Floating Point Arithmetic

CPE166 Advanced Logic Design

Handout

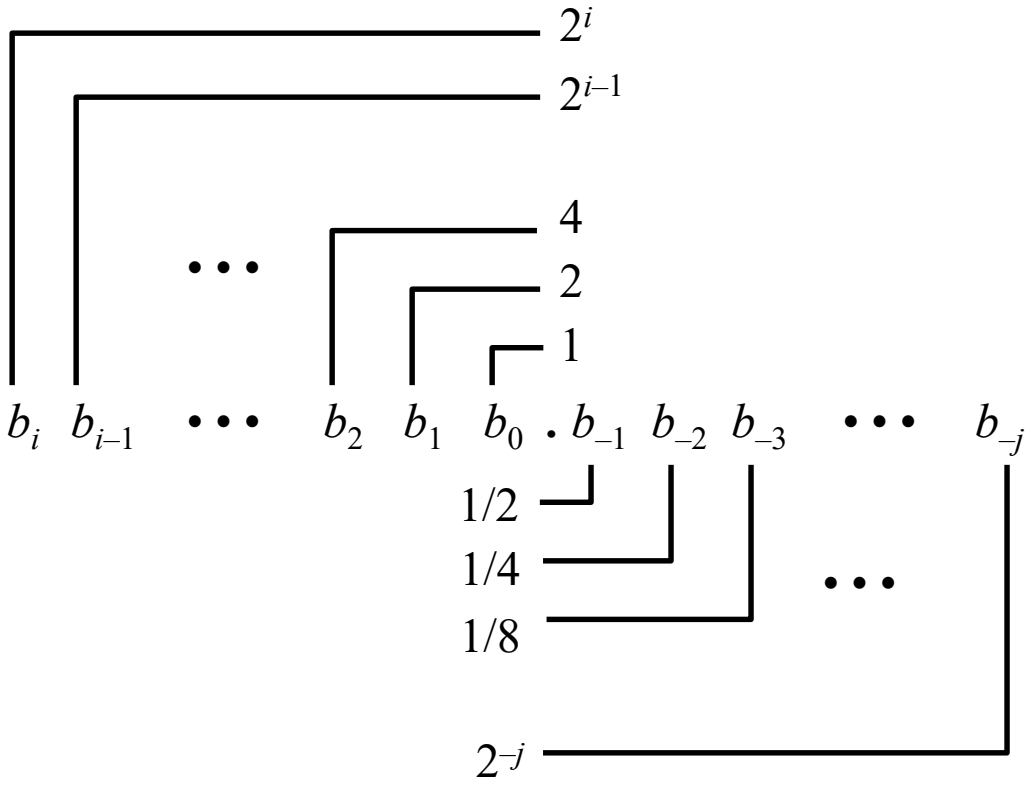
Floating Point Arithmetic

- Topics
  - IEEE Floating Point Standard
  - Rounding
  - Floating Point Operations

IEEE Floating Point

- IEEE Standard 754
  - Established as uniform standard for floating point arithmetic

Fractional Binary Numbers



- Representation
  - Bits to right of “binary point” represent fractional powers of 2
  - Represents rational number: $\sum_{k=-j}^{i} b_k \cdot 2^k$

Fractional Binary Number Examples

- Value Representation

5.75 (5+3/4) 101.11_2

Method1:

```
   2     1     0    -1    -2
1x2 + 0x2 + 1x2 +1x2  +1x2
= 4 + 0 + 1 + 0.5 + 0.25 = 5.75
```

Method2:

$0.11_2 = 11_2$ Shift right by 2 bits $= 3/4_{10} = 0.75_{10}$

$101_2 = 5_{10} \Rightarrow 101.11_2 = 5.75_{10}$

General Fractional Binary Number Examples

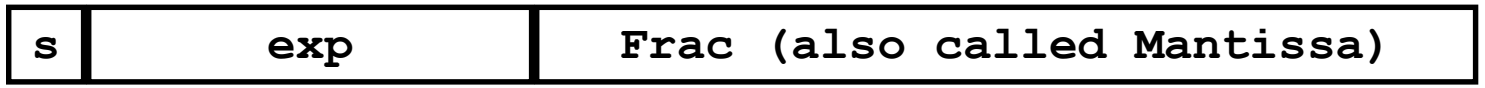
- Value Representation

  5.75 (5+3/4) 101.11_2
  2.785 (2+7/8) 10.111_2
  0.984375 (63/64) 0.111111_2

- Observation
  - Divide by 2 by shifting right
  - Numbers of form $0.111111\ldots_2$ just below 1.0

IEEE 754 Floating Point Representation

- Numerical Form
  - $(-1)^s\ M\ \ 2^E$
    - Sign bit ***s*** determines whether number is negative or positive
    - Significand ***M*** (=1.frac) normally a fractional value in range [1.0,2.0).
    - Exponent ***E*** (=Exp – Bias) weights value by power of two
- Encoding



  - MSB (most significant bit) is sign bit ***S (Sign)***
  - `exp` field encodes *Exponent*
  - `frac` field encodes *Mantissa*
- Sizes
  - Single precision: 8 `exp` bits, 23 `frac` bits
    - 32 bits total
  - Double precision: 11 `exp` bits, 52 `frac` bits
    - 64 bits total

Format Parameters of IEEE 754 Floating Point Standard

Parameter	Format	
	Single Precision	Double Precision
Format width in bits	32	64
Precision (p) = fraction + hidden bit	23 + 1	52 + 1
Exponent width in bits	8	11
Maximum value of exponent (IEEE 754 Biased Format)	+ 127 + 127 = 254	+ 1023 + 1023 = 2046
Minimum value of exponent (IEEE 754 Biased Format)	-126 + 127 = 1	-1022 + 1023 = 1

“Normalized” Numeric Values

- Condition
  - `exp` ≠ `000…0` and `exp` ≠ `111…1`
  - `exp` >= `000…1` and `exp` <= `111…0`

- Exponent coded as *biased* value

  $E = Exp - Bias$

  - *Exp* : **unsigned value** denoted by **`exp`**
  - *Bias* : Bias value
    - Single precision: 127 (*Exp*: 1…254, *E*: -126…127)
    - Double precision: 1023 (*Exp*: 1…2046, *E*: -1022…1023)
    - in general: *Bias* = $2^{n-1} - 1$, where n is the number of exponent bits
- Significand coded with implied leading 1

  $M = 1.xxx…x_2$

  - **`xxx…x`**: bits of `frac`
  - Minimum when **`000…0`** (*M* = 1.0)
  - Maximum when **`111…1`** (*M* = 2.0 – ε)
  - Get extra leading bit for “free”

Normalized Encoding Example

- Value

  `Float F = 15213.0;`

  - $15213_{10} = 11101101101101_2 = 1.1101101101101_2 \times 2^{13}$

- Significand

  M = $1.\underline{1101101101101}_2$

  `frac` = $\underline{1101101101101}0000000000_2$

- Exponent

  E = 13

  Bias = 127

  Exp = 140 = 10001100_2

Example IEEE-decimal conversion

- Let's find the decimal value of the following IEEE number.

  1 01111100 11000000000000000000000

- First convert each individual field to decimal.
  - The sign bit s is 1.
  - The e field contains 01111100 = 124_{10}.
  - The mantissa is 0.11000… = 0.75_{10}.
- Then just plug these decimal values of s, e and f into our formula.

  $$- (1 + f) * 2^{exp-bias}$$

- This gives us $- (1 + 0.75) * 2^{124-127} = (-1.75 * 2^{-3}) = -0.21875$.

