

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



Chapter 3 — Arithmetic for Computers — 28

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



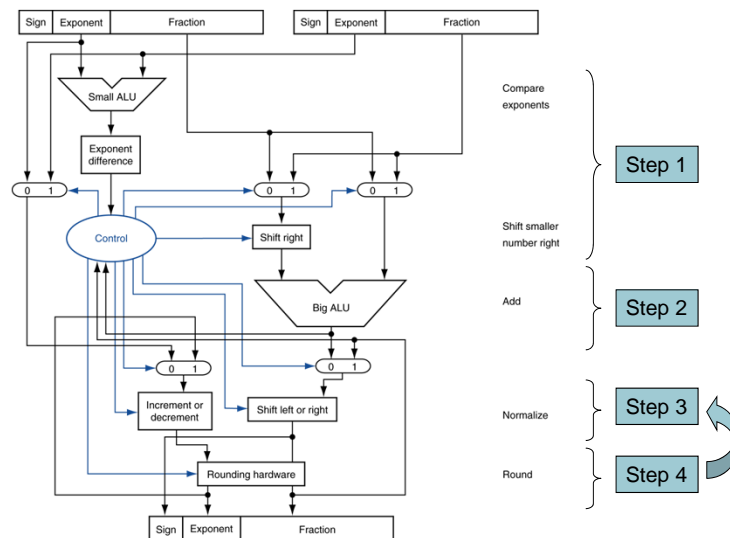
Chapter 3 — Arithmetic for Computers — 29

FP Adder Hardware

- Much more complex than integer adder
- Doing it in one clock cycle would take too long
 - Much longer than integer operations
 - Slower clock would penalize all instructions
- FP adder usually takes several cycles
 - Can be pipelined



FP Adder Hardware



Floating-Point Multiplication

- Consider a 4-digit decimal example
 - $1.110 \times 10^{10} \times 9.200 \times 10^{-5}$
- 1. Add exponents
 - For biased exponents, subtract bias from sum
 - New exponent = $10 + -5 = 5$
- 2. Multiply significands
 - $1.110 \times 9.200 = 10.212 \Rightarrow 10.212 \times 10^5$
- 3. Normalize result & check for over/underflow
 - 1.0212×10^6
- 4. Round and renormalize if necessary
 - 1.021×10^6
- 5. Determine sign of result from signs of operands
 - $+1.021 \times 10^6$



Chapter 3 — Arithmetic for Computers — 32

Floating-Point Multiplication

- Now consider a 4-digit binary example
 - $1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2}$ (0.5×-0.4375)
- 1. Add exponents
 - Unbiased: $-1 + -2 = -3$
 - Biased: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- 2. Multiply significands
 - $1.000_2 \times 1.110_2 = 1.1102 \Rightarrow 1.110_2 \times 2^{-3}$
- 3. Normalize result & check for over/underflow
 - $1.110_2 \times 2^{-3}$ (no change) with no over/underflow
- 4. Round and renormalize if necessary
 - $1.110_2 \times 2^{-3}$ (no change)
- 5. Determine sign: +ve \times -ve \Rightarrow -ve
 - $-1.110_2 \times 2^{-3} = -0.21875$



Chapter 3 — Arithmetic for Computers — 33

FP Arithmetic Hardware

- FP multiplier is of similar complexity to FP adder
 - But uses a multiplier for significands instead of an adder
- FP arithmetic hardware usually does
 - Addition, subtraction, multiplication, division, reciprocal, square-root
 - FP \leftrightarrow integer conversion
- Operations usually takes several cycles
 - Can be pipelined



Chapter 3 — Arithmetic for Computers — 34

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, ...
 - Release 2 of MIPS ISA supports 32 \times 64-bit FP reg's
- 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)



Chapter 3 — Arithmetic for Computers — 35

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



FP Example: °F to °C

- C code:


```
float f2c (float fahr) {
    return ((5.0/9.0)*(fahr - 32.0));
}
```

 - fahr in \$f12, result in \$f0, literals in global memory space
- Compiled MIPS code:


```
f2c: lwc1   $f16, const5($gp)
     lwc2   $f18, const9($gp)
     div.s  $f16, $f16, $f18
     lwc1   $f18, const32($gp)
     sub.s  $f18, $f12, $f18
     mul.s  $f0, $f16, $f18
     jr     $ra
```

