

4.0 Floating Point (FP)

Basic representation of floating point number

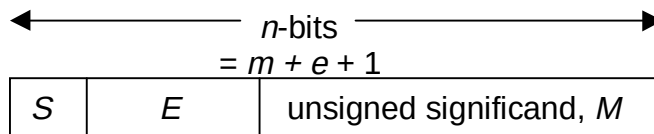
$$F = M \cdot \beta^E$$

where M is Mantissa

β is base of E

E is an exponent

Since β is constant, encoding of F takes form



where $F = (-1)^S \cdot M \cdot \beta^E$ $S \in \{1,0\}$

and $M_{max} = 1 - ulp$ (fractional)

Selecting β

Floating point operations require E adjustment to avoid M overflow, denormalization, and for add/subtract alignment

Note that

$$M \cdot \beta^E = (M/\beta) \cdot (\beta^{E+1}) = \beta M \beta^{E-1}$$

So, we want division and multiplication by β to be shift operations

$$\Rightarrow \beta = 2^k \quad k = 1, 2, \dots$$

Ex.

$\beta = 2$ very common

$\beta = 16$ IBM/370

if $\beta = 2$

$$M2^E = (M/2^3) \cdot 2^{E+3}$$

3-bit shift to right

if $\beta = 16$

$$M \cdot 16^E = (M/16^3) \cdot 16^{E+3}$$

three 4-bit shifts to right

Normalization

Significand left aligned to allow maximum number of significant digits.

Ex.

1 if fraction and $\beta = 2$

↓

$$\begin{array}{ll} \beta = 2 & \Rightarrow 0.00000110 * 2^{50} = 0.110 \times 2^{45} \\ \beta = 16 & \Rightarrow 0.00000110 * 16^{50} = 0.0110 \times 16^{49} \end{array}$$

Note: If two numbers are normalized, comparison is simplified (S and E are examined first).

Normalized Range:

$$M_{min} = 1/\beta \text{ and } M_{max} = 1 - ulp$$

Note: Zero usually represented by $E = 0$ (allows zero test similar to fixed point)

Exponent Representation:

- need + and - values
- want all 0's to be smallest
- want easy comparison of exponents

This is possible with “biased” exponent

$$E = E^{true} + \text{bias}$$

where

E is value (bit pattern) used

E^{true} is true value in two's complement

Bias is constant selected equal to most neg. exp ($\Rightarrow E=0$ for it)

For e -bit E

$$0 \leq E \leq 2^e - 1$$

If $e = 8$

$$0 \leq E \leq 255 \quad \text{“excess 128” and bias} = 2^{8-1} = 128$$

$$-128 \leq E^{true} \leq 127 \text{ or } E^{true} = E - \text{bias}$$

E (biased)	E^{true} (2's compl)	
11111111	01111111	127
⋮	⋮	
10000000	00000000	0
01111111	11111111	-1
⋮	⋮	
00000000	10000000	-128
↑	↑	
only msb reversed		

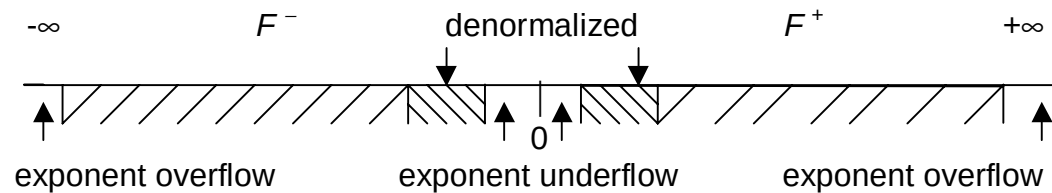
Complete Range:

positive numbers, F^+

$$M_{min} \cdot \beta^{E_{min}} \leq F^+ \leq M_{max} \cdot \beta^{E_{max}}$$

and negative F^-

$$-M_{max} \cdot \beta^{E_{max}} \leq F^- \leq -M_{min} \cdot \beta^{E_{min}}$$



Ex. IBM short F.P (370), $F = (-1)^S \cdot M \cdot 16^{E-64}$

S	E , excess 64	M , unsigned fraction
---	-----------------	-------------------------

1 7 24

$$E_{min} = -64, E_{max} = 63 \quad M_{min} = 16^{-1}, M_{max} = 1 - 16^{-6} = 1 - 2^{-24}$$

$$F_{min}^+ = 16^{-1} \cdot 16^{-64} \cong 5.4 \times 10^{-79}$$

$$F_{max}^+ = 1 - 2^{-24} \cdot 16^{+63} \cong 7.23 \times 10^{75}$$

What would be the “370” representation of –56 in short FP?