Denormalized Values (Underflow)

- Condition
  - `exp` = `000…0`
- Value
  - Exponent value *E* = –*Bias* + 1
  - Significand value *m* = `0.xxx…x`$_2$
    - **xxx…x:** bits of `frac`
- Cases
  - `exp` = `000…0`, `frac` = `000…0`
    - Represents value 0
    - Note that have distinct values +0 and –0
  - `exp` = `000…0`, `frac` ≠ `000…0`
    - Numbers very close to 0.0
    - Lose precision as get smaller
    - “Gradual underflow”

Denormalized Numbers

- To reduce the loss of precision when an underflow occurs, IEEE 754 includes the ability to represent fractions smaller than are possible in the normalized representation, by making the implicit leading digit a 0. Such numbers are called denormal. They enable a gradual loss of precision when the result of an arithmetic operation is not exactly zero but is too close to zero to be represented by a normalized number.
- A denormal number is represented with a biased exponent of all 0 bits, which represents an exponent of −126 in single precision (not −127), or −1022 in double precision (not −1023).

Special Values of Inifity and NaN (not a number)

- Condition
  - `exp` = `111…1`
- Cases
  - `exp` = `111…1`, `frac` = `000…0`
    - Represents value ∞ (infinity)
    - Operation that overflows
    - Both positive and negative
    - E.g., 1.0/0.0 = $+\infty$, -1.0/0.0 = $-\infty$
  - `exp` = `111…1`, `frac` $\neq$ `000…0`
    - Not-a-Number (NaN)
    - Represents case when no numeric value can be determined
    - E.g., $\infty - \infty$

Floating Point Operations

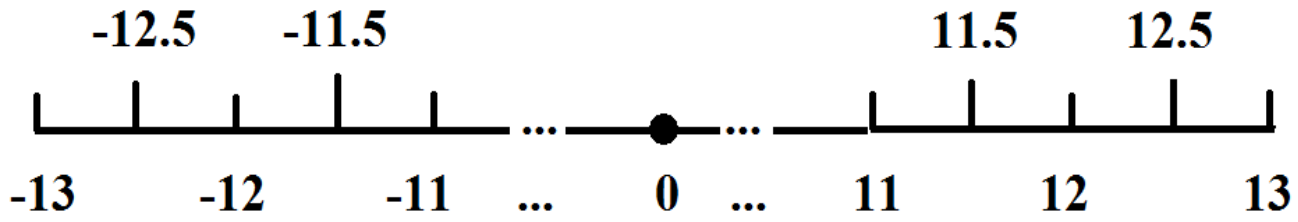
- Conceptual View
  - First compute exact result
  - Make it fit into desired precision
    - Possibly overflow if exponent too large
    - Possibly round to fit into `frac`

Rounding Modes

- Rounding Modes
  - Round toward Zero
  - Round toward -∞
  - Round toward +∞
  - Round toward Nearest, ties to Even
  - Round toward Nearest, ties away from zero

