FP Accuracy

- Extremely important in scientific calculations
- Very tiny errors can accumulate over time
- IEEE 754 FP standard has four rounding modes
 - always round up (toward $+\infty$)
 - always round down (toward $-\infty$)
 - truncate
 - round to nearest
 - => in case of tie, round to nearest even
- Requires extra bits in intermediate representations

Extra Bits for FP Accuracy

- *Guard bits* -- bits to the right of the least significant bit of the significand computed for use in normalization (could become significant at that point) and rounding.
- IEEE 754 has three extra bits and calls them *guard*, *round*, and *sticky*.

Key Points

- Multiplication and division take much longer than addition, requiring multiple addition steps.
- Floating Point extends the range of numbers that can be represented, at the expense of precision (accuracy).
- FP operations are very similar to integer, but with pre- and post-processing.
- Rounding implementation is critical to accuracy over time.