Dividing by 16

$$\begin{aligned} - (56/16) \cdot 16^1 &= - (7/2) \cdot 16^1 \\ &= - (7/32) \cdot 16^2 \end{aligned}$$

$$F = (-1) \cdot M \cdot 16^2$$

$$\begin{aligned} \Rightarrow S=1, E &= E^{true} + 64, M = 7/32 = 0.001110..0 \\ E &= 2 + 64 = 66 \\ E &= 01000010 \end{aligned}$$

$$(S,E,M) = \underline{1 \ 1000010 \ 001110000000000000000000}$$

$$(S,E,M) = (C2380000)_{16}$$

What is “370” short FP precision?

$$\text{Ans: } ulp = 16^{-6} = 2^{-24} \cong 0.6 \times 10^{-7} \text{ (about 7 significant decimal digits)}$$

see Tables 4.1 and 4.4

Hidden Bit

For base 2, normalized value always has leading 1

$\Rightarrow$ it can be eliminated & precision increased by 1 bit (ex. from $2^{-23} \rightarrow 2^{-24}$)

But, $M = 0 \Rightarrow 0.100\dots$ since the one is implied.

So, what is $E=0, M=0$?

$$0 \text{ or } 0.1 \cdot 2^{0-128} = 2^{-129} ?$$

Solution: only use $E = 0$ if $F = 0$

$$\text{but now } F_{\min}^+ = 1/2 \cdot 2^{1-128} = 2^{-128}$$

4.2 Floating Point Operations

Assumptions:

Significands are normalized fractions ($S-M$), exponents are biased

$$\left. \begin{aligned} F_1 &= (-1)^{S_1} M_1 \cdot \beta^{E_1 - \text{bias}} \\ F_2 &= (-1)^{S_2} M_2 \cdot \beta^{E_2 - \text{bias}} \end{aligned} \right\} \text{operands}$$

$$F_3 = (-1)^{S_3} M_3 \cdot \beta^{E_3 - \text{bias}} \quad \text{result}$$

Multiplication:

- 1.) $S_3 = S_1 \oplus S_2$
 $E_3 = E_1 + E_2 - \text{bias}$
 Note : $E^{\text{true}} = E - \text{bias}$
 (think of $E^{\text{true}} = 0$ and $E = \text{bias}$)
 for $\text{bias} = 2^{e-1}$,
 only need to complement msb
 $M_3 = M_1 \cdot M_2$

can be done in parallel
- 2.a) If $E_3 > E_{\max} \Rightarrow \text{overflow}$
 $F_3 = + \text{ or } - \infty$ for some FPU's
- 2.b) If $E_3 < E_{\min}$ can occur for $E_3 < 0 \Rightarrow \text{underflow}$
 $F_3 = 0$ for some FPU's
- 3.) If necessary, post-normalize
 since $1/\beta \leq M_i < 1$ ($i = 1, 2$)
 $1/\beta^2 \leq M_1 M_2 < 1$
 use base- β left shift and $E_3 \leftarrow E_3 - 1$
- 4.) If $E_3 < E_{\min} \Rightarrow \text{underflow}$

Division: (F_1 / F_2)

$$1.) \quad S_3 = S_1 \oplus S_2$$

$$E_3 = E_1 - E_2 + \text{bias}$$

$$M_3 = M_1 / M_2$$

$$2.a) \quad \text{If } E_3 > E_{\max} \Rightarrow \text{overflow } (\pm \infty)$$

$$2.b) \quad \text{If } E_3 < E_{\min} \Rightarrow \text{underflow}$$

$$3.) \quad \text{If necessary post-normalize.} \\ \text{base-}\beta \text{ shift to right and } E_3 \leftarrow E_3 + 1$$

$$4.) \quad \text{If } E_3 > E_{\max} \Rightarrow \text{overflow } (\pm \infty)$$

