3.5 Floating Point Numbers

Floating Point Numbers

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

Floating Point Standard

- Defined by IEEE Std 754
- 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)
- Exponent: excess representation: actual exponent + Bias
  - Ensures exponent is unsigned
  - Single Precision: Bias = 127; Double Precision: Bias = 1023
- 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

Floating-Point Example

- 다음의 2진수가 표현하는 single-precision float number 는?

  11000000101000…00
  - S = 1
  - Fraction = $01000\ldots00_2$
  - Exponent = $10000001_2 = 01111111_2 + 2$
- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$

  $= (-1) \times 1.01 \times 2^2$

  $= -5.0$

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
  - $\pm 1.0 \times 2^{-126} = (2^{10})^{-12} \times 2^{-6} \approx (10^{3})^{-12} \times 1/64$
    $\approx 10^{-36} \times 10^{-2} \approx \pm\, 2.0 \times 10^{-38}$
- Largest value
  - exponent: 11111110
    $\Rightarrow$ actual exponent = 254 – 127 = +127
  - Fraction: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+127} \approx 2 \text{ x } (2^{10})^{12} \times 2^{7} \approx 2 \text{ x } (10^{3})^{12} \times 127$
    $\approx 2 \text{ x } 10^{36} \times 10^{2} \approx \pm 2.0 \times 10^{+38}$

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

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
  - $\pm 1.0 \times 2^{-1022} \approx \pm 2.0 \times 10^{-308}$
- Largest value
  - Exponent: 11111111110
    $\Rightarrow$ actual exponent = 2046 – 1023 = +1023
  - Fraction: 111…11 $\Rightarrow$ significand ≈ 2.0
  - $\pm 2.0 \times 2^{+1023} \approx \pm 2.0 \times 10^{+308}$

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

Comparison of floating point numbers

- **Comparison of floating point numbers:**

  1. **Compare signs**
  2. **Compare remaining part as an unsigned integer**

  – **`Example : 0.75 vs. 3.45`**

  `(How do we compare unsigned integers?)`

Example : Decimal → Binary

- **0.75**
  - **decimal: .75 = -3/4 = $-3/2^2$**
  - **binary: .11 = -1.1×2^{-1}**
  - **floating point: exponent = 126 = 01111110**

  - **IEEE single precision:**

`0 01111110 100 0000 0000 0000 0000 0000`

- **3.45**

= $(-1)^0 \times (1.10111001100110 0\ldots) \times 2^1$

`0 10000000 101 1100 1100 1100 1100 1101`

Denormal Numbers

- Exponent = 000...0 $\Rightarrow$ hidden bit is 0

$$x = (-1)^S \times (0 + \text{Fraction}) \times 2^{-\text{Bias}}$$

- Smaller than normal numbers
  - allow for gradual underflow, with diminishing precision
- Denormal number with fraction = 000...0

$$x = (-1)^S \times (0 + 0) \times 2^{-\text{Bias}} = \pm 0.0$$

Two representations of 0.0!

Infinities and NaNs

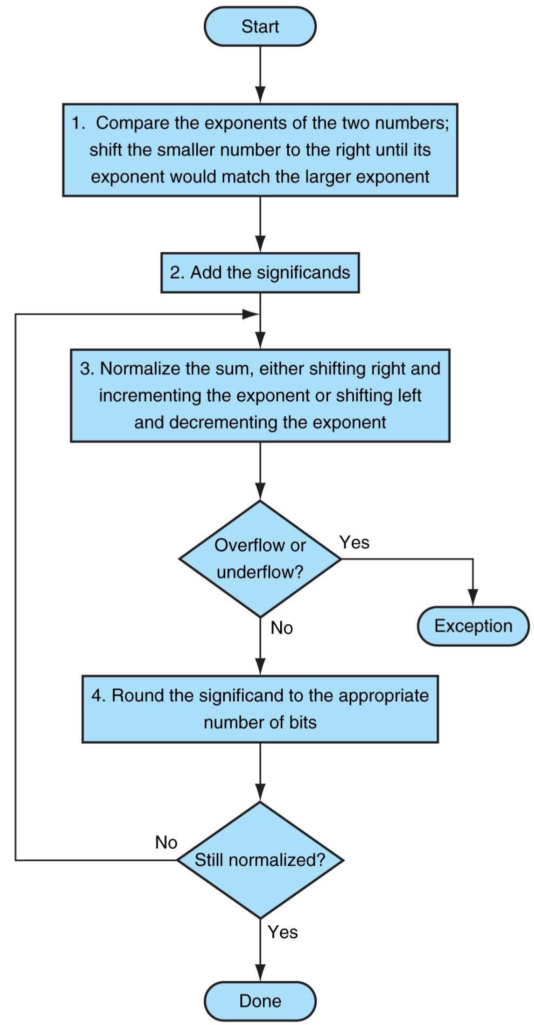
- Exponent = 111…1, Fraction = 000…0
  - ±Infinity
  - Can be used in subsequent calculations, avoiding need for overflow check
- Exponent = 111…1, Fraction ≠ 000…0
  - Not-a-Number (NaN)
  - Indicates illegal or undefined result
    - e.g., 0.0 / 0.0
  - Can be used in subsequent calculations

