COMPSCI 210 S1T Computer Systems

Floating Point Numbers Arithmetic

Agenda & Reading

- Agenda:
  - Floating Point operations
    - Addition
    - Subtraction
    - Multiplication
    - Division

Floating Point Operations

- Exponents and fraction must be handled separately & differently.
- Consider the Floating point numbers:
  - $X = Mx * 2^{Ex}$, $Y = My * 2^{E}$
- Basic Operations
  - Addition: $X + Y = (Mx * 2^{Ex-Ey} + My) * 2^{Ey}$, $Ex <= Ey$
  - Subtraction: $X - Y = (Mx * 2^{Ex-Ey} - My) * 2^{Ey}$, $Ex <= Ey$
- Procedures for addition/subtraction:
  - Adjust exponents and align mantissa
    - The exponent of the operands must be made equal for addition and subtraction.
    - If Ey>Ex Right shift Mx to form $Mx*2^{Ex-Ey}$
    - If Ex>Ey Right shift My to form $My*2^{Ey-Ex}$
  - Add or subtract mantissa
  - Normalize the result
    - Left shift result, decrement result exponent (e.g., if result is 0.001xx...) or
    - Right shift result, increment result exponent (e.g., if result is 10.1xx...)
  - Check result
    - If larger than maximum exponent allowed return exponent overflow
    - If smaller than minimum exponent allowed return exponent underflow
    - If result mantissa is 0, may need to set the exponent to zero to return a zero.

Example on Decimal Value

- $3.25 * 10^{2} + 2.63 * 10^{-1}$
- Steps:
  - 1) Align decimal points:
    - Start by adjusting the smaller exponent to be equal to the larger exponent, and modify the fraction accordingly.
    - Take $2.63 * 10^{-1}$
      - Original value = $2.63 * 10^{-1}$ (right-shift mantissa-> increase exponent)
      - Shift 1 place = $0.263 * 10^{0}$
      - Shift 2 places = $0.0263 * 10^{1}$
      - Shift 3 places = $0.00263 * 10^{2}$
  - 2) Add the numbers
    - $\quad 3.25 * 10^{2}$
    - $\underline{+\ 0.00263 * 10^{2}}$
    - $\quad 3.2563 * 10^{2}$
  - 3) Normalize the result
    - No need to change. It is already normalized.
  - 4) Check result
    - OK. Answer = $3.2563 * 10^{2}$

Example on Binary value

- $0.101*2^2 + 0.111*2^4$
- Steps:
  - 1) Align decimal points:
    - Start by adjusting the smaller exponent to be equal to the larger exponent, and modify the fraction accordingly.
      - Take $0.101*2^2$
      - Original value = $0.101*2^2$
      - Shifted 1 place = $0.101*2^2 => 0.0101*2^3$
      - Shifted 2 places = $0.0101*2^3 => 0.00101*2^4$
  - 2) Add the numbers
    - $\ 0.111\ \ *2^4$
    - $\underline{+\ 0.00101*2^4}$
    - $\ 1.00001*2^4$
  - 3) Normalize the result
    - No need to change. It is normalized.
  - 4) Check result
    - OK. Answer = $1.0011*2^4$

