CMPE 411
Computer Architecture

<u>Lecture 9</u>

Floating Point Operations

September 28, 2017

www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm

Lecture's Overview

Previous Lecture:

- Algorithms for dividing unsigned numbers
  (Evolution of optimization, complexity)
- Handling of sign while performing a division
  (Remainder sign matches the dividend's)
- Hardware design for integer division
  (Same hardware as Multiply)

This Lecture:

- Representation of floating point numbers
- Floating point arithmetic
- Floating point hardware

Introduction

- What can be represented in N bits?
  - ➔ Unsigned 0 to 2^N
  - ➔ 2s Complement -2^{N-1} to $2^{N-1} - 1$
  - ➔ 1s Complement $-2^{N-1} + 1$ to $2^{N-1} - 1$
  - ➔ Excess M (E = e + M) $-M$ to $2^N - M - 1$
  - ➔ BCD 0 to $10^{N/4} - 1$

- But, what about?
  - ➔ very large numbers? 9,349,398,989,787,762,244,859,087,678
  - ➔ very small number? 0.0000000000000000000000045691
  - ➔ rational numbers 2/3
  - ➔ irrational numbers $\sqrt{2}$
  - ➔ transcendental numbers e, Π

* Slide is courtesy of Dave Patterson

Binary Coded Decimal (BCD)

- Each binary coded decimal digit is composed of 4 bits.

(a) $\underbrace{0\ 0\ 0\ 0}_{(0)_{10}}\quad \underbrace{0\ 0\ 1\ 1}_{(3)_{10}}\quad \underbrace{0\ 0\ 0\ 0}_{(0)_{10}}\quad \underbrace{0\ 0\ 0\ 1}_{(1)_{10}}\quad (+301)_{10}$ Nine's and ten's complement

(b) $\underbrace{1\ 0\ 0\ 1}_{(9)_{10}}\quad \underbrace{0\ 1\ 1\ 0}_{(6)_{10}}\quad \underbrace{1\ 0\ 0\ 1}_{(9)_{10}}\quad \underbrace{1\ 0\ 0\ 0}_{(8)_{10}}\quad (-301)_{10}$ Nine's complement

(c) $\underbrace{1\ 0\ 0\ 1}_{(9)_{10}}\quad \underbrace{0\ 1\ 1\ 0}_{(6)_{10}}\quad \underbrace{1\ 0\ 0\ 1}_{(9)_{10}}\quad \underbrace{1\ 0\ 0\ 1}_{(9)_{10}}\quad (-301)_{10}$ Ten's complement

- <u>Example</u>: Represent $+079_{10}$ in BCD: 0000 0111 1001
- <u>Example</u>: Represent -079_{10} in BCD: 1001 0010 0001

1. Subtract each digit of –079 from 9 to obtain the nine's complement, so 999 - 079 = 920.
2. Adding 1 produces the ten's complement: 920 + 1 = 921.
3. Converting each base 10 digit of 921 to BCD produces 1001 0010 0001

Excess (Biased)

- The leftmost bit is the sign (usually 1 = positive, 0 = negative).
- Representations of a number are obtained by adding a bias to the two's complement representation. This goes both ways, converting between positive and negative numbers.
- The effect is that numerically smaller numbers have smaller bit patterns, simplifying comparisons for floating point exponents.
- <u>Example</u> (excess 128 "adds" 128 to the two's complement version, ignoring any carry out of the most significant bit):

  $+12_{10} = 10001100_2$, $-12_{10} = 01110100_2$
- Only one representations for zero:

  $+0 = 10000000_2$, $-0 = 10000000_2$
- Range for an 8-bit representation is $[+127_{10}, -128_{10}]$
- Range for an N-bit representation is $[+(2^{N-1}-1)_{10}, -(2^{N-1})_{10}]$

* **Slide is courtesy of M. Murdocca and V. Heuring**

3-Bit Signed Integer Representations

Decimal	Unsigned	Sign–Mag.	1's Comp.	2's Comp.	Excess 4
7	111	–	–	–	–
6	110	–	–	–	–
5	101	–	–	–	–
4	100	–	–	–	–
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
+0	000	000	000	000	100
-0	–	100	111	000	100
-1	–	101	110	111	011
-2	–	110	101	110	010
-3	–	111	100	101	001
-4	–	–	–	100	000

*** Slide is courtesy of M. Murdocca and V. Heuring**

Introduction

❑ What can be represented in N bits?