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)

0 01111110 0..(x23) 1 01111101 110.. (x21)

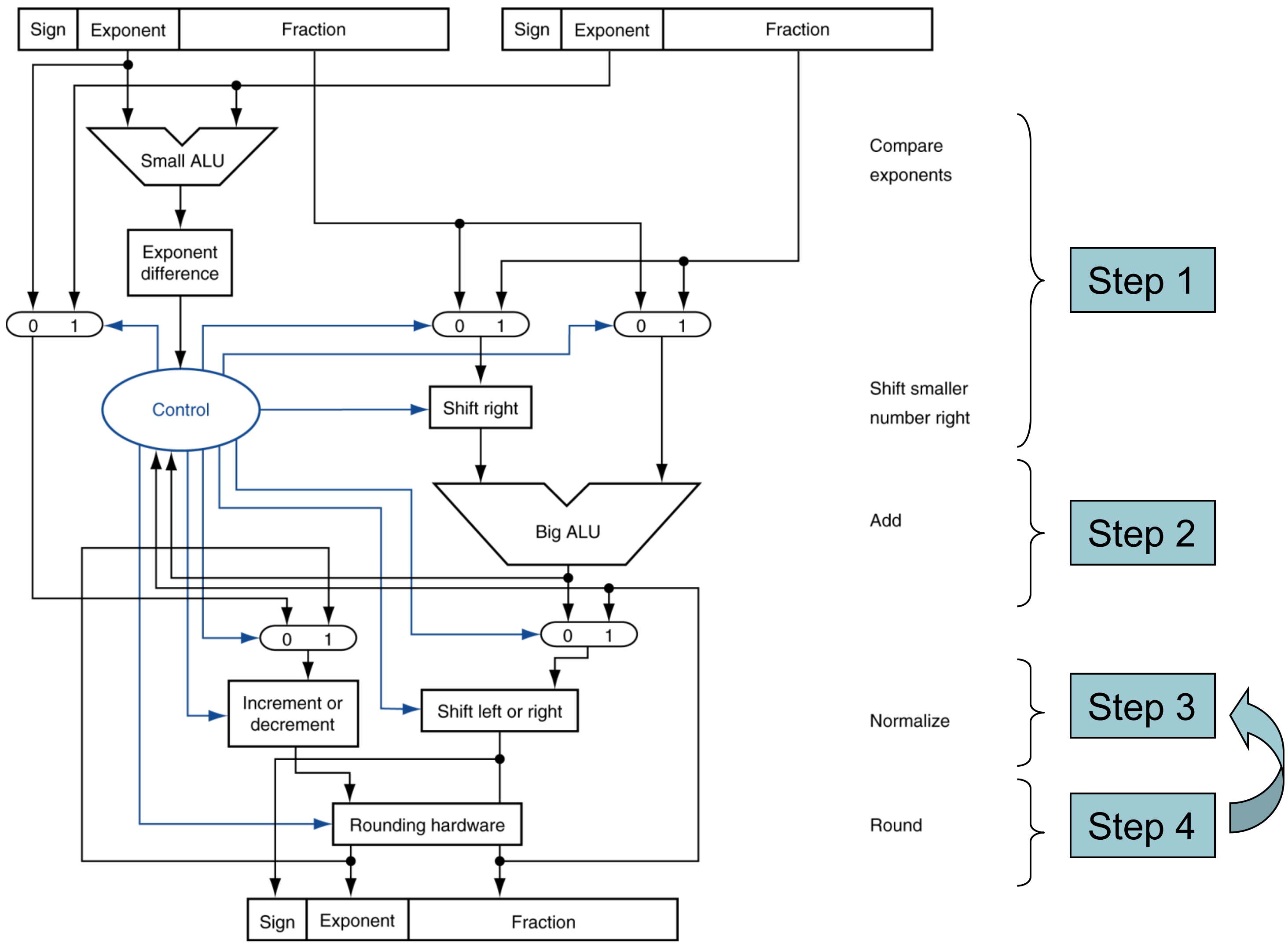
- 1. Align binary points
  - Shift number with smaller exponent
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1}$
- 2. Add significands
  - $1.000_2 \times 2^{-1} + -0.111_2 \times 2^{-1} = 0.001_2 \times 2^{-1}$
- 3. Normalize result & check for over/underflow
  - $1.000_2 \times 2^{-4}$, with no over/underflow
- 4. Round and renormalize if necessary
  - $1.000_2 \times 2^{-4}$ (no change) = 0.0625



