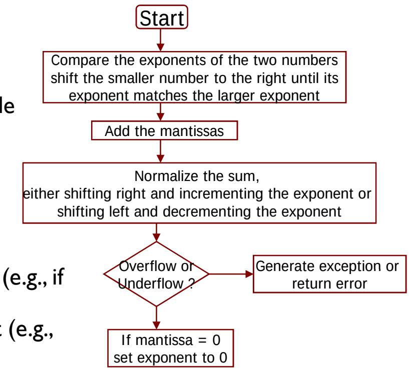
IEEE 754 Floating Point Operations

- 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
    - Overflow/underflow
    - If result mantissa is 0, may need to set the exponent to zero to return a zero.



Flow chart for add operation

Floating Point Addition

- 1.25 + 0.25 (Exponent bits)
  - 0 01111111 010…000 + 0 01111101 000…000 (Sign bit)
  - Steps: (Mantissa bits)
    - 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) (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:
      - No need to change. It is normalized.
    - Check result
      - OK. Answer = 0 01111111 100..000

Floating Point Subtraction

- 1.25 - 0.25
  - 0 01111111 010…000 - 0 01111101 000…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

Multiplication

- Floating Point Operations:
  - $X = (-1)^{Sx}$ Mx * 2^{Ex}, $Y = (-1)^{Sy}$ My * 2^{Ey}
  - 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., 0.001xx…) o
    - Right shift result, increment result exponent (e.g., 10.1xx…)
  - Check result
    - If larger/smaller than maximum exponent allowed return exponent overflow/underflow

Floating Point Multiplication

- -18 * 9.5
  - 1 10000011 0010…000 * 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

Division

- Floating Point Operations:
  - X = $(-1)^{Sx}$ Mx * 2^{Ex}, Y = $(-1)^{Sy}$ My * 2^{Ey}
  - 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 nbr of mantissa bits)
  - Subtract exponents
    - Division: biased exponent (Ex) - biased exponent (Ey) + bias
  - Normalize the result
    - Left shift result, decrement result exponent (e.g., 0.001xx…)
    - Right shift result, increment result exponent (e.g., 10.1xx…)
  - Check result
    - If larger/smaller than maximum exponent allowed return exponent overflow/underflow

Floating Point Division

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

```
      _1 01
11)11.11
   11___
      11
      11
       0 =
```

      - = 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

Conversion Examples:

- Example 1
- 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) => 1.101×2^2
    - 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
- Example 2
- 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