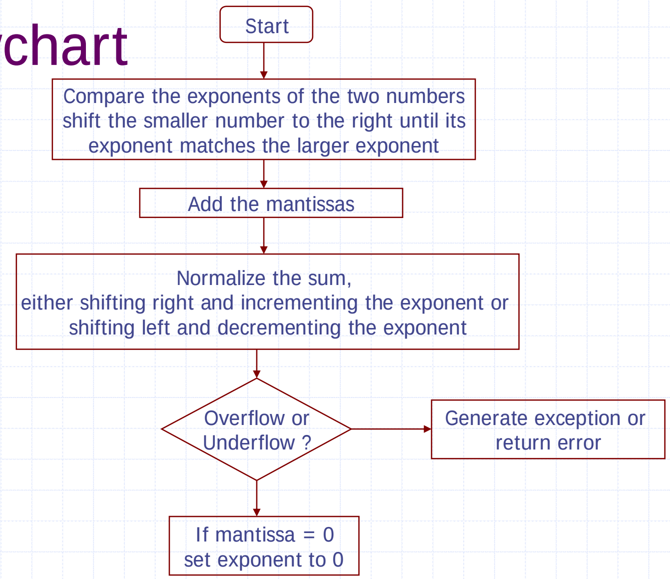
Floating Point Addition

- 1.25 + 0.25
  - 0.25 = 0 01111101 000...000
  - 1.25 = 0 01111111 010...000
- Steps:
  - Adjust exponents and align mantissa
    - Start by adjusting the smaller exponent to be equal to the larger exponent
    - Take 0.25 (0 01111101 000...000) (with smaller exponent)
      - Original Value: E:01111101 M:000...000
      - Shifted 1 place: E:01111110 M:100...000 (Note: "1" is the hidden bit)
      - Shifted 2 places: E:01111111 M:010...000
  - Add mantissa bits
    - 0 01111111 1.010…000
    - +0 01111111 0.010…000
    - 0 01111111 1.100…000
  - Normalize result: The hidden bit
    - No need to change. It is normalized.
  - Check result
    - OK. Answer = 0 01111111 100..000

Floating Point Addition

- 6.5 + 1.75
  - 6.5 = 0 10000001 1010...000
  - 1.75 = 0 01111111 110...000
- Steps:
  - Adjust exponents and align mantissa
    - Start by adjusting the smaller exponent to be equal to the larger exponent
    - Take 1.75 (0 01111111 110...000) (with smaller exponent)
      - Original Value: E:01111111 M:110...000
      - Shifted 1 place: E:10000000 M:11100...000 (Note: "1" is the hidden bit)
      - Shifted 2 places: E:10000001 M:01110...000
  - Add mantissa bits
    - 0 10000001 1.1010…000
    - +0 10000001 0.01110…00
    - 0 10000001 10.0001000…0
  - Normalize result:
    - Shift the mantissa right by 1 bit increase the exponent by 1
    - E: 10000001 +1 = 10000010, mantissa = 1.0000100..0
  - Check result:
    - OK. Answer = 0 10000010 0000100...000

Flowchart



Floating Point Subtraction

- 1.25 - 0.25
  - 0.25 = 0 01111101 000…000
  - 1.25 = 0 01111111 010…000
- Steps:
  - Adjust exponents and align mantissa
    - Start by adjusting the smaller exponent to be equal to the larger exponent
    - Take 0.25 (0 01111101 000…000) (with smaller exponent)
      - Shifted 1 place: E:01111110 M:100…000 (Note: "1" is the hidden bit)
      - Shifted 2 places: E:01111111 M:010…000
  - Subtract mantissa bits
    - `0 01111111 1.010…000`
    - `-0 01111111 0.010…000`
    - `0 01111111 1.000…000`
  - Normalize result:
    - No need to change. It is normalized.
  - Check result
    - OK. Answer = 0 01111111 000..000

Floating Point Subtraction

- 6.5 – 1.75
  - 6.5 = 0 10000001 1010…000
  - 1.75 = 0 01111111 110…000
- Steps:
  - Adjust exponents and align mantissa
    - Start by adjusting the smaller exponent to be equal to the larger exponent
    - Take 1.75 (0 01111111 110…000) (with smaller exponent)
      - Shifted 1 place: E:10000000 M:1110…000 (Note: "1" is the hidden bit)
      - Shifted 2 places: E:10000001 M:01110…000
  - Subtract mantissa bits
    - `0 10000001 1.10100…000`
    - `-0 10000001 0.01110…000`
    - `0 10000001 1.001100…000`
  - Normalize result:
    - No need to change.
  - Check result:
    - OK. Answer = 0 10000001 00110…00

Addition Notes

- Truncation Errors
  - If the exponents differ by more than 24, the smaller number will be shifted right entirely out of the mantissa field, producing a zero mantissa.
  - The sum will then equal the larger number.
  - Such truncation errors occur when the numbers differ by a factor of more than 2^{24} , which is approximately 1.6 x 10^7 .
  - Thus, the precision of IEEE single precision floating point arithmetic is approximately 7 decimal digits.
- Alternatively, floating point subtraction is achieved simply by inverting the sign bit and performing addition of signed mantissas.
  - Negative mantissas are handled by first converting to 2's complement and then performing the addition.
  - After the addition is performed, the result is converted back to sign-magnitude form.
- When adding numbers of opposite sign, cancellation may occur, resulting in a sum which is arbitrarily small, or even zero if the numbers are equal in magnitude.