Note: if $F_1 = F_2 = 0 \Rightarrow F_3$ is NAN (or $F_1 = 0, F_2 = \infty$)

Addition/subtraction :

$$1.) \quad \text{If } E_1 \neq E_2, \text{ align significands} \\ \text{If } E_1 > E_2 \quad M_2 \text{ shifted } |E_1 - E_2| \text{ base-}\beta \text{ positions to right} \\ \text{so, } E_3 = E_1$$

$$2.) \quad \text{perform operations}$$

add	subtract
$M_3 = M_1 + M_2$	$M_3 = M_1 - M_2$
$1/\beta \leq M_3 < 2$	$0 \leq M_3 < 1$

$$3.) \quad \text{post-normalization}$$

add	subtract
if $M_3 > 1$ shift M_3 to right and $E_3 \leftarrow E_3 + 1$	if $M < 1/\beta$ shift M_3 to left as many times as necessary and decrease E_3 by same number.

$$4.) \quad \text{If } E_3 > E_{\max} \Rightarrow \text{overflow}$$

$$\text{if } E_3 < E_{\min} \Rightarrow \text{underflow}$$

Ex.

$F_1 - F_2$ where

$$F_1 = (0.100000)_{16} \cdot 16^3$$

$$F_2 = (0.FFFFFF)_{16} \cdot 16^2$$

$E_1 - E_2 = 1$, M_2 shift right 1 position

F_1	0.100000	↓	.16 ³	
F_2 aligned	0.0FFFFFF(F)	—	.16 ³	
$F_1 - F_2$	0.000001		.16 ³	
Post-norm	0.100000		.16 ⁻²	← wrong, 16 ⁻³ with unlimited bits

↑
lost digit.

 needed additional "guard bits"

4.3 Floating-Point Format Selection

Let x = real number

$Fl(x)$ = machine representation of x

We call

$Fl(x) - x$ the *absolute representation error*.

Let the two closest machine values to x be

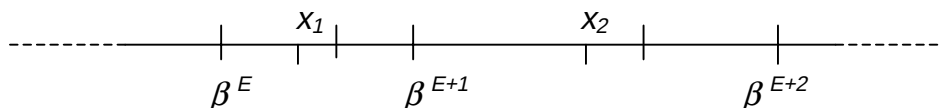
$$F_1 \leq x \leq F_2$$

If $F_1 = M \beta^E$ and $F_2 = (M + ulp) \beta^E$

the maximum absolute error is

$$(F_1 - F_2)/2 = (ulp/2) \beta^E$$

Note: error varies with E



Relative Representation Error

Defined as

$$\delta(x) = (Fl(x) - x)/x$$

Maximum relative representation error (MRRE) is upper bound of δ

$$\delta(x) \leq \frac{1/2 ulp \beta^E}{M \beta^E} = \frac{1}{2} \frac{ulp}{M} \leq \frac{1}{2} \frac{ulp}{1/\beta} = \frac{1}{2} ulp \beta$$

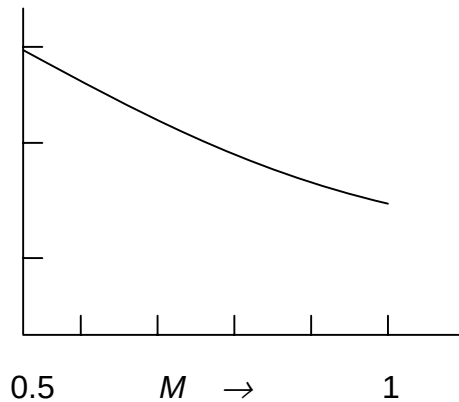
$$MRRE = (1/2) ulp \cdot \beta$$

$\Rightarrow$ lower β and more bits help reduce the error

Alternate Measure Based on Floating-Point Distributions

Observed density functions: $1/(M \cdot \ln \beta)$; $1/\beta \leq M \leq 1$

Ex. $\beta = 2$



Define *average real representation error* (ARRE) as
1/2 MRRE compounded with $1/(M \cdot \ln \beta)$ distribution
or

$$ARRE = \int_{1/\beta}^1 \frac{1}{M \ln \beta} \frac{ulp}{4M} dM = \frac{\beta - 1}{\ln \beta} \frac{ulp}{4}$$

