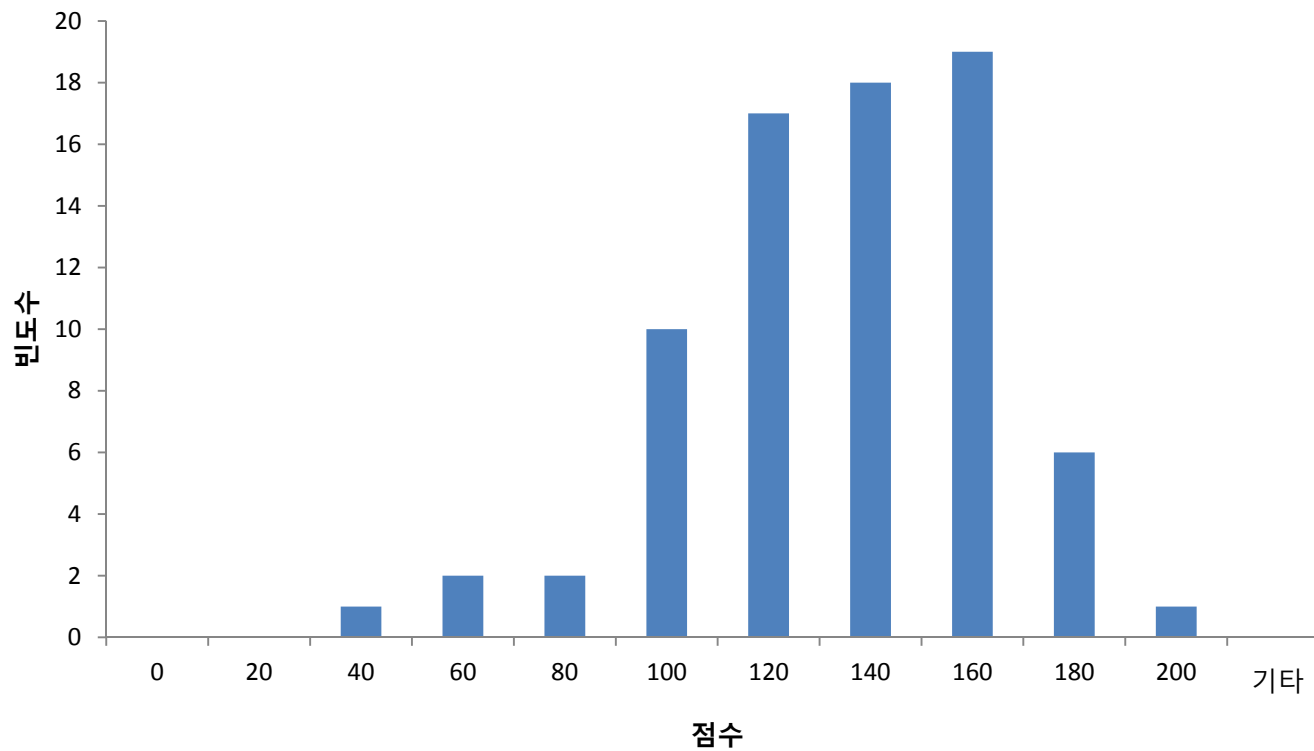


Exam #1

- Visit #400629 to check your exam sheet

Exam #1 Result
(Average = 127/200)



Floating Point Arithmetic

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
<http://csl.skku.edu>



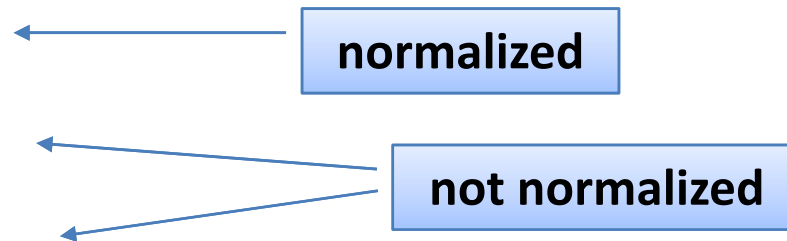
Floating Point (1)