Multiplication

- Floating Point Operations:
  - $X = (-1)^{Sx}$ Mx * 2^{Ex} ,$Y = (-1)^{Sy}$ My * 2^{E}
  - Multiplicatoin: X * Y = (Mx * My) * 2^{Ex+Ey}
- Procedures for Multiplicatoin:
  - Check Zeros
    - If one or both operands is equal to zero, return the result as zero.
  - Compute the sign of the result Sx XOR Sy
  - Multiply mantissa
    - Mx*My
    - Round the result to the allowed number of mantissa bits
  - Add exponents
    - biased exponent (Ex) + biased exponent (Ey) - bias
  - Normalize the result
    - Left shift result, decrement result exponent (e.g., if result is 0.001xx…) or
    - Right shift result, increment result exponent (e.g., if result is 10.1xx…)
  - Check result
    - If larger than maximum exponent allowed return exponent overflow
    - If smaller than minimum exponent allowed return exponent underflow

Decimal Multiplication/Division

- Example 1: $3.0 * 10^1 * 4.5 * 10^2$
- Steps:
  - Sign = 0 XOR 0 = 0
  - Multiply mantissa (don't forget the hidden bit)
    - 3.0 * 4.5 = 13.5
  - Add exponents
    - 1 + 2 = 3
  - Normalize the result
    - $13.5 * 10^3$,Shift 1 place -> increase exponent = $1.35 * 10^4$
  - Check result
    - OK. Answer = $1.35 * 10^4$
- Example 2: $4.0 * 10^3 * 0.5 * 10^2$
  - Sign = 0 XOR 0 = 0
  - Divide mantissa (don't forget the hidden bit)
    - 4.0 / 0.5 = 8.0
  - Subtract exponents
    - 3 -2 = 1
  - Normalize the result
    - $8.0 * 10^1$ No change.
  - Check result
    - OK. Answer = $8.0 * 10^1$

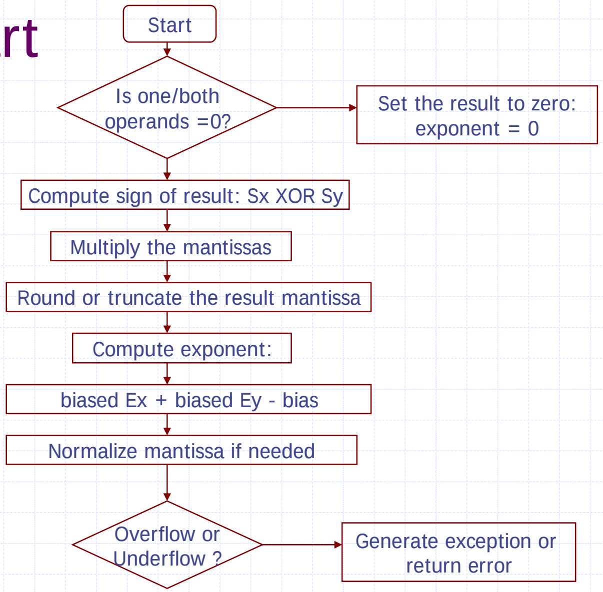
Floating Point Multiplication

- -18 * 9.5
  - -18 = 1 10000011 0010...000
  - 9.5 = 0 10000010 0011...000
- Steps:
  - Sign = 0 XOR 1 = 1
  - Multiply mantissa (don't forget the hidden bit)

```
       1.0010
     * 1.0011
        10010
       10010
      00000
     00000
    10010
    101010110  = 1.01010110
```

  - Add exponents
    - 1000 0011 + 1000 0010 - 01111111 = 1000 0110
  - Normalize the result
    - It is normalized. No change
  - Check result
    - OK. Answer = 1 10000110 01010110...0

Flowchart



Division

- Floating Point Operations:
  - $X = (-1)^{Sx}\ Mx * 2^{Ex}$,$Y = (-1)^{Sy}\ My * 2^{E}$
  - Division: $X / Y = (Mx / My) * 2^{Ex-Ey}$
- Procedures for Division:
  - Check Zeros
    - If both operands is equal to zero, return the result as NaN
    - If Y is equal to zero, return the result as infinity.
  - Compute the sign of the result Sx XOR Sy
  - Divide mantissa
    - Mx/My
    - Round the result to the allowed number of mantissa bits
  - Subtract exponents
    - Division: biased exponent (Ex) - biased exponent (Ey) + bias
  - Normalize the result
    - Left shift result, decrement result exponent (e.g., if result is 0.001xx...) or
    - Right shift result, increment result exponent (e.g., if result is 10.1xx...)
  - Check result
    - If larger than maximum exponent allowed return exponent overflow
    - If smaller than minimum exponent allowed return exponent underflow