See table 4.2 for examples.

$\Rightarrow \beta = 2$ best, but larger β 's can cut down on alignment.

4.4 IEEE Floating-Point Standard (IEEE 754)

Single Precision

(Double Precision)

- $\beta = 2, e = 8, m = 23$
- significand normalized to **1.f**

($e = 11, m = 52$)

$$F = (-1)^S 1.f \times 2^{E-127}$$

$$(F = (-1)^S 1.f \times 2^{E-1023})$$

↑

$2^{e-1} - 1$ (not 2^{e-1}) gives more range & 1.f also helps

- $1 \leq E \leq 254$

($1 \leq E \leq 2046$)

signed zeroes: $E = f = 0$

signed infinities: $E = E_{max}, f = 0$

NANS: $E = E_{max}, f \neq 0$

- Denormalized numbers - allows graceful overflow and smaller representations

$E = 0, f \neq 0$

Denormalized valuation: $F = (-1)^S 0.f \times 2^{-126}$ ($F = (-1)^S 0.f \times 2^{-1022}$)

4.5 Round-Off Schemes

- Need to dispose of extra digits to get final result
- Accuracy/speed trade-off

Let x and y be real numbers

and Fl be floating point machine representation

$Fl(x)$ be floating point machine representation of x

Must satisfy conditions:

1. $Fl(x) \leq Fl(y)$ for $x \leq y$
2. If $x \in Fl$ then $Fl(x) = x$
3. If F_1 and F_2 are consecutive in Fl where $F_1 \leq x \leq F_2$ then either $Fl(x) = F_1$ or $Fl(x) = F_2$

Assume (for convenience) our numbers look like this:

$$\frac{\text{XX} \dots \text{X}}{m} . \frac{\text{XXX}}{d}$$

↑
radix point

Truncation

Remove d extra digits and leave m remaining digits unchanged

Ex. $Trunc(2.99) = 2$

Advantage: fast

Disadvantage: numerical performance poor (error larger than necessary)

See Fig 4.1

Let $\text{Error} = \text{Trunc}(x) - x$

If $d = 2, r = 2$

No.	Trunc(x)	Error
X.00	X	0
X.01	X	-1/4
X.10	X	-1/2
X.11	X	-3/4

$$\text{Average Error (uniform distribution)} = \frac{0 + (-1/4) + (-1/2) + (-3/4)}{4}$$

$$= (-6/4)/4 = -3/8 \leftarrow \text{negative bias}$$

If $d = 3, r = 2$

$$Average \ Error = \frac{-\sum_{i=0}^{2^d-1} i}{2^d \cdot 2^d} = -\frac{28}{64} = -\frac{7}{16}$$

Implies for larger d , average error $\approx -1/2$

Round-to-Nearest

Add $1/2$ *ulp* and retain integer part

$$\text{Round}(x) = \lfloor x + 1/2 \text{ ulp} \rfloor$$

Ex. $\text{Round}(2.49) = \lfloor 2.49 + 0.5 \rfloor = \lfloor 2.99 \rfloor = 2$
 $\text{Round}(2.50) = \lfloor 2.50 + 0.5 \rfloor = \lfloor 3.00 \rfloor = 3$

See Fig 4.2

If $d = 2, r = 2$, $\text{Error} = \text{Round}(x) - x$

No.	$\text{Round}(x)$	Error
X.00	X	0
X.01	X	-1/4
X.10	X + 1	+1/2
X.11	X + 1	+1/4

$$\text{Average Error} = (0 - 1/4 + 1/2 + 1/4)/4 = 1/8 \text{ positive bias}$$

or in general $(1/2) 2^{-d}$. Note, only a function of d .