FP 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



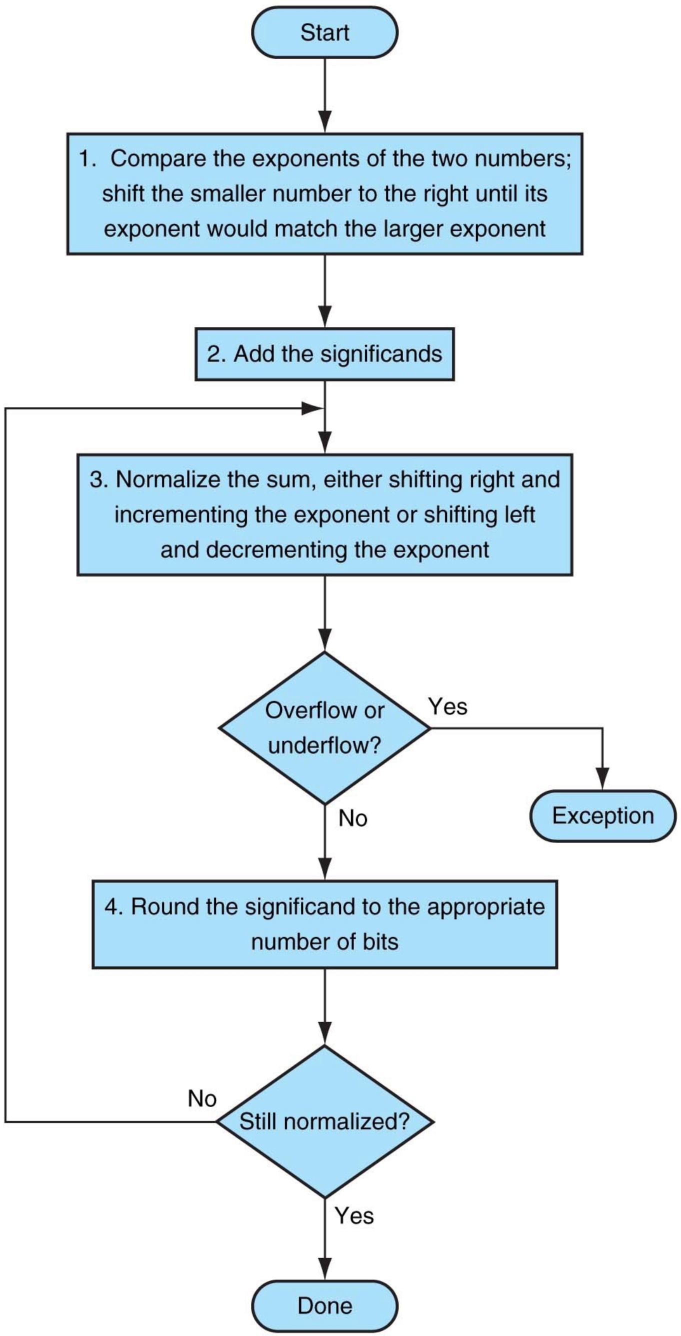
Floating-point addition

- **Binary Example :**