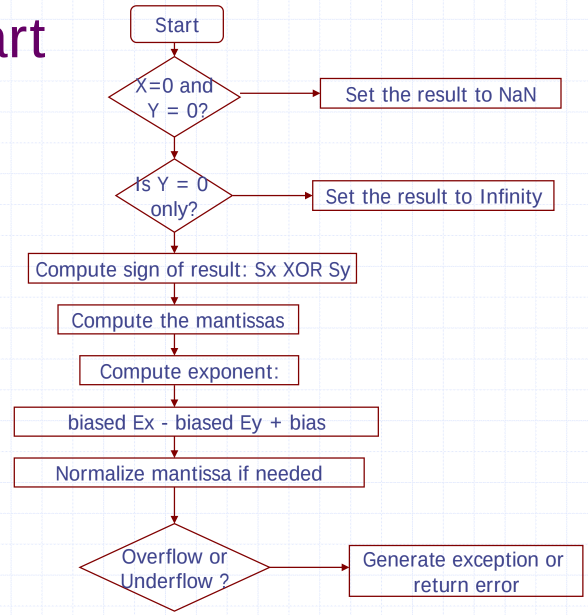
Floating Point Division

- 3.75 / 1.5
  - 3.75 = 0 10000000 1110...000
  - 1.5 = 0 01111111 100...000
- Steps:
  - Sign = 0 XOR 0 = 0
  - Divide mantissa (don't forget the hidden bit)

```
     _101
 11)1111
    11___
      11
     _11
       0 =
```

    - = 101 => 1.01
  - Subtract exponents
    - 10000000 - 01111111 + 0111111 = 1000 0000
  - Normalize the result
    - It is normalized. No change
  - Check result
    - OK. Answer = 0 10000000 0100..000

Flowchart



Conversion Example:

- What is the IEEE floating point representation of 6.5_{10}? Hence, what is the representation for 52?
  - Step 1: 6.5_{10} :
    - Sign = 0
    - 6.5_{10}=> Binary number = 110.1
    - Normalization (shift radix point to left by 2 places) => 1.101 x 2^2
    - Exponent = 127+2= 129 = 10000001
    - Mantissa = 1010...0
    - Answer = 0 10000001 1010...0 = 40D00000
  - Step 2: 52/6.5 = 8 = 2^3 i.e. 6.5 x 2^3 = 52
    - Sign bit : unchanged
    - Mantissa : unchanged
    - Exponent : exponent from step 1 + 3 = (129 + 3) = 10000100
    - Answer = 0 10000100 1010...0 = 42500000

Conversion Example (con't)

- What is the IEEE floating point representation of 1.25_{10}? Hence, what is the representation for 1.75 and 1.875?
  - Step 1: 1.25 = 3FA00000 = 0 01111111 010...0
  - Step 2: 1.75
    - 1.75 = 1.25* 2^0 + 0.5 * 2^0
    - Sign bit: unchanged
    - Exponent bit: unchanged
    - Mantissa = + 0.5
    - Answer = 0 01111111 110...0 = 3FE00000
  - Step 3:
    - 1.875 = 1.25* 2^0 + 0.5 * 2^{0} + 0.125 * 2^0
    - Sign bit: unchanged
    - Exponent bit: unchanged
    - Mantissa = + 0.5 + 0.125
    - Answer = 0 01111111 1110...0 = 3FF00000
- Note: exponent = 2^0 = 1 only
  - If exponent is not equal to 1, you need to take this into account

Conversion Example (con't)

- What is the IEEE floating point representation of 0.625_{10}? Hence, what is the representation for 0.875_{10}?
  - Step 1: 0.625 = 3F200000 = 0 01111110 010...0
  - Step 2: 0.875
    - 0.875 = 0.625 + 0.25
    - $= 0.625 + 0.5 * 2^{-1}$
    - $= 1.25 * 2^{-1} + 0.5 * 2^{-1}$
    - Sign bit: same
    - Exponent bit: same
    - Mantissa = + 0.5
    - Answer = 0 01111110 110...0 = 3F600000