- ➔ Unsigned 0 to 2^{N}
- ➔ 2s Complement -2^{N-1} to $2^{N-1} - 1$
- ➔ 1s Complement $-2^{N-1} + 1$ to $2^{N-1} - 1$
- ➔ Excess M (E = e + M) $-M$ to $2^{N} - M - 1$
- ➔ BCD 0 to $10^{N/4} - 1$

❑ But, what about?

- ➔ very large numbers? 9,349,398,989,787,762,244,859,087,678
- ➔ very small number? 0.0000000000000000000000045691
- ➔ rational numbers $2/3$
- ➔ irrational numbers $\sqrt{2}$
- ➔ transcendental numbers e, Π

* Slide is courtesy of Dave Patterson

Floating Point Numbers

decimal point

exponent

Sign, magnitude

6.02×10^{23} $\qquad$ 1.673×10^{-24}

Mantissa $\qquad$ *radix (base)*

Sign, magnitude

IEEE F.P. $\quad \pm 1.M \times 2^{e - 127}$

- Issues:
  - ➔ Arithmetic (+, -, *, /)
  - ➔ Representation, Normal form (no leading zeros)
  - ➔ Range and Precision
  - ➔ Rounding
  - ➔ Exceptions (e.g., divide by zero, overflow, underflow)
  - ➔ Errors
  - ➔ Properties (negation, inversion, if $A \geq B$ then $A - B \geq 0$)

* Slide is courtesy of Dave Patterson

Normalization

- The base 10 number 254 can be represented in floating point form as 254×10^0, or equivalently as:

  25.4×10^1, or

  2.54×10^2, or

  $.254 \times 10^3$, or

  $.0254 \times 10^4$, or

  infinitely many other ways, which creates problems when making comparisons
- Floating point numbers are usually normalized, with the radix point located in <u>only</u> one possible position for a given number
- Usually, but not always, the normalized representation places the radix point immediately to the left of the leftmost, nonzero digit in the fraction, as in: $.254 \times 10^3$

* Slide is courtesy of M. Murdocca and V. Heuring

Floating-Point Representation

- The size of the exponent determines the range of represented numbers
- Precision of the representation depends on the size of the significand
- The fixed word size requires a trade-off between accuracy and range
- Too large number cannot be represented causing an “overflow” while a too small number causes an “underflow”
- Negative and positive mantissas are designated by a sign bit using a sign and magnitude representation
- Exponents are usually represented using “excess M” representation to facilitate comparison between floating point numbers
- Double precision uses multiple words to expand the range of both the exponent and mantissa and limits overflow and underflow conditions

1	8	23
S	**Exponent**	**Significand**

Single precision

1	11	52
S	**Exponent**	**Significand**

Double precision

IEEE 754 Standard Representation

- Virtually has been used in every computer since after 1980

Single precision

	1	8	23
sign	S	E	M

exponent:
excess 127
binary integer

mantissa:
sign + magnitude, normalized
binary significand w/ hidden
integer bit: 1.M

Actual exponent is
e = E - 127

0 < E < 255

$$N = (-1)^{S}\, 2^{E-127}\, (1.M)$$

0 = 0 00000000 0 . . . 0   **-1.5 = 1 01111111 10 . . . 0**

- Magnitude of numbers that can be represented is in the range:

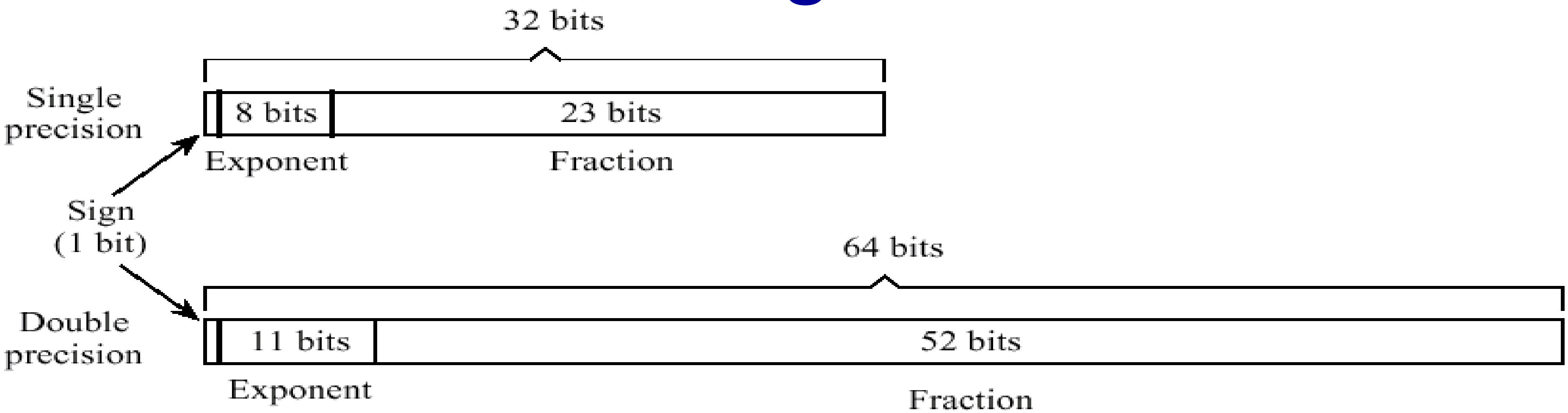
$$2^{-126}\,(1.0) \quad \text{to} \quad 2^{127}\,(2 - 2^{-23})$$

which is approximately:

$$1.8 \times 10^{-38} \quad \text{to} \quad 3.40 \times 10^{38}$$

Integer comparison is valid on IEEE Floating Point numbers of same sign

IEEE-754 Floating Point Formats



Example: show -12.625_{10} in single precision IEEE-754 format.

Step #1: Convert to target base. $-12.625_{10} = -1100.101_2$

Step #2: Normalize. $-1100.101_2 = -1.100101_2 \times 2^3$

Step #3: Fill in bit fields. Sign is negative, so sign bit is 1. Exponent is in excess 127 (not excess 128!), so exponent is represented as the unsigned integer 3 + 127 = 130. Leading 1 of significand is hidden, so final bit pattern is:

1 1000 0010 . 1001 0100 0000 0000 0000 000

* Slide is courtesy of M. Murdocca and V. Heuring

An Example

Show the IEEE 754 binary representation of -0.75 in single & double precision

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign Exponent Significand

$(-0.75)_{10} = (-3/4\ )_{10} = (-3/2^2)_{10} = (-11 \times 2^{-2})_2 = (-0.11)_2 = (-1.1 \times 2^{-1})_2$

Single precision representation is: $(-1)^s \times (1+\text{Significand}) \times 2^{(\text{Exponent - 127})}$

$(-0.75)_{10}$ is represented as $(-1)^1 \times (1+.1000\ 0000\ 0000\ 0000\ 0000\ 000) \times 2^{(126)}$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Sign Exponent First 20-bit of Significand

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Last 32-bit of Significand

Double precision representation is: $(-1)^s \times (1+\text{Significand}) \times 2^{(\text{Exponent - 1023})}$

$(-0.75)_{10}$ is represented as $(-1)^1 \times (1+.1000\ 0000\ \ldots\ 0000\ 0000) \times 2^{(1022)}$

Floating Point Arithmetic

- Floating point arithmetic differs from integer arithmetic in that exponents must be handled as well as the magnitudes of the operands.
- The exponents of the operands must be made equal for addition and subtraction. The fractions are then added or subtracted as appropriate, and the result is normalized.