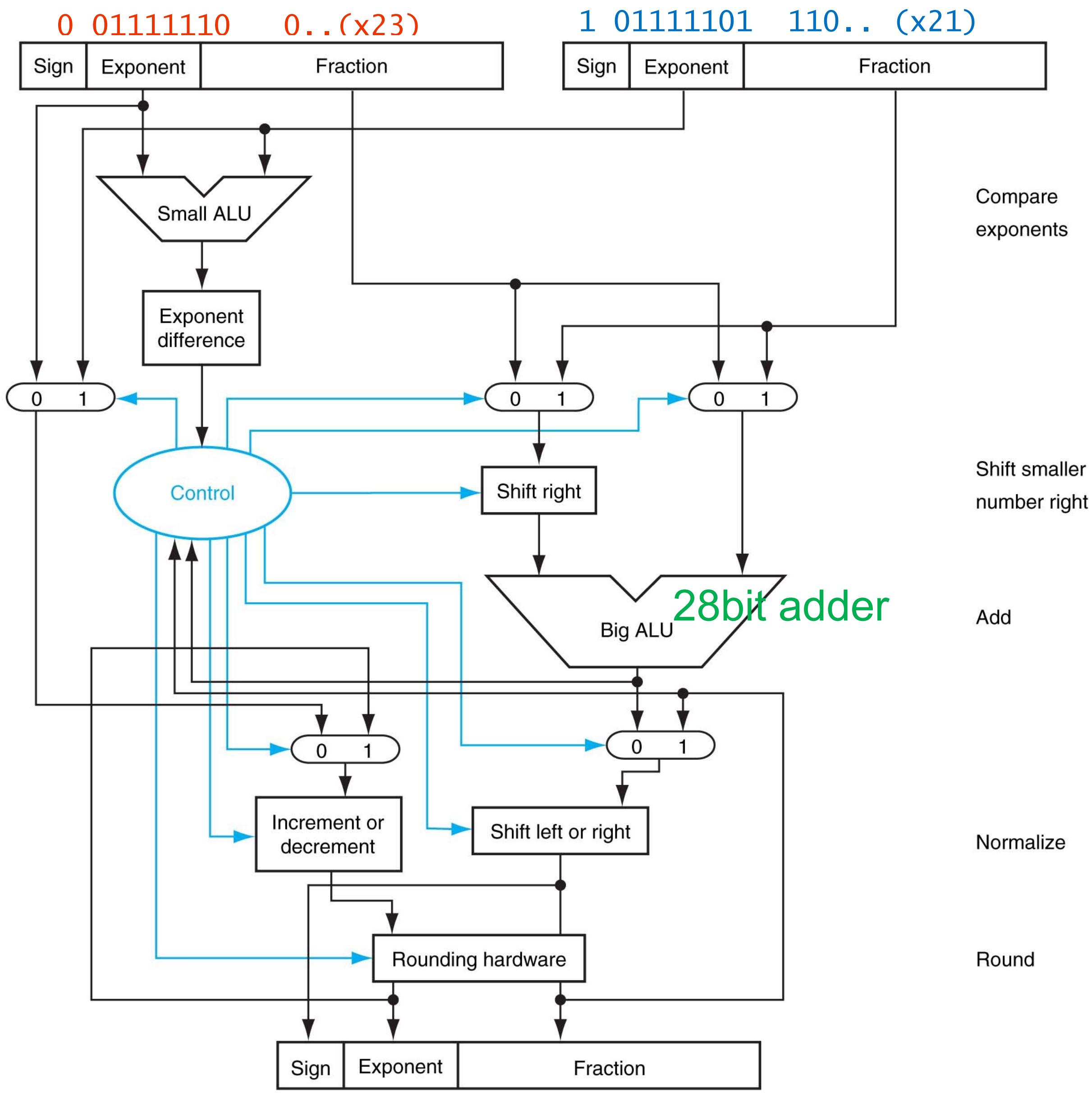
0.5 + -0.4375

0 01111110 0..(x23)

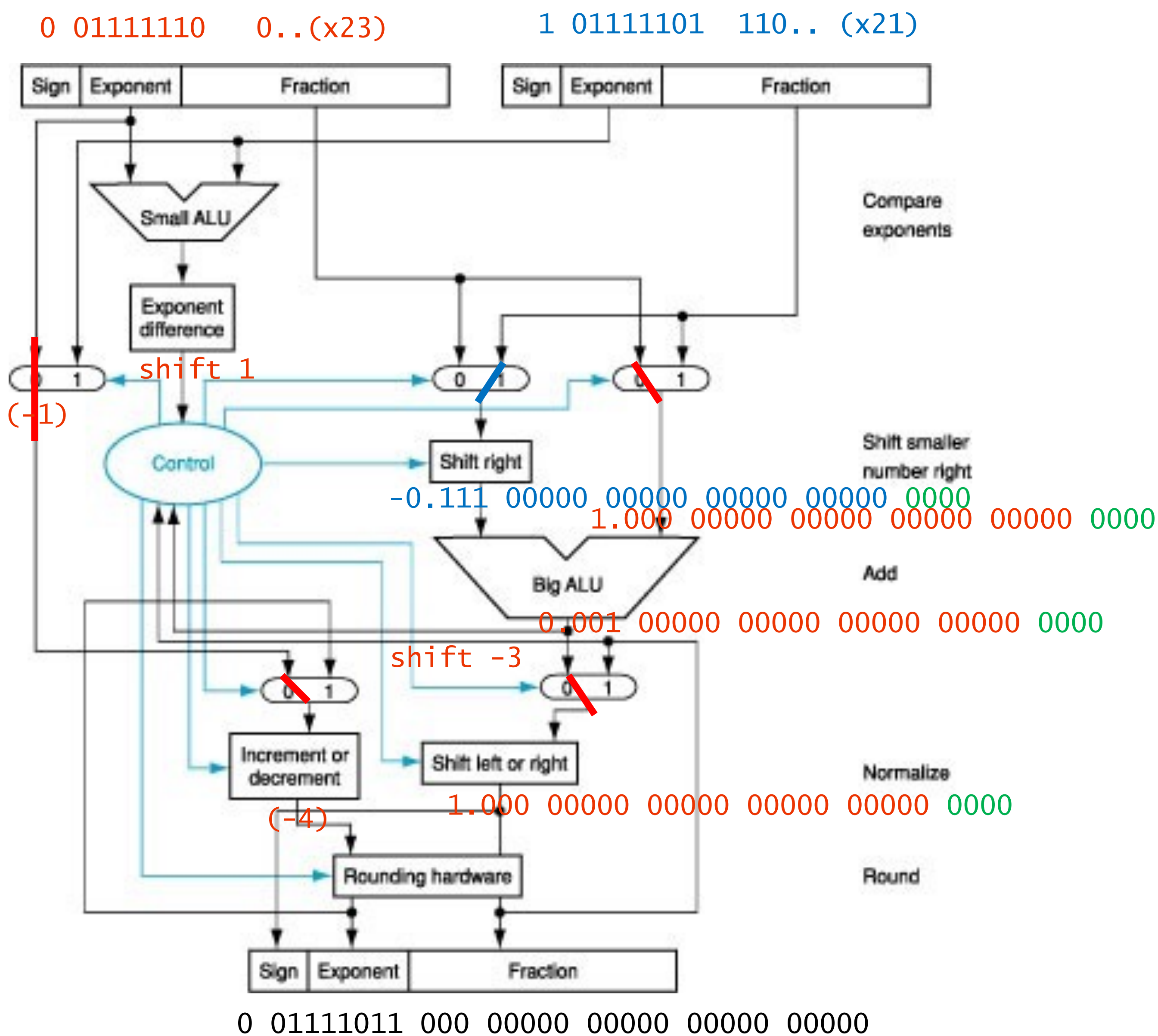
1 01111101 110.. (x21)