Example of rounding to integers using the IEEE 754 rules

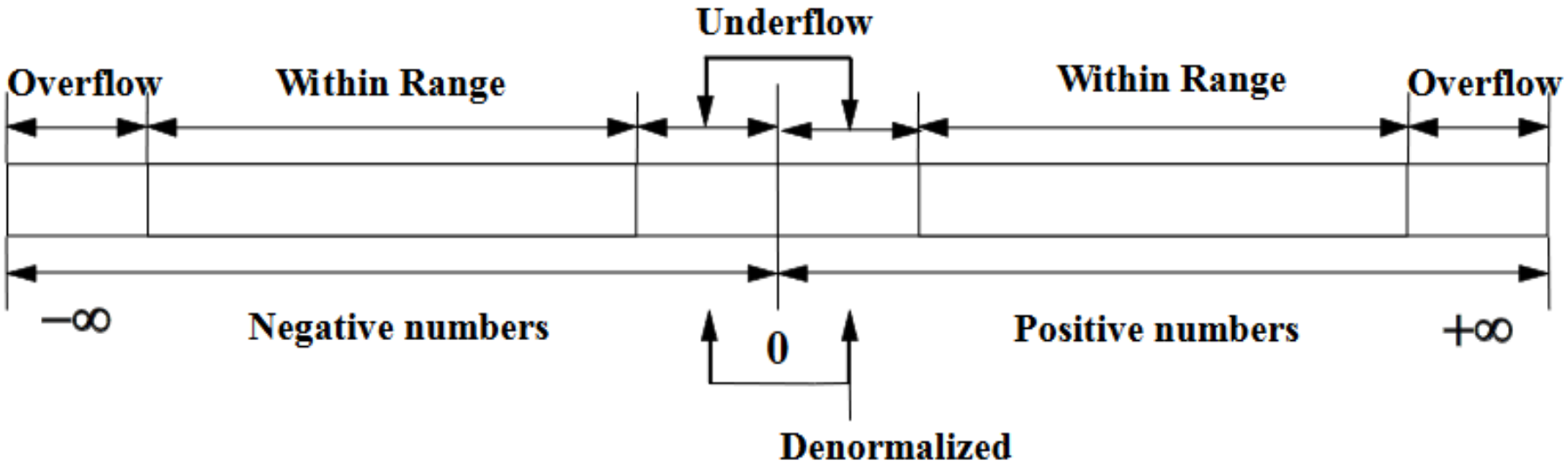
Mode / Example Value	+11.5	+12.5	−11.5	−12.5
to nearest, ties to even	+12.0	+12.0	−12.0	−12.0
to nearest, ties away from zero	+12.0	+13.0	−12.0	−13.0
toward 0	+11.0	+12.0	−11.0	−12.0
toward +∞	+12.0	+13.0	−11.0	−12.0
toward −∞	+11.0	+12.0	−12.0	−13.0



Exceptions in IEEE 754

Exception	Remarks
Overflow	Result can be $\pm \infty$ or default maximum value
Underflow	Result can be 0 or denormal
Divide by Zero	Result can be $\pm \infty$
Invalid	Result is NaN
Inexact	System specified rounding may be required

Range of Floating Point Numbers



Basic Floating Point Multiplication Algorithm

Assuming that the operands are already in the IEEE 754 format, performing floating point multiplication:

$$\text{Result} = R = X * Y = (-1)^{Xs} (Xm \times 2^{Xe}) * (-1)^{Ys} (Ym \times 2^{Ye})$$

involves the following steps:

(1) If one or both operands is equal to zero, return the result as zero, otherwise:

(2) Compute the sign of the result Xs XOR Ys

(3) Compute the mantissa of the result:

- Multiply the mantissas: Xm * Ym
- Round the result to the allowed number of mantissa bits

(4) Compute the exponent of the result:

Result exponent = biased exponent (X) + biased exponent (Y) - bias

(5) Normalize if needed, by shifting mantissa right, incrementing result exponent.

(6) Check result exponent for overflow/underflow:

- If larger than maximum exponent allowed return exponent overflow
- If smaller than minimum exponent allowed return exponent underflow

FP Multiplication

- Operands
  - $(-1)^{s1}\ M1\ \ 2^{E1}$
  - $(-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result
  - $(-1)^{s}\ M\ \ 2^{E}$
    - Sign *s*: *s1* ^ *s2*
    - Significand *M*: *M1* * *M2*
    - Exponent *E*: *E1* + *E2*
- Fixing
  - If $M \geq 2$, shift *M* right, increment *E*
  - If *E* out of range, overflow
  - Round *M* to fit `frac` precision

FP Addition

- Operands
  - $(-1)^{s1}\ M1\ \ 2^{E1}$
  - $(-1)^{s2}\ M2\ \ 2^{E2}$
    - Assume $E1 > E2$
- Exact Result
  - $(-1)^{s}\ M\ \ 2^{E}$
    - Sign s, significand M:
      - Result of signed align & add
    - Exponent E: $E1$
- Fixing
  - If $M \geq 2$, shift M right, increment E
  - if $M < 1$, shift M left k positions, decrement E by k
  - Overflow if E out of range
  - Round M to fit `frac` precision