Positive bias due to always rounding up on X.1000..0.

Round-to-Nearest-Alternating

Add $1/2$ *ulp* if least significant digit even (round up)

Ex.

$$\text{Roundn}(011.1)_2 = (011)_2 \Rightarrow \text{error} = -1/2$$

$$\text{Roundn}(010.1)_2 = (011)_2 \Rightarrow \text{error} = +1/2$$

$$\text{Average Error over two groups of size } 2^d = 0$$

ROM-Rounding

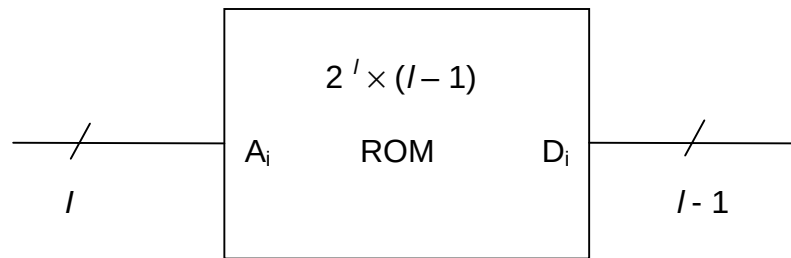
Use ROM look-up with l address lines for rounding and limit carry ripples to $l-1$ least significant bit (binary) by rounding down for case of $l-1$ ones

Ex. $l = 3$

$ROM(X10.1) = X11$ (+1/2 error)

$ROM(X11.1) = X11$ (-1/2 error).

Implementation:



For round-to-nearest

$$\begin{aligned} \text{Average Bias} &= 1/2((1/2)^d - (1/2)^{l-1}) \\ &= 1/2 (1/2)^d \text{ for large } l \end{aligned}$$

For round-to-nearest-even

$\text{Average Bias} \Rightarrow 0$ for large l

Contrasting round-to-nearest-even with ROM

<u>Round-to-nearest-even</u>			<u>ROM</u>		
add operation			lookup over $l-1$ bits		
001.1	→	010	001.1	→	010
010.1	→	010	010.1	→	011
011.1	→	100	011.1	→	011

Guard Digits

Want to know how many extra bits are needed for rounding and post-normalization

Division: none for post-norm

$$1/2 \leq M_1/M_2 < 2, \beta = 2$$

only a right shift possibly required

Multiplication: One guard digit for post-normalization

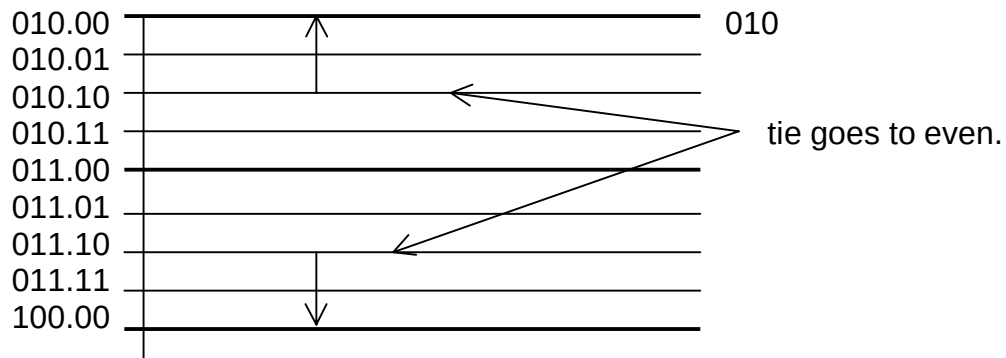
$$1/4 \leq M_1 \times M_2 < 1, \beta = 2$$

1-bit left shift possible

So, with rounding (if we use rounding-to-nearest) we need one more guard bit

- We call them G (*guard*) and R (*round*)

For Round-to-nearest-even will need one more to break tie



- Will call it S (*sticky*) bit = 1 if any non-zero additional trailing bits exists