Floating-point addition hardware



Floating-point addition hardware



Floating-point addition hardware

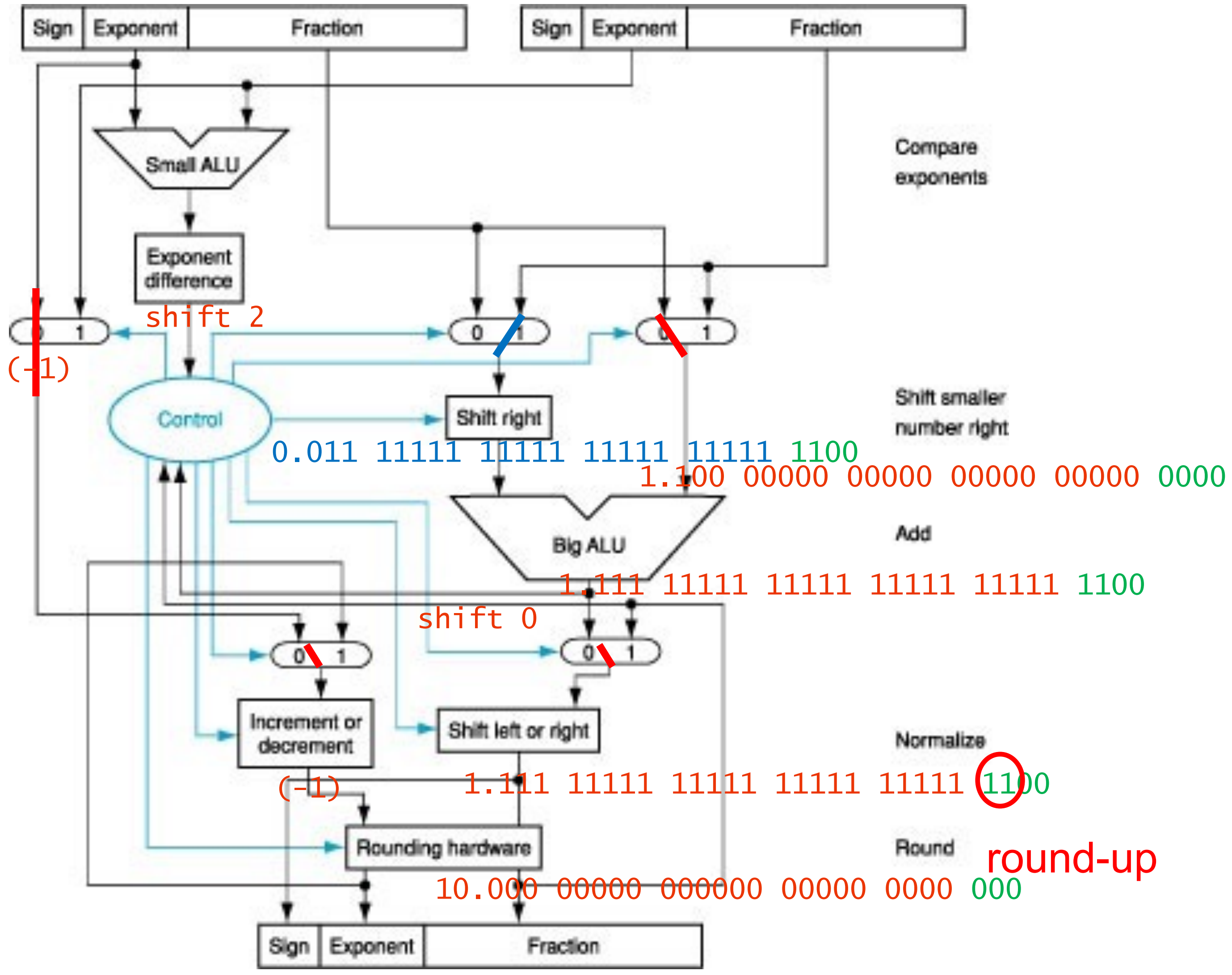
- **when 2nd normalization is performed**