<u>Example</u>: Perform the following addition: $(.101 \times 2^3 + .111 \times 2^4)_2$

- Start by adjusting the smaller exponent to be equal to the larger exponent, and adjust the fraction accordingly. Thus we have $.101 \times 2^3 = .010 \times 2^4$, losing $.001 \times 2^3$ of precision in the process.
- The resulting sum is $(.010 + .111) \times 2^4 = 1.001 \times 2^4 = .1001 \times 2^5$ and rounding to three significant digits, $.100 \times 2^5$, and we have lost another 0.001×2^4 in the rounding process.

* Slide is courtesy of M. Murdocca and V. Heuring

Floating Point Addition

For addition (or subtraction) this translates into the following steps:

(1) Compute Ye - Xe *(getting ready to align)*

(2) Right shift Xm to form Xm $2^{(Xe - Ye)}$

(3) Compute Xm $2^{(Xe - Ye)}$ + Ym

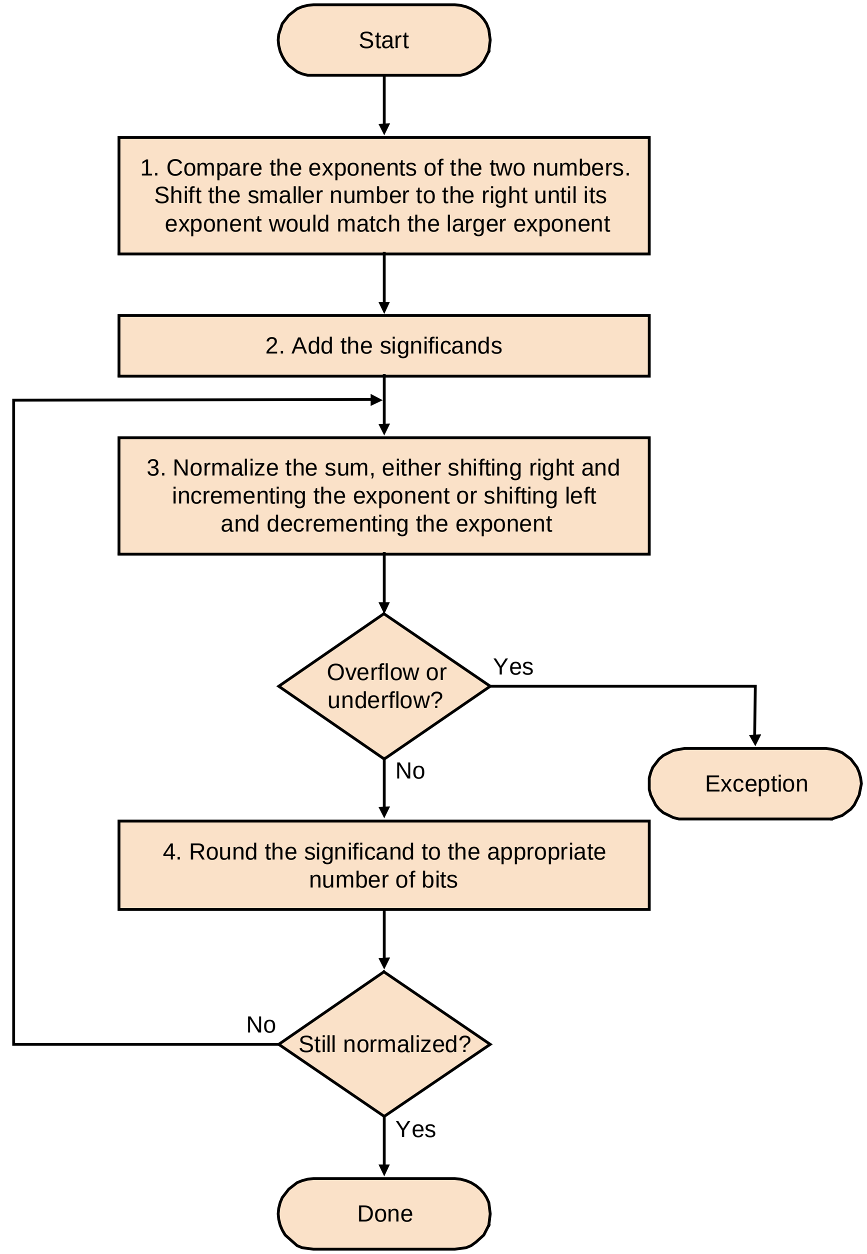
If representation demands normalization, then the following step:

(4) Left shift result, decrement result exponent

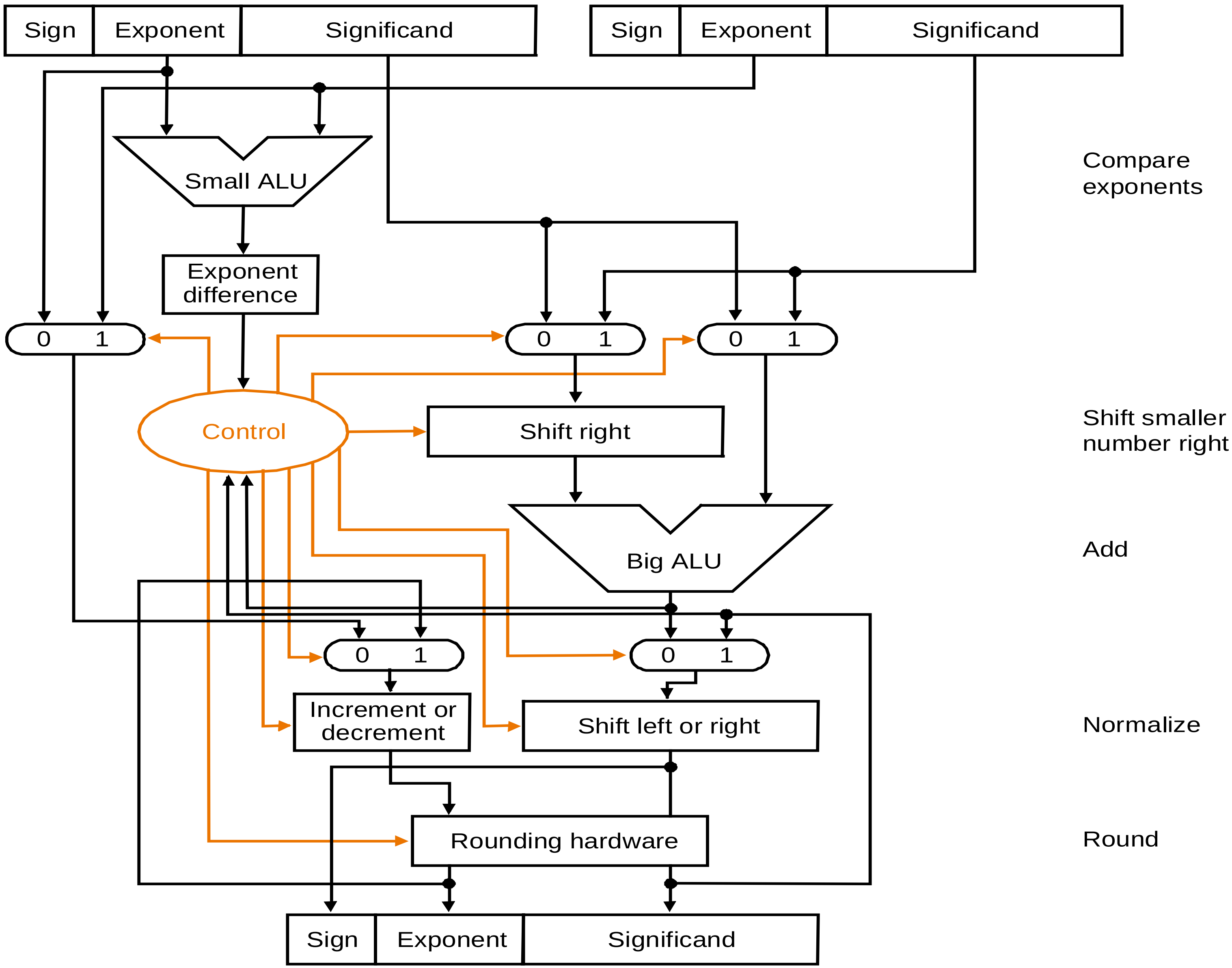
Right shift result, increment result

Continue until MSB of data is (Hidden bit)