$\Rightarrow$ so G , R and S are sufficient for $\times$ and $\div$

With round-to-nearest will also need R

Ex. 3

A	0.1 0 0 0 0 0 1 0 1 1 0 0	0 0 0 0 0 0
B (Aligned)	0.0 0 0 0 0 0 1 1 0 0 0 0	0 1 0 1 1 0
$A - B$	0.0 1 1 1 1 1 1 1 0 1 1	1 0 1 0 1 0
Post-norm	0.1 1 1 1 1 1 1 0 1 1 1	0

Would have been one if only G and S bits used

So, round $\neq$ sticky bit.
 $\Rightarrow$ need R

		G	R	S
A	0.1 0 0 0 0 0 1 0 1 1 0 0	0	0	0
B (Aligned)	0.0 0 0 0 0 0 1 1 0 0 0 0	0	1	1
$A - B$	0.0 1 1 1 1 1 1 1 0 1 1	1	0	1

If no post-normalization required we make

$$S \leftarrow R + S$$

$$R \leftarrow G$$

For round-to-nearest-even add ulp if

$$R \cdot S + R \cdot \bar{S} \cdot L = R \cdot (S + L)$$

$\uparrow$
 least significant bit.

Ex. 4

No postnormalization needed

A	0.1 0 0 0 0 1 0 1 0 1 0 0	
B	0.1 1 0 0 0 0 0 1 0 0 0 1	$E_A - E_B = 6$
		G R S
A	0.1 0 0 0 0 1 0 1 0 1 0 0	0 0 0
B aligned	0.0 0 0 0 0 0 1 1 0 0 0 0	0 1 1
$A - B$	0.1 0 0 0 0 0 1 0 0 0 1 1	1 0 1
		R S
After-rounded	0.1 0 0 0 0 0 1 0 0 0 1 1	1 1
-to-nearest	0.1 0 0 0 0 0 1 0 0 1 0 0	

$\uparrow$ $R \cdot S = 1 \Rightarrow +ulp$

Note : can avoid extra delay due to rounding by adding 1 to L at same time significands are added $\Rightarrow$ two adders are needed since least significant bit can be in two possible positions (postnormalization may be required)

Round-off errors

Let relative round-off error be

$$\varepsilon = (Fl(x * y) - (x * y)) / (x * y)$$

where $*$ is a floating point operation (+, -, $\times$, $\div$)

Rearranging

$$Fl(x * y) = (x * y)(1 + \varepsilon)$$

For truncation

$$|\varepsilon_{\text{trunc}}| \leq (\beta^{-M})(1/\beta) = \beta^{-M+1}$$

For round-to-nearest

$$|\varepsilon_{\text{Round}}| \leq (1/2 \beta^{-M})(1/\beta) = 1/2 \beta^{-M+1}$$

Effect of operations

Addition:

For two machine values

$$A_1 = \underbrace{A_1^c(1 + \varepsilon_1)}_{\text{correct values}} \text{ and } A_2 = \underbrace{A_2^c(1 + \varepsilon_2)}_{\text{correct values}}$$

$$\varepsilon = \frac{A_1 + A_2 - (A_1^c + A_2^c)}{A_1^c + A_2^c}$$

$$\varepsilon = \frac{A_1^c(1 + \varepsilon_1) + A_2^c(1 + \varepsilon_2) - (A_1^c + A_2^c)}{A_1^c + A_2^c}$$

$$\varepsilon = \frac{A_1^c \varepsilon_1 + A_2^c \varepsilon_2}{A_1^c + A_2^c}$$

This implies that ε_i of large A_i will dominate

Subtraction:

$$\varepsilon = \frac{A_1^c \varepsilon_1 - A_2^c \varepsilon_2}{A_1^c - A_2^c}$$

$\Rightarrow$ severe error if $A_1^c - A_2^c$ is small

Multiplication:

$$\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_1 \varepsilon_2$$

Division:

$$\varepsilon = (\varepsilon_1 - \varepsilon_2)/(1 + \varepsilon_2)$$