0 01111110 100 00000 00000 00000 00000 = 1.1 x 2^{-1} = 0.75

0 01111100 111 11111 11111 11111 11111 = 1.11111…1 x 2^{-3} ≈ 2^{-2} = 0.25

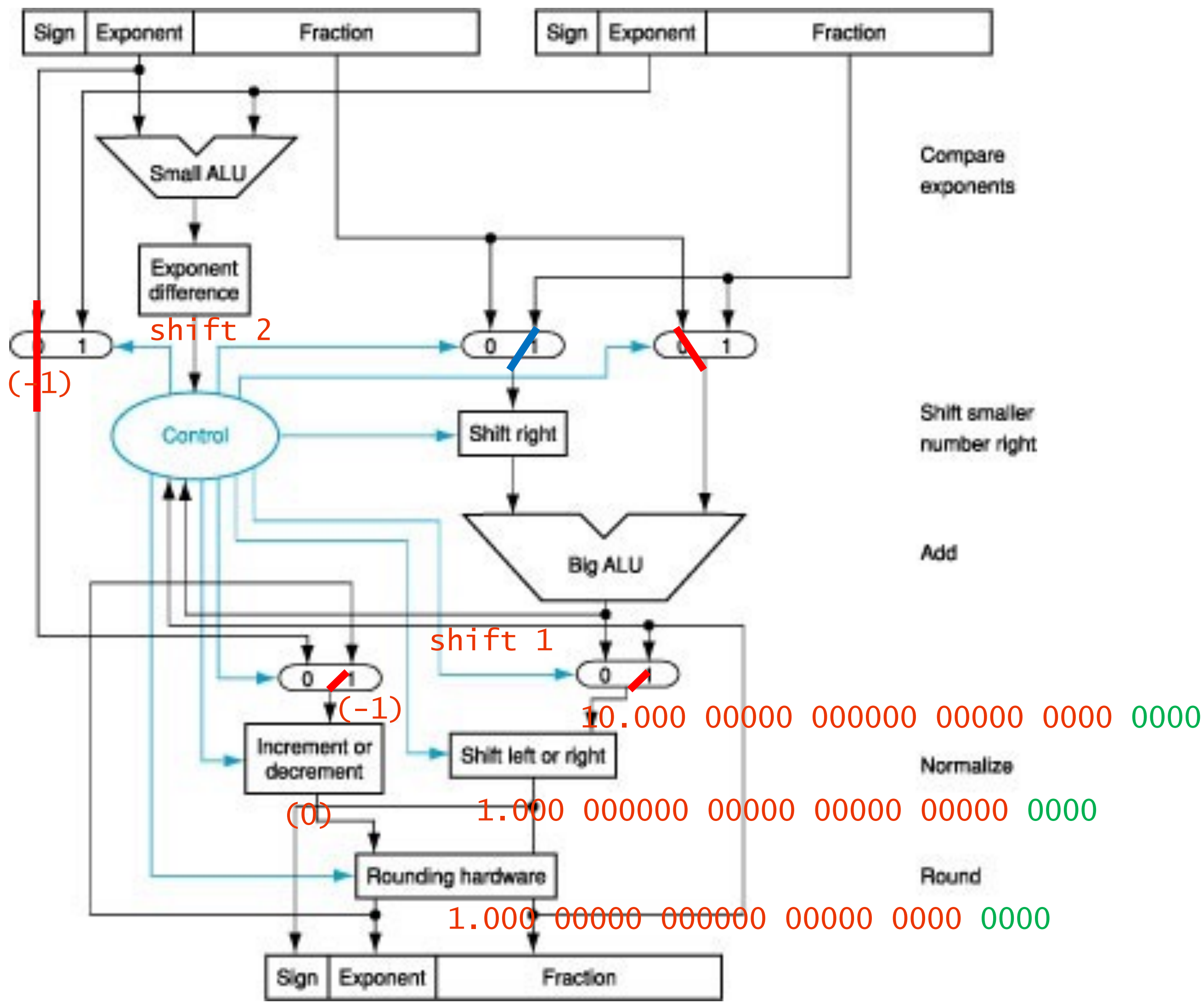
Floating-point addition hardware (when 2nd normalization is performed)