(5) If result is 0 mantissa, may need to set exponent to zero by special step



Floating Addition Hardware



Floating Point Multiplication/Division

- Floating point multiplication/division are performed in a manner similar to floating point addition/subtraction, except that the sign, exponent, and fraction of the result can be computed separately.
- Like/unlike signs produce positive/negative results, respectively
- Exponent of result is obtained by adding/subtracting exponents for multiplication/division. Fractions are multiplied or divided according to the operation, and then normalized.

Example: Perform : $(+.110 \times 2^5) / (+.100 \times 2^4)_2$

- The source operand signs are the same, which means that the result will have a positive sign. We subtract exponents for division, and so the exponent of the result is 5 – 4 = 1.
- We divide fractions, producing the result: 110/100 = 1.10.
- Putting it all together, the result of dividing $(+.110 \times 2^5)$ by $(+.100 \times 2^4)$ produces $(+1.10 \times 2^1)$. After normalization, the final result is $(+.110 \times 2^2)$.

* Slide is courtesy of M. Murdocca and V. Heuring

Floating Point Multiplication

For addition (or subtraction) this translates into the following steps:

(1) Compute Ye + Xe *(adding exponents)*

(2) doubly biased exponent must be corrected:

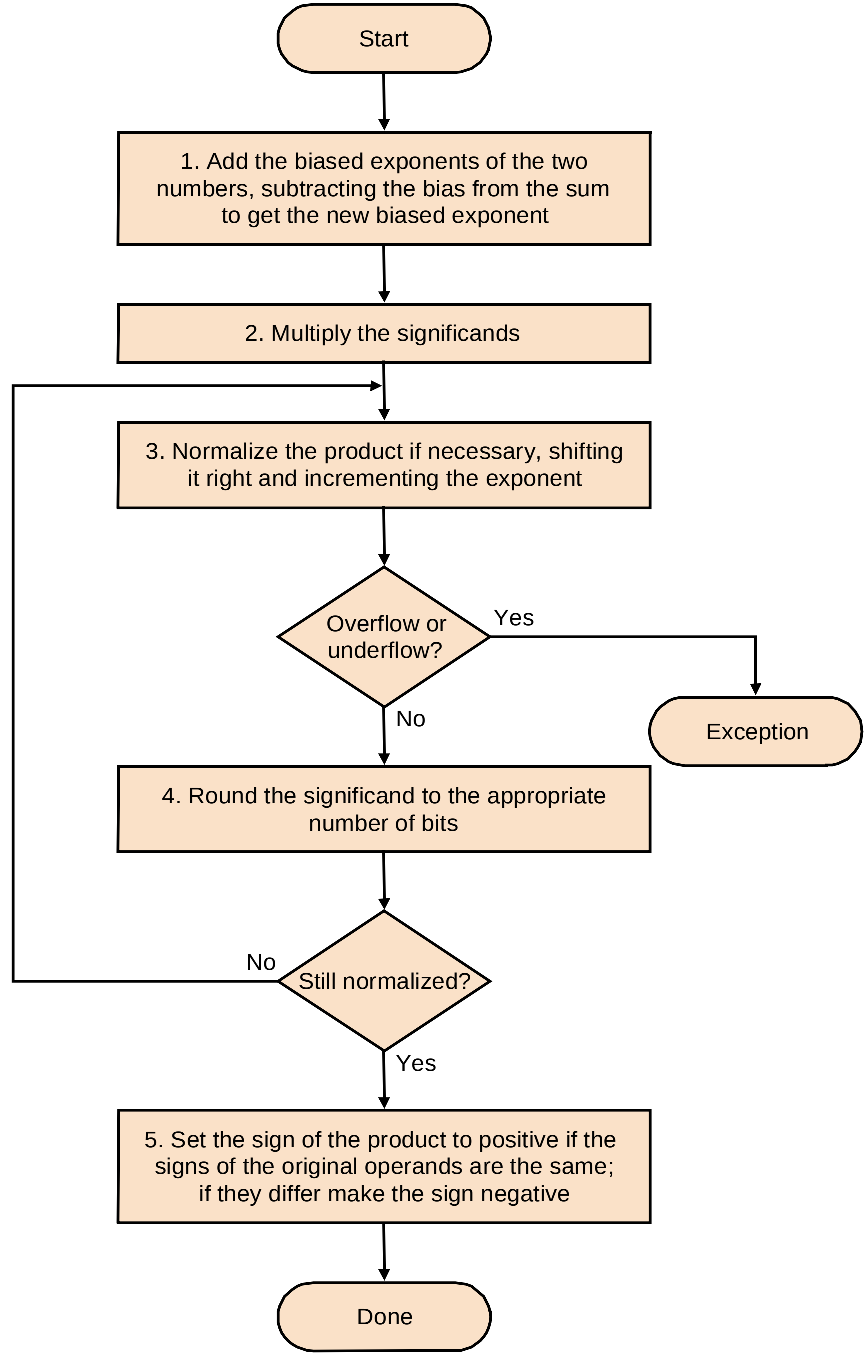
Xe = 7	Xe = 1111	= 15	= 7 + 8
Ye = -3	Ye = 0101	= 5	= -3 + 8
Excess 8	10100	20	4 + 8 + 8