- **Representation for non-integral numbers**

- Including very small and very large numbers
- Types `float` and `double` in C

- **Like scientific notation**

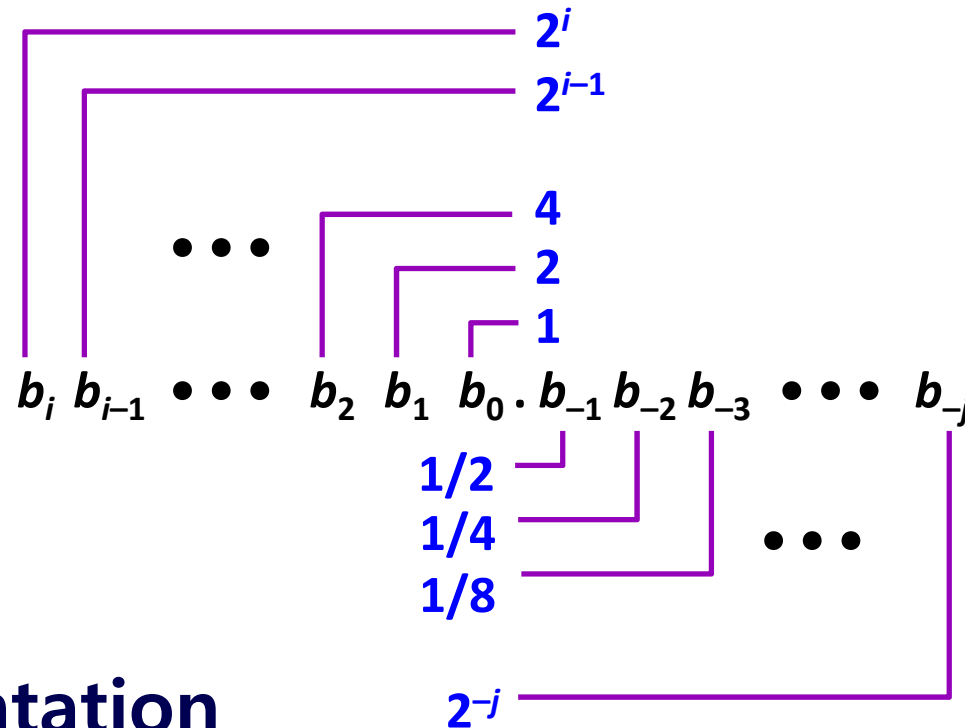
- -2.34×10^{56}
- $+0.002 \times 10^{-4}$
- $+987.02 \times 10^9$



- **In binary**

- $\pm 1.xxxxxxx_2 \times 2^{yyyy}$

Floating Point (2)



■ Representation

- Bits to right of "binary point" represent fractional powers of 2

- Represents rational number:
$$\sum_{k=-j}^i b_k \cdot 2^k$$

Floating Point (3)

▪ 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 (1)

■ Format

single: 8 bits

single: 23 bits

double: 11 bits

double: 52 bits



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

- 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: actual exponent + Bias
 - Ensures exponent is unsigned
 - Single: Bias = 127, Double: Bias = 1023

IEEE Floating Point (2)

▪ Single-precision range

- Exponents 00000000 and 11111111 reserved
- Smallest value
 - Exponent: 00000001
 - ⇒ actual exponent = $1 - 127 = -126$
 - Fraction: 000...00 ⇒ significand = 1.0
 - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Largest value
 - Exponent: 11111110
 - ⇒ actual exponent = $254 - 127 = +127$
 - Fraction: 111..11 ⇒ significand ≈ 2.0
 - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

IEEE Floating Point (3)