0 01111110 100 00000 00000 00000 00000 0 01111100 111 11111 11111 11111 11111



Floating-point addition hardware (when 2^{nd} normalization is performed)

0 01111110 100 00000 00000 00000 00000   0 01111100 111 11111 11111 11111 11111



0 01111111 000 00000 00000 00000 00000 → 1.0×2^0

FP Associativity

- Floating-point addition is not associative

  $(-1.5\times10^{38} + 1.5\times10^{38}) + 1.0 \neq -1.5\times10^{38} + (1.5\times10^{38} + 1.0)$

		(x+y)+z	x+(y+z)
x	-1.50E+38		-1.50E+38
y	1.50E+38	0.00E+00	
z	1.0	1.0	1.50E+38
		1.00E+00	0.00E+00

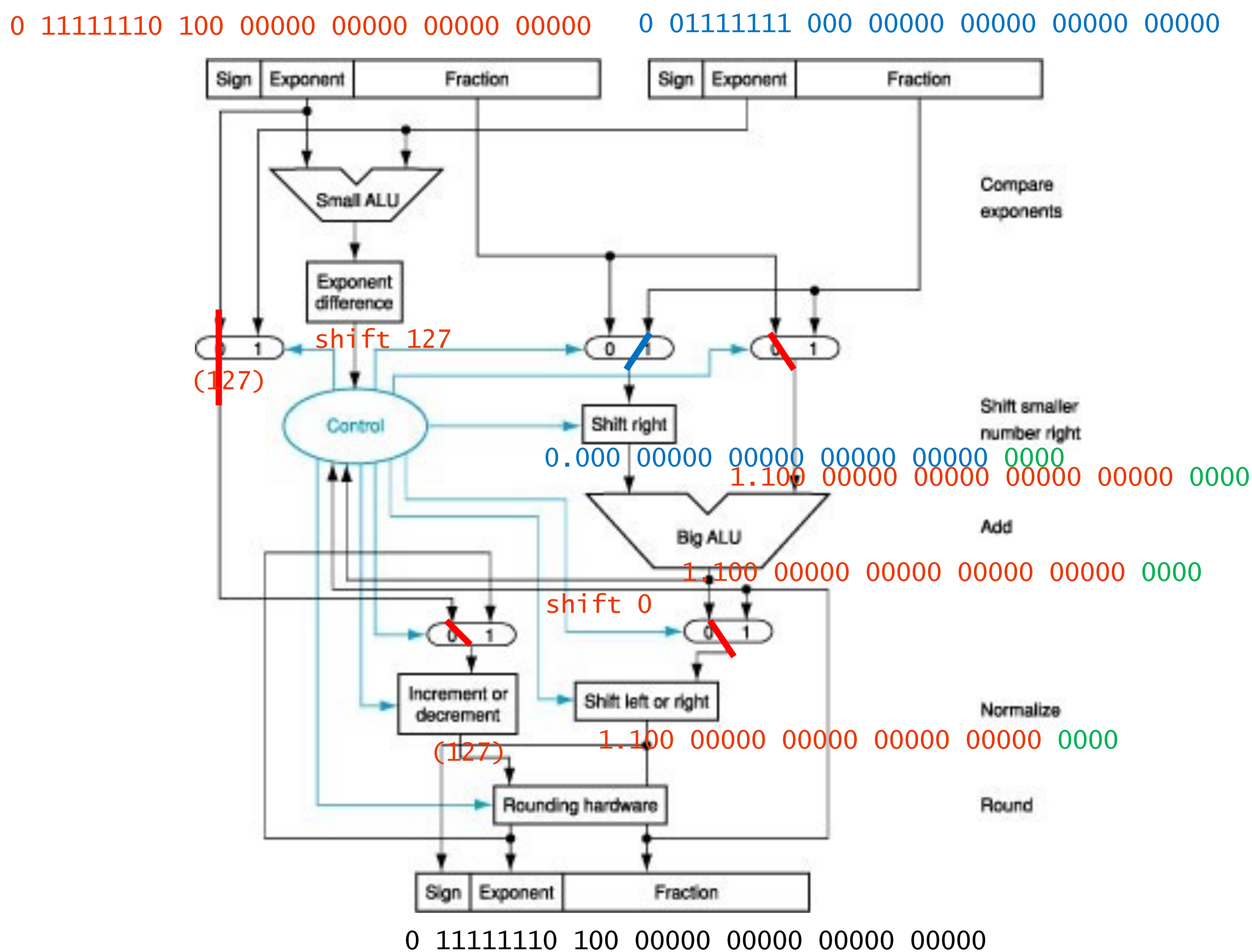
- Parallel programs may interleave operations in unexpected orders
  - Assumptions of associativity may fail
- Need to validate parallel programs under varying degrees of parallelism