(3) Multiply the signficands

(4) Perform normalization

(4) Round the number to the specified size

(5) Calculate the sign of the product



Denormalized Numbers

- The smallest single precision normalized number is
  $1.0000\ 0000\ 0000\ 0000\ 0000\ 001 \times 2^{-126}$

  while the smallest single precision denormalized number is
  $0.0000\ 0000\ 0000\ 0000\ 0000\ 001 \times 2^{-126}$ or 1.0×2^{-149}

- The IEEE 754 standard allows some floating point number to be denormalized in order to narrow the gap between 0 and the smallest normalized number
- Demorlaized numbers are allowed to degrade in significance until it becomes 0 (gradual underflow)
- The potential of occasional denormalized operands complicates the design of the floating point unit
- PDP-11, VAX cannot represent denormalized numbers and underflow to zero instead

Encoding of IEEE 754 Numbers

+/- infinity

S	1 . . . 1	0 . . . 0

- result of operation *overflows*, i.e., is larger than the largest number that can be represented
- overflow is not the same as divide by zero (raises a different exception)

NaN

S	1 . . . 1	non-zero

HW decides what goes here

- Not a number, but not infinity (e.q. sqrt(-4))
- Generates invalid operation exception (unless operation is comparison)
- NaNs propagate: f(NaN) = NaN

Single Precision		**Double Precision**		**Object represented**
Exponent	*Significand*	*Exponent*	*Significand*	
0	0	0	0	0
0	Nonzero	0	Nonzero	± de-normalized number
1-254	Anything	1-2046	Anything	± floating-point number
255	0	2047	0	± infinity
255	Nonzero	2047	Nonzero	NaN (Not a Number)

Conclusion

- ***Summary***
  - ➔ Representation of floating point numbers

    (Sign, exponent, mantissa, single & double precision, IEEE 754)
  - ➔ Floating point arithmetic

    (Addition and Multiplication)
  - ➔ Normalizing Floating point numbers

    (Rounding, zero floating point number, special interpretation)

- ***Next Lecture***
  - ➔ Processor datapath and control
  - ➔ Simple hardwired implementation
  - ➔ Design of a control unit

Read section 3.5 in 5th Ed., or section 3.5 in 4th Ed. of the textbook