▪ Double-precision range

- Exponents 0000...00 and 1111...11 reserved
- Smallest value
 - Exponent: 000000000001
 - ⇒ actual exponent = $1 - 1023 = -1022$
 - Fraction: 000...00 ⇒ significand = 1.0
 - $\pm 1.0 \times 2^{-1022} \approx \pm 2.2 \times 10^{-308}$
- Largest value
 - Exponent: 111111111110
 - ⇒ actual exponent = $2046 - 1023 = +1023$
 - Fraction: 111..11 ⇒ significand ≈ 2.0
 - $\pm 2.0 \times 2^{+1023} \approx \pm 1.8 \times 10^{+308}$

IEEE Floating Point (4)

▪ 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

IEEE Floating Point (5)

▪ Example: -0.75

- $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
- $S = 1$
- Fraction = $1000\dots00_2$
- Exponent = $-1 + \text{Bias}$
 - Single: $-1 + 127 = 126 = 01111110_2$
 - Double: $-1 + 1023 = 1022 = 01111111110_2$
- Single: $10111111101000\dots00$
- Double: $10111111111101000\dots00$

IEEE Floating Point (6)

- Example: **1**1000000**101000...00** (single)

- $S = 1$
- Fraction = $01000...00_2$
- Exponent = $10000001_2 = 129$

- $x = (-1)^1 \times (1 + 0.01_2) \times 2^{(129 - 127)}$
 $= (-1) \times 1.25 \times 2^2$
 $= -5.0$

IEEE Floating Point (7)

▪ Denormal numbers

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

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

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

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

**Two representations of
0.0!**

IEEE Floating Point (8)

■ Infinities

- Exponent = 111..1, Fraction = 000..0
- \pm Infinity (∞)
- Can be used in subsequent calculations, avoiding need for overflow check

■ NaNs

- Exponent = 111...1, Fraction \neq 000...0
- Not-a-Number (NaN)
- Indicates illegal or undefined result
 - e.g., $0.0 / 0.0$
- Can be used in subsequent calculations

FP Addition (1)

- 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

FP Addition (2)

■ 4-digit binary example:

$$1.000_2 \times 2^{-1} + -1.110_2 \times 2^{-2} \quad (0.5 + -0.4375)$$

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

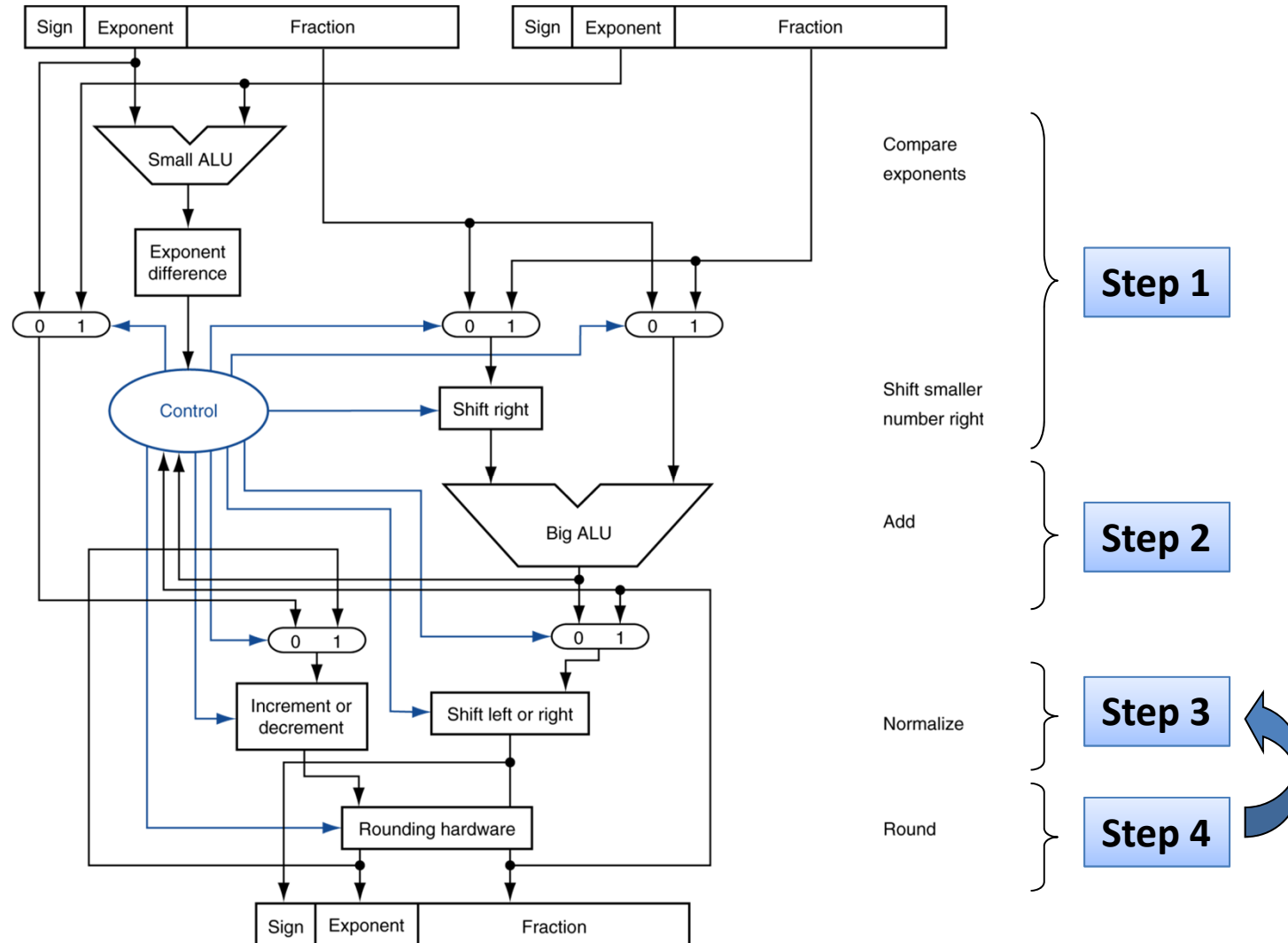
- $1.000_2 \times 2^{-4}$ (no change) = 0.0625

FP Addition (3)

■ 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 Addition (4)



FP Multiplication (1)

- 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

- $+1.021 \times 10^6$

FP Multiplication (2)

■ 4-digit binary example:

$$1.000_2 \times 2^{-1} \times -1.110_2 \times 2^{-2} \quad (0.5 \times -0.4375)$$

1. Add exponents

- Unbiased: $-1 + -2 = -3$
- Biased: $(-1+127) + (-2+127) = -3 + 254 - 127 = -3 + 127$

2. Multiply significands

- $1.000_2 \times 1.110_2 = 1.110_2 \Rightarrow 1.110_2 \times 2^{-3}$

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

- $-1.110_2 \times 2^{-3} = -0.21875$

FP Instructions (1)

■ 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 take several cycles
 - Can be pipelined

FP Instructions (2)

■ 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 registers
- FP instructions operate only on FP registers
 - More registers with minimal code-size impact
- FP load and store instructions
 - lwc1, ldc1, swc1, sdc1
 - e.g., ldc1 \$f8, 32(\$sp)

FP Instructions (3)

■ FP instructions in MIPS (cont'd)

- 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`, ...)
 - e.g., `c.lt.s $f3, $f4`
- Branch on FP condition code true or false
 - `bclt`, `bclf`
 - e.g., `bclt TargetLabel`

FP Instructions (4)

- **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)  
     lwc1    $f18, const9($gp)  
     div.s  $f16, $f16, $f18  
     lwc1    $f18, const32($gp)  
     sub.s  $f18, $f12, $f18  
     mul.s  $f0, $f16, $f18  
     jr     $ra
```

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