$1.5 \times 10^{38} + 1.0$

0 11111110 100 00000 00000 00000 00000 = 1.1×2^{127} = 1.5×10^{38}

0 01111111 000 00000 00000 00000 00000 = 1.0×2^{0} = 1.0

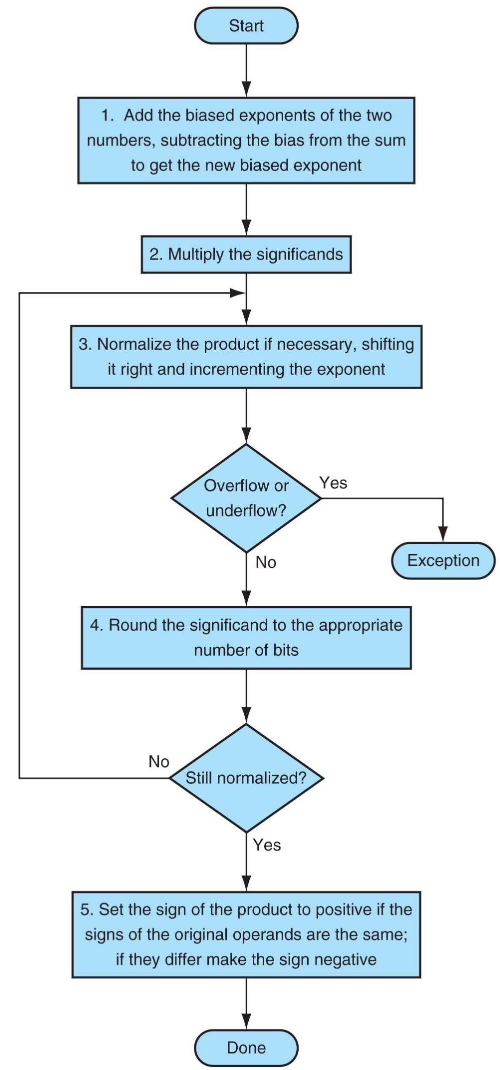
Floating-point addition $1.5 \times 10^{38} + 1.0$



Floating-Point Multiplication

$1.000\ldots0_2 \times 2^{-1} \times -1.110\ldots0_2 \times 2^{-2}$ (0.5×-0.4375)

- 1. Add exponents
  - Unbiased: $-1 + -2 = -3$
  - Biased: $(-1 + 127) + (-2 + 127) = -3 + 254 - 127 = -3 + 127$
- 2. Multiply significands
  - $1.000\ldots0_2 \times 1.110\ldots0_2 = 1.110\ldots0 \Rightarrow 1.110\ldots0_2 \times 2^{-3}$
- 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: $(+) \times (-) \Rightarrow (-)$
  - $-1.110_2 \times 2^{-3} = -0.21875$



FP Arithmetic Hardware

- FP multiplier is of similar complexity to FP adder
- FP arithmetic hardware usually does
  - Addition, subtraction, multiplication, division, reciprocal, square-root
  - FP ↔ integer conversion
- Operations usually takes several cycles
  - Can be pipelined

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 × 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)`

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
```

```
.data 0x10008000
f5: .float 5.0
    .float 9.0
    .float 32.0
    .float 70.0

.text
.globl main
main:
   addi $sp, $sp, -4
   sw    $ra, 0($sp)
   lwc1 $f12, 12($gp)
   jal    f2c
   lw     $ra, 0($sp)
   addi $sp, $sp, 4
   jr     $ra

f2c:
   lwc1 $f16, 0($gp)
   lwc1 $f17, 4($gp)
   div.s  $f16, $f16, $f17
   lwc1 $f17, 8($gp)
   sub.s $f17, $f12, $f17
   mul.s $f0, $f16, $f17
   jr $ra
```

```
int a,b;
a = a+b;

float c,d;
c=c+d;

double e,f;
e=e+f;

a = a + e;
```

Floating Point Complexities

- Operations are somewhat more complicated.
- In addition to overflow we can have underflow
- Accuracy can be a big problem
  - positive divided by zero yields infinity
  - zero divide by zero yields not a number (NaN)
  - other complexities
- Implementing the standard can be tricky
- Floating-point addition is not associative

$$(-1.5\times10^{38} + 1.5\times10^{38}) + 1.0 \neq -1.5\times10^{38} + (1.5\times10^{38} + 1.0)$$

Single precision		Double precision		Object represented
Exponent	Fraction	Exponent	Fraction	
0	0	0	0	0
0	Nonzero	0	Nonzero	± denormalized number
1–254	Anything	1–2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)